Neural Operators for Learning Complex Nonlocal Mappings in Fluid Dynamics

Xu-Hui Zhou

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Aerospace Engineering

Heng Xiao, Chair
Christine M. Gilbert
Eric G. Paterson
Christopher J. Roy

September 19, 2024
Blacksburg, Virginia

Keywords: Neural operator, Turbulence modeling, Numerical simulation acceleration, Data assimilation

Neural Operators for Learning Complex Nonlocal Mappings in Fluid Dynamics

Xu-Hui Zhou

(ABSTRACT)

Accurate physical modeling and accelerated numerical simulation of turbulent flows remain primary challenges in CFD for aerospace engineering and related fields. This dissertation tackles these challenges with a focus on Reynolds-Averaged Navier–Stokes (RANS) models, which will continue to serve as the backbone for many practical aircraft applications. Specifically, in RANS turbulence modeling, the challenges include developing more efficient ensemble filters to learn nonlinear eddy viscosity models from observation data that move beyond the classical Boussinesq hypothesis, as well as developing non-equilibrium models that break away from the weak equilibrium assumption while maintaining computational efficiency. For accelerating RANS simulations, the challenges include leveraging existing simulation data to optimize the computational workflow while maintaining the method's adaptability to various computational settings. From a fundamental and mathematical perspective, we view these challenges as problems of modeling and learning complex nonlinear and nonlocal mappings, which we categorize into three types: field-to-point, field-to-field, and ensemble-to-ensemble. To model and resolve these mappings, we build up on recent advancements in machine learning and develop novel neural operator-based methods that not only possess strong representational capabilities but also preserve critical physical and mathematical principles. With the developed tools, we have demonstrated promising preliminary results in addressing these challenges and have the potential to significantly advance the state of the art in RANS turbulence modeling and simulation acceleration.

Neural Operators for Learning Complex Nonlocal Mappings in Fluid Dynamics

Xu-Hui Zhou

(GENERAL AUDIENCE ABSTRACT)

Understanding and accurately predicting turbulent flows, such as those around airplanes or ships, are among the biggest challenges in computational fluid dynamics (CFD). This research aims to improve Reynolds-Averaged Navier–Stokes (RANS) models, which are widely used in practical engineering applications. Traditional RANS turbulence models are based on simplified assumptions that are linear and local, making it difficult to capture the true complexity of turbulent flows. My work addresses this limitation by developing new models that leverage advanced machine learning techniques to better represent turbulence. Specifically, I have focused on developing methods that extend beyond conventional approaches by learning more accurate local nonlinear constitutive relations and incorporating nonlocal effects—an important step toward improving simulation accuracy. In addition, I have explored strategies to accelerate RANS simulations by making more effective use of existing data, providing better initial conditions for simulations, and ultimately reducing computational costs. Preliminary results indicate that these new methods have the potential to push the boundaries of RANS turbulence modeling, enabling more accurate and efficient simulations.

Dedication

To my family, for their unwavering support and encouragement.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Prof. Heng Xiao, for his unwavering support, mentorship, and guidance throughout my Ph.D. journey. His insights, encouragement, and dedication to my development as a researcher have been invaluable. I am profoundly grateful for the opportunities he provided for me to explore and expand my academic and professional horizons. I am also sincerely thankful to my collaborators, Prof. Qi Li, Prof. Cheng Chen, Prof. James McClure, and Dr. Yanle Lu, for their significant contributions and the enriching exchange of ideas that greatly benefited my research. I owe special thanks to Dr. Jiequn Han, whose expertise in applied mathematics and passion for research have been a true source of inspiration. I am deeply appreciative of my committee members for their guidance and support. I am particularly grateful to Prof. Christopher Roy, who provided steady mentorship and support following Prof. Xiao's transition to Germany, ensuring I remained on track. I also extend my heartfelt thanks to Prof. Christine Gilbert for her dedication to my progress, especially for attending my pre-defense in person during her sabbatical. I would like to acknowledge my office mates, Dr. Xuning Zhao and Dr. Wentao Ma, for their friendship and support, which have been a constant source of strength over the past four years. Their camaraderie, especially during the challenging times of the COVID-19 pandemic, helped me maintain focus and balance in both work and life. I am equally grateful to the other Ph.D. students in our group for their thoughtful discussions and shared experiences that made this journey all the more meaningful. Finally, I am profoundly thankful to my family for their unwavering support and encouragement. Although circumstances have prevented me from returning home, their faith in me has been my anchor throughout this journey. Their love and belief in my abilities have given me the strength to persevere, and for that, I am forever grateful.

Part of this material is based on research sponsored by the U.S. Air Force under agreement number FA865019-2-2204. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

Contents

Chapter 1

Introduction

Computational fluid dynamics (CFD) in aerospace engineering faces persistent challenges in improving the accuracy, efficiency, and robustness of simulations [1]. Among these challenges, two primary obstacles are the accurate modeling of turbulent flows and the efficient acceleration of their numerical simulations with desired fidelity. From a fundamental and mathematical perspective, addressing these challenges involves modeling and identifying complex mappings that are inherently nonlinear and nonlocal, making them either difficult to capture or computationally expensive and time-consuming to resolve with traditional approaches. Recent advancements in machine learning have shown promise in overcoming these limitations [2, 3], leveraging data-driven approaches to learn complex multi-scale dependencies. This dissertation builds upon these developments by introducing novel neural operator-based methods to model and learn these complex mappings, providing a new perspective on tackling key challenges in CFD.

1.1 Challenges of CFD in Aerospace Engineering

This section expands on the two primary challenges highlighted above: (1) accurately modeling turbulent flows and (2) developing fast-converging and robust computational methods for numerical simulations. The focus of this dissertation is on improving Reynolds-averaged Navier–Stokes (RANS) models for modeling turbulent flows through the development of advanced turbulence models and accelerating RANS simulations while maintaining the desired level of computational accuracy. Although high-fidelity methods such as direct numerical simulation (DNS) [4], large eddy simulation (LES) [5], and hybrid RANS/LES[6] provide detailed insights into fluid flow physics, their substantial computational costs make them impractical for routine use in many aircraft applications. In scenarios where only mean flow characteristics are of interest, such as in aerodynamic design and optimization, RANS models remain the preferred choice due to their computational efficiency [7]. As emphasized in NASA's CFD Vision 2030 Study [8], RANS models will continue to be indispensable for a broad range of aircraft applications, particularly in "outer-loop" processes that require large numbers of simulations, including design optimization, load prediction, and uncertainty quantification [9].

1.1.1 Challenges of RANS Closure Modeling

To model the mean flow motion, the RANS equations are derived by decomposing the instantaneous velocity and pressure fields into their mean and fluctuating components, followed by applying an ensemble-average to the Navier–Stokes equations. The resulting incompressible,

steady-state RANS equations for the mean velocity $\overline{\boldsymbol{u}}$ and pressure $\overline{p}$ are expressed as:

$$\begin{aligned} \boldsymbol{\nabla} \cdot \overline{\boldsymbol{u}} &= 0, \\ \overline{\boldsymbol{u}} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{u}} - \nu \boldsymbol{\nabla}^2 \overline{\boldsymbol{u}} + \frac{1}{\rho} \boldsymbol{\nabla} \overline{p} - \boldsymbol{\nabla} \cdot \boldsymbol{\tau} &= 0, \end{aligned} \tag{1.1}$$

where ν is the kinematic viscosity of the fluid, ρ is the fluid density, and $\boldsymbol{\tau}$ represents the unclosed Reynolds stress tensor. To solve the RANS equations, it is necessary to close the system by employing a turbulence model that models the Reynolds stress tensor $\boldsymbol{\tau}$ as a function of the mean velocity field $\overline{\boldsymbol{u}}$. This is the fundamental problem of turbulence, which aims to develop accurate turbulence models to represent the additional unknowns introduced by Reynolds stresses [10].

A widely used approach to close the RANS equations is the linear eddy viscosity models, including one-equation models like the Spalart–Allmaras (S-A) model [11] and two-equation models such as k-ε model and k-ω model [10, 12]. These turbulence models are based on two fundamental hypotheses: (1) the Boussinesq assumption, which states that the Reynolds stress tensor is proportional to the trace-less mean strain rate tensor, and (2) the weak equilibrium assumption, which neglects the transport of Reynolds stress anisotropy by assuming that the transport of Reynolds stress is proportional to that of turbulent kinetic energy [13]. While these assumptions significantly simplify the modeling of Reynolds stresses and reduce computational costs, they also constrain the predictive capabilities of linear eddy viscosity models. Specifically, the Boussinesq assumption overlooks the anisotropic nature of turbulence by assuming that the turbulent viscosity is isotropic, implying that the ratio between Reynolds stresses and the mean strain rate is uniform in all directions. This oversimplification makes it challenging for linear eddy viscosity models to accurately capture flow physics in regions with strong curvature, separation, or secondary motions [14]. Furthermore, the weak equilibrium assumption disregards the transport effects of Reynolds stress anisotropy and its nonlocal interactions with large-scale flow structures, resulting in inaccuracies when predicting adverse pressure gradient flows and flows under rapidly changing conditions [15, 16]. Therefore, advancing linear eddy viscosity models for more accurate turbulence modeling fundamentally requires breaking away from these two underlying assumptions.

Challenges of developing nonlinear eddy viscosity models. Over the past decade, substantial research efforts have focused on developing nonlinear eddy viscosity models to address the limitations of the Boussinesq assumption [2]. These models introduce higher-order terms and establish nonlinear constitutive relations between the Reynolds stress anisotropy and the mean strain rate and rotation rate tensors, providing a more appropriate representation of the Reynolds stress tensor without the need for additional differential equations [10]. The nonlinear constitutive relations can be efficiently obtained through data-driven training, enabled by the growing availability of high-quality turbulent flow datasets (e.g., [17, 18, 19]) and the rapid advancements in machine learning methodologies [20]. The foundation of this line of research is the development of the tensor basis neural network (TBNN) [21],

which provides a general, complete, and frame-independent representation of Reynolds stress anisotropy [13]. The TBNN utilizes tensor invariants and polynomial expansions to establish a comprehensive basis for representing the Reynolds stress anisotropy in terms of the mean flow variables. By leveraging the symmetry and rotational properties of tensors, TBNN provides a physically consistent mapping that aligns well with the anisotropic nature of turbulence. The most straightforward approach for training TBNN is to utilize high-fidelity datasets through the backpropagation algorithm. For instance, DNS data from plane channel and square duct flows at various friction Reynolds numbers [22, 23, 24, 25] have been used to train TBNN models effectively [26]. Building on the TBNN framework, other studies have explored alternative formulations, such as replacing neural networks with random forest regression for robustness [27], using symbolic regression for interpretability [14, 28, 29], and applying sparse Bayesian learning to optimize tensor polynomial representations [30, 31, 32].

However, the above studies for learning nonlinear constitutive models typically require extensive training data, including the direct data of Reynolds stresses. This dependency presents significant limitations, as obtaining high-fidelity datasets under various flow conditions is resource-intensive and often impractical for real-world aircraft applications [33]. As a result, some recent research has focused on reducing data dependency by utilizing sparse indirect data sources, such as velocity measurements or aerodynamic force coefficients, for model training. One particularly promising approach is the use of ensemble Kalman inversion (EKI) [34, 35], which iteratively infers the TBNN-based turbulence model to minimize the discrepancy between the simulated and observed velocity data. By leveraging the solver's propagation, EKI effectively approximates the covariance between the TBNN weights and the sparse velocity measurements, thereby eliminating the need for direct Reynolds stress data or the development of adjoint solvers [35]. This approach has demonstrated robust performance and strong generalizability across several benchmark turbulent flows, such as square duct and periodic hill flows. Building upon this framework, the subsequent work has further enhanced the learning accuracy of turbulence models by combining sparse indirect measurements with limited direct Reynolds stress data [36]. This integrated approach capitalizes on the complementary strengths of both data types, significantly reducing uncertainty and improving predictive accuracy through an effective regularization strategy.

Despite the advancements, this method still faces significant challenges when applied to complex three-dimensional turbulent flows. This limitation arises from the fundamental nature of the ensemble Kalman filter (EnKF), which approximates the covariance matrix using a finite ensemble of samples [37]. The covariance matrix reflects how variations in the TBNN weights influence the observed quantities. Accurately capturing these relationships in complex three-dimensional flow fields often requires a large number of ensemble members, each corresponding to a separate RANS simulation, leading to a substantial increase in computational time. Furthermore, EnKF tends to perform more effectively in linear or weakly nonlinear cases [38], where the parameter-observation relationships are relatively straightforward. In high-Reynolds-number flows or flows over complex geometries [39, 40], where the covariance relationships exhibit strong nonlinearity, achieving accurate representation

requires an even larger ensemble size. This further exacerbates computational costs, making the learning of turbulence models particularly slow and resource-intensive. Thus, the key challenge is to develop more efficient filtering (i.e., data assimilation) algorithms that can better infer the underlying turbulence models from sparse indirect measurements while maintaining computational feasibility. Addressing this challenge is crucial for extending the applicability of the method to realistic, complex engineering scenarios and advancing the state-of-the-art in RANS turbulence modeling.

Challenges of developing non-equilibrium models. Despite the substantial advancements made in nonlinear eddy viscosity models over the past decade, these models still rely on the weak equilibrium assumption, which neglects the transport effects of Reynolds stress anisotropy. While this assumption simplifies the formulation and reduces computational costs, it inherently limits the models' ability to capture complex flow behaviors, especially in regions with rapid changes or strong curvature effects [13]. As highlighted in NASA's CFD Vision 2030 Study [8], advancing the development of non-equilibrium turbulence models that move beyond the weak equilibrium assumption is essential for achieving higher predictive accuracy and extending the applicability to more practical engineering scenarios.

Reynolds stress transport model (RSTM) is a non-equilibrium turbulence model that solves additional transport equations for each component of the Reynolds stress tensor, allowing for the direct modeling of Reynolds stresses in turbulent flows [13]. Unlike eddy viscosity models, which establish a constitutive relationship between the Reynolds stress and the local mean strain rate (and/or rotation rate), RSTM accounts for the anisotropic nature of turbulence and the nonlocal effects of turbulent transport. Classic RSTM formulations, such as the Launder–Reece–Rodi (LRR) model and the Speziale–Sarkar–Gatski (SSG) model [41, 42], have significantly improved the modeling of pressure-strain correlation and turbulent dissipation, enabling RSTM to better capture complex flow features such as strong streamline curvature and flow separation over curved surfaces. As a result, RSTM has demonstrated enhanced predictive capability in low-pressure turbine flows and other engineering applications involving swirling or corner flows [8, 43]. However, RSTMs present several inherent limitations that constrain their broader adoption in industrial and engineering applications [8, 10, 13]. The primary limitation is their increased computational complexity and numerical stiffness, often necessitating advanced solvers to maintain stability and accuracy, which makes them less practical for routine use in aircraft applications. Another critical issue is RSTM's dependency on closure models for terms such as the pressure-strain rate correlation, which governs energy redistribution and anisotropy in turbulent flows. Modeling these terms accurately is particularly challenging in complex scenarios with strong non-equilibrium effects. Additionally, the lack of a unified RSTM formulation and inconsistencies across different flow conditions further complicate their application, while their reduced robustness in capturing separated flows limits their effectiveness in complex configurations.

To address these challenges, data-driven modeling has been explored as a means to enhance

the predictive accuracy of RSTMs, though progress in this area remains very limited. Recent studies have developed neural network-based models to predict the pressure-strain correlation in turbulent channel flows using DNS data, achieving significantly improved alignment with DNS targets compared to traditional models [44, 45]. Despite this progress, the predictions still exhibit notable issues, such as unsmooth and oscillatory profiles near the channel center and wall regions, which raise concerns regarding their robustness and numerical stability when incorporated into RSTM frameworks. Thus, the key challenge is to develop non-equilibrium models that effectively capture the transport physics implied by RSTMs while maintaining robustness and computational efficiency in complex turbulent flow simulations.

1.1.2 Challenges of RANS Simulation Acceleration

RANS-based computational models are, and will continue to be, extensively utilized in aircraft engineering over the coming decade for applications such as aerodynamic design optimization, load prediction, and flow control analysis, owing to their efficiency in simulating mean flow fields and cost-effectiveness compared to higher-fidelity methods [1, 7]. However, accelerating RANS simulations remains essential when high computational costs and long convergence times are present. For instance, a high-resolution RANS simulation for a complete aircraft configuration can take several hours to days, depending on the flow conditions, geometry complexity, and available computational resources [46]. When RANS simulations are integrated into optimization frameworks, the computational burden increases significantly. For example, aerodynamic shape optimization of complex wing-body configurations using high-fidelity RANS simulations often requires millions of computational cells and up to a few hundred design variables, translating into thousands of computational hours [47]. This highlights the need for advanced acceleration strategies to reduce turnaround time and make RANS-based optimization more feasible for industrial applications.

The acceleration of RANS simulations can be addressed at various stages of the computational process, including optimizing simulation initialization, improving iterative solvers, and leveraging hardware-based strategies. Recent developments in iterative solvers have focused on enhancing convergence rates and stability, particularly for RANS equations coupled with turbulence models. Techniques such as nonlinear Gauss–Seidel smoothing with coarse-grid corrections [48] and fully coupled multigrid methods [49] have improved convergence but still face stability challenges in high-gradient regions like boundary layers. Additionally, hardware-based strategies, such as GPU-assisted acceleration, have significantly reduced computation time for RANS-based fluid-structure interaction simulations in turbomachinery [50]. Similarly, CPU-GPU hybrid RANS solvers have shown up to 23% performance gains over pure CPU approaches [51]. Furthermore, integrating the high-performance linear solver library `PETSc` with `OpenFOAM` has enhanced RANS solver efficiency, reducing simulation time by 40% for large-scale three-dimensional flow simulations around a submarine [52]. However, these hardware-based methods require specialized resources and extensive code modifications, limiting their accessibility and increasing costs for broader applications of

RANS solvers.

In this dissertation, we focus on accelerating RANS simulations by optimizing the initialization process, as it serves as the starting point of the simulation and offers the most direct opportunity for improvement. A better initial guess that is closer to the final solution enables faster convergence of the iterative solver, reducing computational costs—particularly for complex turbulent flow simulations prone to early instabilities. Classical initialization methods for RANS simulations often rely on uniform flow fields or interpolated solutions from coarser grids to provide an initial guess for the solver [53]. A common approach is to use linear or quadratic extrapolation based on available boundary conditions and flow parameters. Although straightforward and easy to implement, these methods often struggle to capture complex flow features, leading to poor convergence behavior and even longer computational times. Another widely used strategy is steady-state preconditioning, where a simplified form of the governing equations, such as inviscid or laminar flow, is solved first to establish a stable initial field [54]. For example, potential flow solutions are frequently used in aerodynamic simulations around airfoils and wings to provide basic velocity and pressure distributions. While this approach offers a good starting point for external flows, it fails to account for viscous effects, boundary layers, and separation points, making it less effective for turbulent flow initialization in complex scenarios [55].

Recently, machine learning has been applied to initialize RANS and other steady-state fluid flow simulations by learning from simulation data and making predictions for similar flow conditions. This data-driven approach can more accurately capture complex flow features compared to the above initialization methods and has the potential to provide robust initial fields, making it particularly effective in reducing convergence time and improving stability in complex turbulent scenarios. For instance, recent studies have demonstrated the effectiveness of using convolutional neural networks (CNNs) for initializing RANS simulations of turbulent flows over airfoils [56]. By providing initial fields with more accurate downstream wake development, CNN-based initialization can reduce computational time and the number of required iterations by up to 20 times, significantly outperforming traditional methods that rely on simple extrapolation or freestream assumptions. Similarly, in the context of porous media flow, a fully connected neural network is trained to provide initial guesses for a two-phase flow problem based on saturation gradients and mobility ratios at each timestep, which reduces simulation time by up to 35% [57]. These successes highlight the critical role of initialization in determining the efficiency of complex flow simulations and demonstrate that machine learning offers a promising alternative for achieving more effective initial fields compared to conventional approaches.

Despite the effectiveness of machine learning-assisted initialization, these methods typically require that the predicted flow fields and the training data share the same grid and discretization. However, in practical aircraft applications, the mesh size often needs to be adjusted according to varying flow conditions and geometrical configurations. For instance, increasing the Reynolds number generally necessitates finer mesh resolution in the boundary layer region. This variability poses a significant challenge when applying machine learning-based

initialization, as the core issue is developing a robust initialization method that can generalize across different mesh discretizations and flow conditions, ensuring consistent performance regardless of changes in the computational setup.

1.1.3 Summary of RANS Closure and Simulation Acceleration Challenges

This dissertation targets three primary challenges of CFD in aerospace engineering. For RANS closure modeling, we aim to develop advanced turbulence models that go beyond the limitations of the two fundamental hypotheses in commonly used linear eddy viscosity models—the Boussinesq approximation and the weak equilibrium assumption.

1. To overcome the limitations of the Boussinesq approximation, nonlinear eddy viscosity models are developed, and a practical ensemble Kalman method-based framework has been established to learn the underlying nonlinear models from sparse and indirect measurements [35]. However, the effectiveness of the ensemble Kalman filter in highly nonlinear flow conditions can be hindered by its linear assumptions, and the high computational cost of simulating each ensemble member makes it especially challenging to apply in real-world, 3D simulations when computational resources are limited. **Therefore, the challenge is to develop more efficient ensemble-based filtering algorithms that can better assimilate observation data while maintaining computational efficiency**.

2. To overcome the limitations of the weak equilibrium assumption, Reynolds stress transport models (RSTMs) are a natural choice, as they directly model the dynamics of Reynolds stress tensors. However, RSTMs require additional closure efforts for terms such as the pressure-strain correlation, and their their computational cost is significantly higher than traditional models. **Thus, the challenge is to develop computationally efficient non-equilibrium models that can effectively capture the anisotropy transport physics implied by the RSTMs**.

For accelerating RANS simulations, we focus on optimizing the initialization process to reduce number of iterations and enhance solver stability.

3. While machine learning-based approaches show promise in providing significantly improved initial fields compared to traditional methods, they are often sensitive to variations in computational settings such as spatial discretization. **Thus, the challenge is to develop robust initialization methods that can generalize across different mesh discretizations and varying flow scenarios in practical aerospace applications**.

1.2 Neural Operators for Learning Complex Mappings

From a fundamental and mathematical perspective, the challenges of RANS closure modeling and simulation acceleration can be viewed as problems of modeling and learning complex mappings. Specifically, developing non-equilibrium turbulence models requires modeling *field-to-point* mappings from mean velocity fields in nonlocal regions to Reynolds stress anisotropies at target points; delivering improved initial conditions involves resolving *field-to-field* mappings from boundary conditions and geometrical configurations to near-converged velocity and pressure fields; and designing efficient ensemble-based filters for enhanced assimilation of sparse indirect observations requires identifying *ensemble-to-ensemble* mappings, transforming prior ensembles of TBNN weights to posterior ones, conditioned on the observation data. Neural operators naturally align with these tasks, as they are specifically designed to model and learn complex nonlocal and nonlinear mappings across different function spaces [58]. This makes them a promising tool for effectively capturing the intricate relationships required for RANS closure modeling and simulation acceleration.

1.2.1 Neural Operator

Recent advancements in data-driven modeling have introduced neural operators as a powerful tool for learning complex mappings between infinite-dimensional function spaces [58]. Unlike traditional neural networks, which operate on finite-dimensional input-output pairs and require re-design and re-training for each discretization, neural operators can approximate entire function spaces, enabling them to generalize across different input domains and resolutions [59].

Various types of neural operators have been developed and successfully applied in scientific and engineering fields. One of the earliest examples is the DeepONet, which employs a dual-network structure to learn nonlinear operators from data [59]. While effective in modeling complex phenomena such as fluid dynamics and material deformation [59, 60], its reliance on a fixed number of input sensors limits its adaptability to varying input dimensions, preventing it from achieving full resolution adaptability. To address these limitations, more advanced neural operators have been proposed. For instance, the graph kernel network (GKN) employs graph neural networks to approximate integral operators over graph structures [61]. This allows GKN to operate on irregular meshes and varying grid resolutions while maintaining consistent prediction accuracy, making it ideal for modeling nonlocal dependencies in complex geometries, such as fluid flows in porous media and atmospheric dynamics. Another prominent example is the Fourier neural operator (FNO), which parameterizes operators in the Fourier space using fast Fourier transforms to perform integration in the form of convolution in the frequency domain [62]. This enables FNO to achieve resolution invariance and perform zero-shot super-resolution, making it particularly effective for problems with high-frequency components, such as turbulence modeling and climate prediction.

Beyond GKN and FNO, other variants such as the multipole graph neural operator [63] and the attention-based neural operator [64] further enhance the capabilities of neural operators by addressing challenges like long-range interactions and nonlocal dependencies. These advancements demonstrate the versatility and potential of neural operators in solving complex problems across various scientific and engineering domains.

1.2.2 Neural Operator for Learning Nonlocal Mappings

Given the advantageous properties and recent advancements of neural operators, it is natural to consider leveraging them to model and learn the three levels of nonlocal mappings—field-to-point, field-to-field, and ensemble-to-ensemble—needed to address the challenges of RANS turbulence modeling and simulation acceleration. Here, we review the application of neural operators for learning the three types of nonlocal mappings in various scientific problems, highlighting their effectiveness and versatility in capturing complex interactions across different scales and areas.

Neural operators for learning field-to-point mappings. Field-to-point mappings are pervasive in modeling multiscale and multiphase systems, where closure models are frequently used to express localized, unresolved phenomena in terms of resolved field quantities [65]. Compared to traditional methods, neural operators can learn accurate closure models directly from data, capturing complex dependencies while maintaining relatively low computational cost in the subsequent simulations.

For example, in molecular dynamics, the Deep Potential method utilizes scalable deep neural networks to construct highly accurate potential energy surfaces by learning complex interatomic interactions directly from quantum mechanical data. This is achieved through a field-to-point mapping that translates the surrounding atomic positions into local energy values at each atom [66, 67]. Such an approach replaces traditional empirical force fields with data-driven models that deliver quantum-level accuracy while substantially reducing computational costs. Similarly, in particle-laden flows, equivariant neural networks are used to develop closure models that predict point-particle forces and torques based on the configurations of neighboring particles [68]. Unlike traditional Reynolds number and particle volume fraction-based models, this data-driven approach accurately captures complex interparticle interactions and force fluctuations, thus leading to more precise predictions across varying flow regimes and particle concentrations.

Neural operators for learning field-to-field mappings. Field-to-field mappings are ubiquitous in numerical simulations, as they capture the complex nonlinear and nonlocal interactions influenced by geometry, material properties, and initial and boundary conditions on fluid flow behavior [54]. These mappings are inherently represented within computational

models through PDEs, and resolving them typically requires solving the corresponding PDEs to accurately describe the flow dynamics. Neural operators can capture these mappings through data-driven training, eliminating the dependence on iterative solvers and reducing the computational burden typically associated with conventional simulation methods.

For instance, the Fourier neural operator has been applied to global weather forecasting by learning mappings between current atmospheric conditions and future states, achieving up to a 45,000× speedup over traditional numerical methods [69]. It has also been utilized for urban microclimate simulations, accurately predicting complex airflow and temperature variations around buildings [70]. In automotive design, neural operators have been used to predict aerodynamic fields around vehicles under varying operational conditions, allowing for rapid evaluation of design changes without the need for costly re-simulations [71]. These examples illustrate the transformative potential of neural operators in modeling field-to-field mappings across a wide range of scientific and engineering applications.

Neural operators for learning ensemble-to-ensemble mappings. Although neural operators have been successfully applied to field-to-point and field-to-field mappings, their application to learning ensemble-to-ensemble mappings remains largely unexplored. To the best of our knowledge, our work is the first to propose and investigate using neural operators to model these complex transformations by simultaneously capturing both ensemble statistics and individual member features, opening new possibilities for advancing ensemble-based data assimilation techniques.

Building on these developments, this dissertation aims to address the three challenges outlined in Subsection 1.1.3 by developing neural operators capable of learning the associated nonlocal and nonlinear mappings. A key aspect of this work is the design of neural operators that incorporate fundamental physical and mathematical principles directly into their architectures. By embedding these essential properties, the proposed neural operators are tailored to respect the underlying physics of fluid dynamics and problem-specific characteristics, ensuring that the resulting models are not only expressive but also robust, providing reliable solutions consistent with physical laws and mathematical constraints.

1.3 Structure and Content

This dissertation follows the *Manuscript* format and is divided into three parts. Part I contains Chapters 2 and 3 and consists of efforts related to the development of non-equilibrium RANS turbulence models through learning nonlocal *field-to-point* mappings from velocity fields in the neighborhood to Reynolds stresses at points of interest. Part II contains Chapter 4 and consists of efforts related to the acceleration of RANS simulations through improved initial conditions, which are obtained through learning *field-to-field* mappings from boundary conditions and geometrical configurations to converged velocity and pressure fields.

Part III contains Chapter 5 and consists of efforts related to the development of advanced ensemble-based filters through designing conditional *ensemble-to-ensemble* mappings from prior ensembles to posterior ones, with the potential of more accurately learning nonlinear eddy viscosity models from sparse indirect measurements and improving sequential data assimilation in various applications.

The content of each chapter is as follows:

Chapter 1: Introduces the challenges of CFD in aerospace engineering with an emphasis on RANS turbulence models and simulation acceleration, presents an overview and literature review of neural operators, and summarizes the structure, content, and contributions of the dissertation.

Part I: Non-equilibrium RANS Turbulence Modeling with Neural Operators

Chapter 2: Consists of a peer reviewed paper, entitled "*Frame-independent vector-cloud neural network for nonlocal constitutive modeling on arbitrary grids*" and published in Computer Methods in Applied Mechanics and Engineering (reproduced here with permission from Elsevier). This paper proposes a vector-cloud neural network for developing nonlocal scalar constitutive models. The model constructs a field-to-point mapping by predicting the closure variable at a point based on the mean flow field in its surrounding neighborhood. It effectively handles unstructured meshes and arbitrary grid points while respecting fundamental physical principles by remaining invariant to coordinate transformations.

Chapter 3: Consists of a peer reviewed paper, entitled "*An equivariant neural operator for developing nonlocal tensorial constitutive models*" and published in Journal of Computational Physics (reproduced here with permission from Elsevier). This paper enhances the vector-cloud neural network with rotational equivariance to develop nonlocal tensorial constitutive models. The predicted tensorial closure variables rotate consistently with the coordinate system, while the network retains all other essential properties inherent to the vector-cloud neural network.

Part II: Enhanced Initialization for RANS Simulations with Neural Operators

Chapter 4: Consists of a paper, entitled "*Neural operator-based super-fidelity: A warm-start approach for accelerating steady-state simulations*" and currently under review by the Journal of Computational Physics. This paper proposes neural operators as a warm-start tool, rather than surrogate models, to accelerate steady-state RANS simulations. Through data-driven training, neural operators can efficiently deliver near-converged flow fields for initialization, thereby reducing computational time.

Part III: Generalized Ensemble Filtering with Neural Operators

Chapter 5: Consists of a paper, entitled "*BI-EqNO: Generalized approximate Bayesian inference with an equivariant neural operator framework*" and in preparation for submission to a peer reviewed journal. This paper introduces a permutation-equivariant neural operator framework for approximate Bayesian inference, which can be further developed for various applications, such as ensemble neural filters for data assimilation. The ensemble neural filter is a generalized ensemble filter, which is trainable and has the potential of learning nonlinear eddy viscosity models from sparse indirect data more accurately.

1.4 Contributions

This dissertation provides innovative solutions to address the three primary challenges in RANS closure modeling and simulation acceleration by introducing a fundamental and mathematical perspective that frames these challenges as learning nonlocal functional mappings. Specifically, the main contributions are as follows:

1. **Development of a frame-independent neural operator for non-equilibrium turbulence modeling**: To address the limitations of the weak equilibrium assumption in conventional linear eddy viscosity models, this dissertation proposes a novel neural operator framework that directly learns the complex field-to-point mapping from nonlocal velocity fields to Reynolds stress tensors. The model's frame-independent design ensures preservation of physical symmetries, offering a robust and computationally efficient alternative to Reynolds stress transport models.

2. **Design and validation of neural operators as initialization tools to accelerate steady-state simulations**: To address the challenge of accelerating RANS simulations, this dissertation presents a neural operator-based approach for delivering improved initial conditions, which involves learning field-to-field mappings from boundary conditions and geometrical configurations to velocity and pressure fields. This acceleration method is well-suited for many practical aerospace applications where computational accuracy is paramount.

3. **Development of a permutation-equivariant neural operator framework for generalized ensemble-based data assimilation**: To enhance the efficiency of learning nonlinear eddy viscosity models from sparse indirect data, this dissertation introduces a novel ensemble neural filter that generalizes traditional ensemble filtering methods. The proposed filter utilizes a permutation-equivariant neural operator to represent the complex ensemble-to-ensemble mappings, thereby offering the potential of improving data assimilation performance through data-driven training, which is particularly promising in highly nonlinear scenarios with limited computational resources.

1.5 Publications

Articles included in this dissertation:

- **X.-H. Zhou**, Z.-R. Liu, and H. Xiao. *BI-EqNO: Generalized approximate Bayesian inference with an equivariant neural operator framework*. Under review. https://arxiv.org/abs/2410.16420.
- **X.-H. Zhou**, J. Han, M. I. Zafar, E. M. Wolf, C. R. Schrock, C. J. Roy, and H. Xiao. *Neural operator-based super-fidelity: A warm-start approach for accelerating steady-state simulations*. Under review. https://arxiv.org/abs/2312.11842.
- J. Han, **X.-H. Zhou**, and H. Xiao. *An equivariant neural operator for developing nonlocal tensorial constitutive models*. Journal of Computational Physics, 2023. https://doi.org/10.1016/j.jcp.2023.112243.
- **X.-H. Zhou**, J. Han, and H. Xiao. *Frame-independent vector-cloud neural network for nonlocal constitutive modelling on arbitrary grids*. Computer Methods in Applied Mechanics and Engineering, 2022. https://doi.org/10.1016/j.cma.2021.114211.

Other articles during PhD:

- M. I. Zafar, **X.-H. Zhou**, C. J. Roy, D. Stelter, and H. Xiao. *Data-driven turbulence modeling approach for cold-wall hypersonic boundary layers*. Under review. https://arxiv.org/abs/2406.17446.
- **X.-H. Zhou**, H. Wang, J. McClure, C. Chen, and H. Xiao. *Inference of relative permeability curves in reservoir rocks with ensemble Kalman method*. The European Physical Journal E, 2023. https://doi.org/10.1140/epje/s10189-023-00296-5.
- Y. Lu, **X.-H. Zhou**, H. Xiao, and Q. Li. *Using machine learning to predict urban canopy flows for land surface modeling*. Geophysical Research Letters, 2023. https://doi.org/10.1029/2022GL102313.
- R. Xu, **X.-H. Zhou**, J. Han, R. P. Dwight, and H. Xiao. *A PDE-free, neural network-based eddy viscosity model coupled with RANS equations*. International Journal of Heat and Fluid Flow, 2022. https://doi.org/10.1016/j.ijheatfluidflow.2022.109051.
- **X.-H. Zhou**, J. McClure, C. Chen, and H. Xiao. *Neural network-based pore flow field prediction in porous media using super resolution*. Physical Review Fluids, 2022. https://link.aps.org/doi/10.1103/PhysRevFluids.7.074302.

- M. I. Zafar, J. Han, **X.-H. Zhou**, and H. Xiao. *Frame invariance and scalability of neural operators for partial differential equations*. Communications in Computational Physics, 2022. https://doi.org/10.4208/cicp.OA-2021-0256.

- **X.-H. Zhou**, J. Han, and H. Xiao. *Learning nonlocal constitutive models with neural networks*. Computer Methods in Applied Mechanics and Engineering, 2021. https://doi.org/10.1016/j.cma.2021.113927.

1.6 Attributions

The nature of technical research is collaborative, which is reflected in the number of co-authored papers produced during my Ph.D. studies. A complete list of these articles is provided in Section 1.5. This dissertation includes only articles related to neural operators, for which I was responsible for the majority of the technical implementation and served as one of the primary writers. The individual contributions of the co-authors to this research are as follows:

- *Dr. Heng Xiao* is my Ph.D. advisor. He contributed to the conceptualization, planning, and discussions, provided advice and guidance, and reviewed and edited my writing. He also played a key role in writing the introduction section of Chapter 2.

- *Dr. Jiequn Han* was a collaborator. He contributed to the methodological design, participated in discussions, provided advice and guidance, and reviewed and edited my writing. He also played a key role in writing the methodology section of Chapter 2 and introduction section of Chapter 3.

- *Dr. Christopher Roy* is a collaborating professor and a member of my Ph.D. committee. He participated in discussions, provided advice and guidance, and reviewed and edited my writing.

- *Dr. Muhammad Irfan Zafar* was a fellow doctoral student. He participated in discussions and contributed to the data generation for Chapter 4.

- *Mr. Zhuo-Ran Liu* is a junior doctoral student. He participated in discussions and contributed to the technical implementation of Chapter 5.

Bibliography

[1] Philippe R Spalart and V Venkatakrishnan. On the role and challenges of cfd in the aerospace industry. *The Aeronautical Journal*, 120(1223):209–232, 2016.

[2] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence modeling in the age of data. *Annual review of fluid mechanics*, 51(1):357–377, 2019.

[3] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual review of fluid mechanics*, 52(1):477–508, 2020.

[4] Junji Huang, Gary L Nicholson, Lian Duan, Meelan M Choudhari, and Rodney D Bowersox. Simulation and modeling of cold-wall hypersonic turbulent boundary layers on flat plate. In *AIAA Scitech 2020 Forum*, page 0571, 2020.

[5] W Cabot and Parviz Moin. Approximate wall boundary conditions in the large-eddy simulation of high reynolds number flow. *Flow, Turbulence and Combustion*, 63:269–291, 2000.

[6] Bruno Chaouat. The state of the art of hybrid RANS/LES modeling for the simulation of turbulent flows. *Flow, turbulence and combustion*, 99:279–327, 2017.

[7] Heng Xiao and Paola Cinnella. Quantification of model uncertainty in RANS simulations: A review. *Progress in Aerospace Sciences*, 108:1–31, 2019.

[8] Jeffrey P Slotnick, Abdollah Khodadoust, Juan Alonso, David Darmofal, William Gropp, Elizabeth Lurie, and Dimitri J Mavriplis. Cfd vision 2030 study: a path to revolutionary computational aerosciences. Technical report, 2014.

[9] K HANJALIC. Will RANS survive LES? a view of perspectives. *Journal of fluids engineering*, 127(5):831–839, 2005.

[10] DC Wilcox. Turbulence modeling for cfd. *DCW industries, La Canada*, 1998.

[11] Philippe Spalart and Steven Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th aerospace sciences meeting and exhibit*, page 439, 1992.

[12] W Peter Jones and Brian Edward Launder. The prediction of laminarization with a two-equation model of turbulence. *International journal of heat and mass transfer*, 15 (2):301–314, 1972.

[13] Stephen B Pope. Turbulent flows. *Measurement Science and Technology*, 12(11):2020–2021, 2001.

[14] Yuan Fang, Yaomin Zhao, Fabian Waschkowski, Andrew SH Ooi, and Richard D Sandberg. Toward more general turbulence models via multicase computational-fluid-dynamics-driven training. *AIAA Journal*, 61(5):2100–2115, 2023.

[15] Florian R Menter. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA journal*, 32(8):1598–1605, 1994.

[16] PA Durbin. *Statistical Theory and Modeling for Turbulent Flows*. Wiley Sons Ltd, 2011.

[17] Heng Xiao, Jin-Long Wu, Sylvain Laizet, and Lian Duan. Flows over periodic hills of parameterized geometries: A dataset for data-driven turbulence modeling from direct simulations. *Computers & Fluids*, 200:104431, 2020.

[18] Yining Luo, Yingfa Chen, and Zhen Zhang. Cfdbench: A comprehensive benchmark for machine learning methods in fluid dynamics. *arXiv preprint arXiv:2310.05963*, 2023.

[19] Ryley McConkey, Eugene Yee, and Fue-Sang Lien. A curated dataset for data-driven turbulence modelling. *Scientific data*, 8(1):255, 2021.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[21] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[22] Myoungkyu Lee and Robert D Moser. Direct numerical simulation of turbulent channel flow up to. *Journal of fluid mechanics*, 774:395–415, 2015.

[23] Yukio Kaneda and Yoshinobu Yamamoto. Velocity gradient statistics in turbulent shear flow: an extension of kolmogorov's local equilibrium theory. *Journal of Fluid Mechanics*, 929:A13, 2021.

[24] Sergio Hoyas, Martin Oberlack, Francisco Alcántara-Ávila, Stefanie V Kraheberger, and Jonathan Laux. Wall turbulence at high friction reynolds numbers. *Physical Review Fluids*, 7(1):014602, 2022.

[25] Sergio Pirozzoli, Davide Modesti, Paolo Orlandi, and Francesco Grasso. Turbulence and secondary motions in square duct flow. *Journal of Fluid Mechanics*, 840:631–655, 2018.

[26] Jiayi Cai, Pierre-Emmanuel Angeli, Jean-Marc Martinez, Guillaume Damblin, and Didier Lucor. Revisiting tensor basis neural network for reynolds stress modeling: Application to plane channel and square duct flows. *Computers & Fluids*, 275:106246, 2024.

[27] Mikael LA Kaandorp and Richard P Dwight. Data-driven modelling of the reynolds stress tensor using random forests with invariance. *Computers & Fluids*, 202:104497, 2020.

[28] Jack Weatheritt and Richard Sandberg. A novel evolutionary algorithm applied to algebraic modifications of the RANS stress–strain relationship. *Journal of Computational Physics*, 325:22–37, 2016.

[29] Yaomin Zhao, Harshal D Akolekar, Jack Weatheritt, Vittorio Michelassi, and Richard D Sandberg. RANS turbulence model development using cfd-driven machine learning. *Journal of Computational Physics*, 411:109413, 2020.

[30] Sarah Beetham and Jesse Capecelatro. Formulating turbulence closures using sparse regression with embedded form invariance. *Physical Review Fluids*, 5(8):084611, 2020.

[31] Soufiane Cherroud, Xavier Merle, Paola Cinnella, and Xavier Gloerfelt. Sparse bayesian learning of explicit algebraic reynolds-stress models for turbulent separated flows. *International Journal of Heat and Fluid Flow*, 98:109047, 2022.

[32] Martin Schmelzer, Richard P Dwight, and Paola Cinnella. Discovery of algebraic reynolds-stress models using sparse symbolic regression. *Flow, Turbulence and Combustion*, 104:579–603, 2020.

[33] Eric J Parish and Karthik Duraisamy. A paradigm for data-driven predictive modeling using field inversion and machine learning. *Journal of computational physics*, 305:758–774, 2016.

[34] Nikola B Kovachki and Andrew M Stuart. Ensemble kalman inversion: a derivative-free technique for machine learning tasks. *Inverse Problems*, 35(9):095005, 2019.

[35] Xin-Lei Zhang, Heng Xiao, Xiaodong Luo, and Guowei He. Ensemble kalman method for learning turbulence models from indirect observation data. *Journal of Fluid Mechanics*, 949:A26, 2022.

[36] Xin-Lei Zhang, Heng Xiao, Xiaodong Luo, and Guowei He. Combining direct and indirect sparse data for learning generalizable turbulence models. *Journal of Computational Physics*, 489:112272, 2023.

[37] Geir Evensen, Femke C Vossepoel, and Peter Jan Van Leeuwen. *Data assimilation fundamentals: A unified formulation of the state and parameter estimation problem*. Springer Nature, 2022.

[38] Geir Evensen. *Data Assimilation: The Ensemble Kalman Filter*. Springer, 2009.

[39] Chao Zhang, Lian Duan, and Meelan M Choudhari. Direct numerical simulation database for supersonic and hypersonic turbulent boundary layers. *AIAA journal*, 56 (11):4297–4311, 2018.

[40] Matthew Aultman, Zhenyu Wang, Rodrigo Auza-Gutierrez, and Lian Duan. Evaluation of cfd methodologies for prediction of flows around simplified and complex automotive models. *Computers & Fluids*, 236:105297, 2022.

[41] Brian Edward Launder, G Jr Reece, and W Rodi. Progress in the development of a reynolds-stress turbulence closure. *Journal of fluid mechanics*, 68(3):537–566, 1975.

[42] Charles G Speziale, Sutanu Sarkar, and Thomas B Gatski. Modelling the pressure–strain correlation of turbulence: an invariant dynamical systems approach. *Journal of fluid mechanics*, 227:245–272, 1991.

[43] Zinon Vlahostergios. Performance assessment of reynolds stress and eddy viscosity models on a transitional dca compressor blade. *Aerospace*, 5(4):102, 2018.

[44] Jyoti P Panda and Hari V Warrior. Modeling the pressure strain correlation in turbulent flows using deep neural networks. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 236(7):3447–3458, 2022.

[45] Jyoti Prakash Panda and Hari Vijayan Warrior. Evaluation of machine learning algorithms for predictive reynolds stress transport modeling. *Acta Mechanica Sinica*, 38(4): 321544, 2022.

[46] Matteo Tugnoli, Davide Montagnani, Monica Syal, Giovanni Droandi, and Alex Zanotti. Mid-fidelity approach to aerodynamic simulations of unconventional vtol aircraft configurations. *Aerospace Science and Technology*, 115:106804, 2021.

[47] Zhoujie Lyu, Gaetan K Kenway, and Joaquim RRA Martins. RANS-based aerodynamic shape optimization investigations of the common research model wing. In *52nd aerospace sciences meeting*, page 0567, 2014.

[48] Jeroen Wackers and Barry Koren. Multigrid solution method for the steady RANS equations. *Journal of Computational Physics*, 226(2):1784–1807, 2007.

[49] Mark Wasserman, Yair Mor-Yossef, Irad Yavneh, and J Barry Greenberg. A robust implicit multigrid method for RANS equations with two-equation turbulence models. *Journal of Computational Physics*, 229(16):5820–5842, 2010.

[50] Andrea Gadda, Luca Mangani, Giulio Romanelli, Paolo Mantegazza, and Ernesto Casartelli. A gpu-accelerated compressible RANS solver for fluid-structure interaction simulations in turbomachinery. In *16th International Symposium on Transport Phenomena and Dynamics of Rotating Machinery*, 2016.

[51] Weicheng Xue, Hongyu Wang, and Christopher J Roy. Cpu–gpu heterogeneous code acceleration of a finite volume computational fluid dynamics solver. *Future Generation Computer Systems*, 158:367–377, 2024.

[52] Simone Bna, Ivan Spisso, Mark Olesen, and Giacono Rossi. Petsc4foam: A library to plug-in petsc into the openfoam framework. *PRACE White paper*, 2020.

[53] Joel H Ferziger, Milovan Perić, and Robert L Street. *Computational methods for fluid dynamics*. springer, 2019.

[54] Jiri Blazek. *Computational fluid dynamics: principles and applications*. Butterworth-Heinemann, 2015.

[55] John Anderson. *Fundamentals of Aerodynamics*. McGraw hill, 2011.

[56] Kazuko W Fuchi, Eric M Wolf, Christopher R Schrock, and Philip S Beran. Acceleration of RANS solver convergence via initialization with wake extension models. *Available at SSRN 4750758*, 2024.

[57] Jassem Abbasi and Pål Østebø Andersen. Improved initialization of non-linear solvers in numerical simulation of flow in porous media with a deep learning approach. In *SPE Europec featured at EAGE Conference and Exhibition?*, page D031S008R005. SPE, 2022.

[58] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[59] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

[60] Somdatta Goswami, Minglang Yin, Yue Yu, and George Em Karniadakis. A physics-informed variational deeponet for predicting crack path in quasi-brittle materials. *Computer Methods in Applied Mechanics and Engineering*, 391:114587, 2022.

[61] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[62] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[63] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020.

[64] Edoardo Calvello, Nikola B Kovachki, Matthew E Levine, and Andrew M Stuart. Continuum attention for neural operators. *arXiv preprint arXiv:2406.06486*, 2024.

[65] Benjamin Sanderse, Panos Stinis, Romit Maulik, and Shady E Ahmed. Scientific machine learning for closure models in multiscale problems: A review. *arXiv preprint arXiv:2403.02913*, 2024.

[66] Jiequn Han, Linfeng Zhang, Roberto Car, et al. Deep potential: A general representation of a many-body potential energy surface. *arXiv preprint arXiv:1707.01478*, 2017.

[67] Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics. *Physical review letters*, 120(14):143001, 2018.

[68] B Siddani and S Balachandar. Point-particle drag, lift, and torque closure models using machine learning: Hierarchical approach and interpretability. *Physical Review Fluids*, 8(1):014303, 2023.

[69] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

[70] Wenhui Peng, Shaoxiang Qin, Senwen Yang, Jianchun Wang, Xue Liu, and Liangzhu Leon Wang. Fourier neural operator for real-time simulation of 3d dynamic urban microclimate. *Building and Environment*, 248:111063, 2024.

[71] Zongyi Li, Nikola Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, et al. Geometry-informed neural operator for large-scale 3D PDEs. *Advances in Neural Information Processing Systems*, 36, 2024.

Part I

Field-to-Point Mapping: Neural Operator for Developing Non-equilibrium RANS Turbulence Models

Chapter 2

Frame-independent vector-cloud neural network for nonlocal constitutive modeling on arbitrary grids

Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 388 (2022) 114211
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Frame-independent vector-cloud neural network for nonlocal constitutive modeling on arbitrary grids

Xu-Hui Zhou[a,1], Jiequn Han[b,*,1], Heng Xiao[a]

[a] *Kevin T. Crofton Department of Aerospace and Ocean Engineering, Virginia Tech, Blacksburg, VA 24060, USA*
[b] *Department of Mathematics, Princeton University, Princeton, NJ 08544, USA*

Received 13 May 2021; received in revised form 20 August 2021; accepted 21 September 2021
Available online 27 October 2021

Abstract

Constitutive models are widely used for modeling complex systems in science and engineering, where first-principle-based, well-resolved simulations are often prohibitively expensive. For example, in fluid dynamics, constitutive models are required to describe nonlocal, unresolved physics such as turbulence and laminar–turbulent transition. However, traditional constitutive models based on partial differential equations (PDEs) often lack robustness and are too rigid to accommodate diverse calibration datasets. We propose a frame-independent, nonlocal constitutive model based on a vector-cloud neural network that can be learned with data. The model predicts the closure variable at a point based on the flow information in its neighborhood. Such nonlocal information is represented by a group of points, each having a feature vector attached to it, and thus the input is referred to as vector cloud. The cloud is mapped to the closure variable through a frame-independent neural network, invariant both to coordinate translation and rotation and to the ordering of points in the cloud. As such, the network can deal with any number of arbitrarily arranged grid points and thus is suitable for unstructured meshes in fluid simulations. The merits of the proposed network are demonstrated for scalar transport PDEs on a family of parameterized periodic hill geometries. The vector-cloud neural network is a promising tool not only as nonlocal constitutive models and but also as general surrogate models for PDEs on irregular domains.

Keywords: Constitutive modeling; Nonlocal closure model; Symmetry and invariance; Inverse modeling; Deep learning

1. Introduction

Constitutive models are widely encountered in science and engineering when the computational costs of simulating complex systems are often prohibitively high. Taking industrial computational fluid dynamics (CFD) for example, constitutive models such as Reynolds stress models and eddy viscosity models are often used to describe unresolved turbulence and to *close* the Reynolds averaged Navier–Stokes (RANS) equations for the mean flows fields, which are of ultimate interest in engineering design. As such, they are also referred to as "closure models", a term that is used interchangeably in this work. Closure models are also used to describe laminar–turbulent transitions, which typically occur near the leading edge of airfoils or gas engine turbines. Despite the

* Corresponding author.
E-mail addresses: jiequnhan@gmail.com (J. Han), hengxiao@vt.edu (H. Xiao).
[1] Contributed equally.

https://doi.org/10.1016/j.cma.2021.114211
0045-7825/

enormous growth of available computational resources in the past decades, even to this day such closure models are still the backbone of engineering CFD solvers. However, these models are typically based on a hybrid of partial differential equations (PDEs) and algebraic relations, which are difficult to calibrate. In particular, it is challenging to accommodate diverse sets of calibration data, especially those from realistic, complex flows.

In the past few years, the emergence of neural networks and other machine learning models has led to unprecedented opportunities for constitutive modeling. Among the most attractive features of neural-network-based models is their expressive power, which could enable flexible constitutive models to be calibrated on a wide range of datasets and consequently lead to a *de facto* universal model, or at least a *unified* model with automatic internal switching depending on the regime. This has long been an objective of constitutive model development such as in turbulence modeling. In light of such strengths and promises, researchers have made efforts to develop data-driven, machine-learning-based constitutive models in a wide range of application areas such as turbulence modeling [1–6] and computational mechanics [7–13] in general.

Most existing machine-learning-based constitutive models are local, algebraic functions of the form $\tau = f(\mathbf{u})$, i.e., the closure variable (denoted as τ) at $\boldsymbol{x}_i$ depends solely on the resolved variable (e.g., velocity $\mathbf{u}$) or at most the derivatives thereof (e.g., the strain rate) at the *same* point $\boldsymbol{x}_i$. Such local neural network models are based on the commonly used turbulence models such as the eddy-viscosity models, which are derived based on the weak equilibrium assumption and assume a linear or nonlinear mapping from the local mean strain-rate to the Reynolds stress anisotropy at the same point [14]. However, the underlying unresolved physics to be represented by the constitutive models can be nonlocal. This is most evidently seen in turbulence models. The unresolved turbulent velocity fluctuation (which can be described by its second order statistics, i.e., the correlation, Reynolds stress) depends on an upstream neighborhood due to the convection and diffusion [15]. In fact, the transport physics of Reynolds stresses can be exactly described by a set of coupled convection–diffusion PDEs, which unfortunately contain unclosed terms of even higher order statistics. Such unclosed terms include triple velocity correlation, pressure-strain-rate and dissipation, all of which require extra modeling. The pressure–strain-rate term, in particular, greatly influences the performance of the Reynolds stress model but is notoriously difficult to model. Moreover, it is often challenging to achieve convergence in industrial applications due to unstructured grids of low quality and complex flow configurations. Therefore, despite the theoretical superiority, the Reynolds stress model has not been widely used because of its lack of robustness [16]. Other PDE-based constitutive models in industrial CFD include transport equations for intermittency [17] and amplification factor [18] in laminar–turbulent transition modeling and the transport of eddy viscosity [19], turbulent kinetic energy, dissipation [20], and frequencies [21] in turbulence modeling. Such closure models all imply a region-to-point mapping from the resolved primary variable (typically the flow field) to the respective closure variable.

In light of the observations above, it seems natural to explore using neural networks to mimic such nonlocal, region-to-point mapping for constitutive modeling. Such neural-network-based constitutive models both preserve the nonlocal mapping dictated by the underlying physics and admit diverse sets of training data. A natural first attempt is to draw inspirations from image recognition, where convolutional neural networks (CNN) are used to learn a nonlocal mapping from a region of pixels to its classification (e.g., nature of the object it represents, presence of tumor). This avenue has been explored in our earlier work [22]. We have demonstrated that the CNN-based model learns not only an accurate closure model but also the underlying Green's function when the physics is described by a linear convection–diffusion PDE. This observation was also confirmed in recent works from other groups [23,24].

Unfortunately, constitutive modeling (and physical modeling in general) poses much more stringent requirements than those in computer vision applications, and a straightforward application of those methods would not suffice. More specifically, the constitutive model must be *objective*, i.e., indifferent to the material frame [25]. To put it plainly, a constitutive model should not vary depending on the coordinate system (e.g., origin and orientation) or reference systems for zero velocity (i.e., Galilean invariance) or pressure (absolute versus gauge pressure). The frame-independence requirement immediately rules out CNN as an ideal vehicle for general constitutive modeling except for the special problems that are already equipped with intrinsic coordinates (e.g., the normal and tangential directions of the wall in the modeling of near-wall turbulence). Needless to say, all equations in classical mechanics from Newton's second law to Navier–Stoke equations already satisfy the frame-independence requirement as they faithfully describe the physics, which are frame-independent. When developing constitutive models, the frame-independence requirement restricts the set of functions that are admissible as constitutive models [26], be it stress–strain model for elastic materials, eddy viscosity models of turbulence [27,28], or pressure–strain-rate models for Reynolds stress transport equations [29]. However, in the realm of data-driven modeling, such

conventional wisdom is not directly applicable, as the candidate function forms are dictated by the adopted type of machine learning model. Specifically, as neural networks are successive compositions of linear functions and nonlinear squashing, it is not straightforward to restrict the function forms as practiced in traditional constitutive modeling [27–29]. Recently, global operators or surrogate models parameterized by machine learning models for approximating solutions of the PDEs [see, e.g., 24,30–39] have emerged as a new computation tool, which also hold the promise to serve as nonlocal constitutive modeling. However, the objectivity of these modeling approaches, such as frame-independence and permutational invariance introduced below, has rarely been discussed in the literature.

Alternative strategies have been proposed to achieve frame-independence for data-driven modeling. The straightforward method borrowed from computer vision [40,41] is to augment the training data by duplicating them to different coordinate systems (e.g., rotated by a series of random angles) before conducting the training [42]. That way, functions that deviate from frame-independence too much are implicitly rejected in the training process. However, this method is inefficient and does not guarantee strict frame-independence. Another strategy is to approximate the constitutive model in an admissible form, a function only processing invariants of the raw features, such as the magnitude of velocities, eigenvalues of strain-rate tensor [42], Q criteria of the velocity [3], or other handcrafted scalars. However, while it does eliminate dependence on the coordinate orientation, this may result in information loss on the relative direction among the input vectors (and tensors). For example, consider a hypothetical constitutive model where the turbulent kinetic energy τ (a scalar quantity) at a given location $\boldsymbol{x}_0$ is formulated as a function of the mean velocities $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$ at three cells in the neighborhood of $\boldsymbol{x}_0$. Restricting admissible functions to the form $\tau = f(|\mathbf{u}_1|, |\mathbf{u}_2|, |\mathbf{u}_3|)$, i.e., depending only on the velocity magnitudes, is too restrictive, although it does ensure rotational invariance. Rather, the *pairwise* inner-products (mutual projections) of the velocities also need to be included as input to make input information complete [43,44] and still rotational invariant, i.e., $\tau = f(|\mathbf{u}_1|, |\mathbf{u}_2|, |\mathbf{u}_3|, \mathbf{u}_1^\top\mathbf{u}_2, \mathbf{u}_1^\top\mathbf{u}_3, \mathbf{u}_2^\top\mathbf{u}_3)$. Note that our convention in this paper is that all vectors are column vectors. More concisely it can be written as $\tau = f\left(\mathcal{Q}\mathcal{Q}^\top\right)$ with $\mathcal{Q} = [\![\mathbf{u}_1^\top, \mathbf{u}_2^\top, \mathbf{u}_3^\top]\!]$ and $[\![\cdot]\!]$ indicating vertical concatenation of velocity vectors to form matrix $\mathcal{Q}$.

In addition to the frame-independence that leads to translational and rotational invariance, the nonlocal mapping used in practice also involves another kind of invariance, the permutational invariance. In computational mechanics, unstructured meshes are often employed for discretization (e.g., in finite volume or finite element methods), where there is no intrinsic order for indexing the cells. Therefore, if we aim to construct an objective nonlocal mapping from the data in these cells, the mapping must be independent of the index order. In the previous hypothetical example, this requirement dictates that the prediction τ must remain the same whether the input is $(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$, $(\mathbf{u}_3, \mathbf{u}_1, \mathbf{u}_2)$, or any other ordering of three velocity vectors. In other words, the closure variable τ must only depend on the set as a whole and not on the ordering of its elements.

In various fields of physical modeling, it has been observed that neural-network (and other data-driven models) that respect all the physical constraints and symmetries tend to achieve the best performance [45–49]. In light of such insight, in recent years researchers have made significant efforts in to imposing physical constraints on neural networks for emulation, closure modeling, reduced-order modeling and discovering of physical systems [see, e.g., 4–6,49–53]. In the present work we draw inspirations from the recently developed neural-network-based potential, Deep Potential [54,55], to develop the nonlocal closure model that satisfies the aforementioned invariant properties in the context of computational mechanics. Potential energy is the basic but central building block in molecular dynamics, which are common tools in many disciplines, including physics, chemistry, and material science. Potential energy is a scalar function that takes all the considered atoms' positions and types as input. Physics dictates that the potential energy is frame-independent and permutational invariant with atoms of the same type. Deep Potential guarantees all these invariances and has achieved remarkable success in accelerating the simulation of large systems with *ab initio* accuracy [56–58] thanks to the approximating ability of neural networks.

Our main contribution in the present work is in using a frame-independent *vector-cloud neural network* to capture the nonlocal physics in convection–diffusion PDEs, which are commonly used in closure modeling. The vector-cloud network maps the input data matrix $\mathcal{Q}$, defined by a set of points in a region (referred to as *cloud*), to the closure variable τ as shown in Fig. 1. Each point in the cloud has a vector $\mathbf{q}$ attached to it, which defines a row in $\mathcal{Q}$. The vector consists of the point's frame-dependent coordinates $\boldsymbol{x}$ and resolved variables $\mathbf{u}$ (e.g., velocity or strain) as well as frame-independent scalar features $\boldsymbol{c}$. The input features of the vector-cloud network (detailed in Table 1 later) are specifically tailored to reflect the underlying transport physics, but the network is otherwise similar to Deep Potential. The vector-cloud neural network is constructed to be invariant both to coordinate translation and

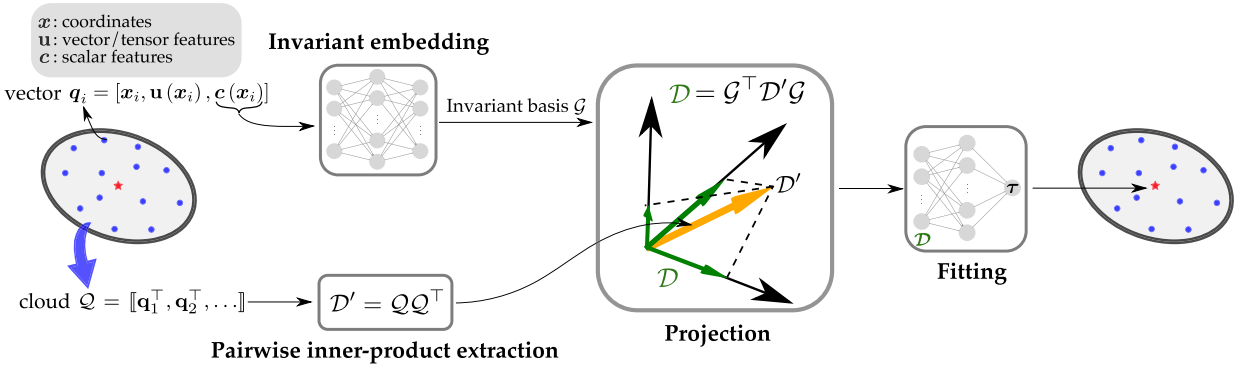


Fig. 1. Schematic of the frame-independent, permutation-invariant vector-cloud neural network for nonlocal constitutive modeling, showing a mapping $\mathcal{Q} \mapsto \tau$ from a cloud (left oval) of feature vectors $\mathcal{Q} = [\![\mathbf{q}_1^\top, \mathbf{q}_2^\top, \ldots]\!]$ to the closure variable τ (center of right oval). We construct the mapping by starting with two simultaneous operations: (i) extract pairwise inner-product to obtain rotational invariant features $\mathcal{D}'_{ii'} = \mathbf{q}_i^\top \mathbf{q}_{i'}$ and (ii) map the scalar quantities $\boldsymbol{c}$ in each vector $\mathbf{q}$ in the cloud through an embedding network to form a permutational invariant basis $\mathcal{G}$, which also inherits its rotational invariance from input $\boldsymbol{c}$. Then, we project $\mathcal{D}'$ onto basis $\mathcal{G}$ (not necessarily orthogonal) to produce final feature matrix $\mathcal{D}$, which is invariant to frame and permutation. Finally, we fit a neural network to map features $\mathcal{D}$ to closure variable τ. More details on the workflow and implementation are shown in Fig. 2.

Table 1
Flow features vector $\mathbf{q}$ used as input to the neural network arranged in three groups separated by lines: relative coordinates, velocities, and scalar quantities, i.e., $\mathbf{q} = [\![\boldsymbol{x}', \mathbf{u}, \boldsymbol{c}]\!]$. The final input features are obtained by processing the corresponding raw features. Notations are as follows: $\boldsymbol{x}_0 = (x_0, y_0)$, coordinate of cloud center; $\boldsymbol{x}' = \boldsymbol{x} - \boldsymbol{x}_0$, relative coordinate to cloud center, $\epsilon_0 (= 10^{-5})$, a small number to avoid singularity; $\hat{u}$ and $\hat{v}$, x- and y-component of velocity $\mathbf{u}$; $\overline{\theta}$, the average cell volume for all sampled data points within a cloud; $\epsilon_r = 0.01$, constant in reciprocal to ensure the diminishing importance of point towards the edge of the cloud; η, the wall distance normalized by boundary layer thickness scale ℓ_δ and capped by 1; $\epsilon_1 = 1.05, \epsilon_2 = 10^{-10}$ m^2/s are used to shift the cosine of the alignment angle to positive range; $\|\cdot\|$ and $|\cdot|$ indicates tensor and vector norms. L_0, U_0 and T_0 denote the characteristic length, flow velocity and time, respectively.

Category	Feature	Description of raw feature	Definition
Relative coordinates	x'	Relative x-coordinate to cloud center	$\frac{x - x_0}{\lvert\boldsymbol{x} - \boldsymbol{x}_0\rvert + \epsilon_0}$
	y'	Relative y-coordinate to cloud center	$\frac{y - y_0}{\lvert\boldsymbol{x} - \boldsymbol{x}_0\rvert + \epsilon_0}$
Velocity vector	u	x-component of velocity	$\hat{u}/U_0$
	v	y-component of velocity	$\hat{v}/U_0$
Scalar quantities	θ	Cell volume $\hat{\theta}$	$\hat{\theta}/\overline{\theta}$
	s	Magnitude of strain rate $\mathbf{s}$	$\lVert\mathbf{s}\rVert T_0$
	b	Boundary cell indicator	1 (yes) or 0 (no)
	u	Velocity magnitude	$\lvert\mathbf{u}\rvert/U_0$
	η	Wall distance $\hat{\eta}$	$\min(\hat{\eta}/\ell_\delta, 1)$
	r	Proximity to cloud center	$\frac{\epsilon_r}{\lvert\boldsymbol{x} - \boldsymbol{x}_0\rvert/L_0 + \epsilon_r}$
	r'	Proximity in local velocity frame	$\frac{r\lvert\mathbf{u}\rvert}{U_0}\left(\epsilon_1 - \frac{\mathbf{u}^\top \boldsymbol{x}'}{\lvert\mathbf{u}\rvert\lvert\boldsymbol{x}'\rvert + \epsilon_2}\right)$

rotation and to the ordering of the points. The invariance is achieved by the following operations as illustrated in Fig. 1: (i) compute pairwise inner products $\mathcal{D}'_{ii'} = \mathbf{q}_i^\top \mathbf{q}_{i'}$ among all vectors in the cloud to obtain rotational invariant features, (ii) map the scalar features to a rotational invariant basis $\mathcal{G}$ with an embedding network, and (iii) project the inner product $\mathcal{D}'$ onto the basis to form invariant feature $\mathcal{D} = \mathcal{G}^\top \mathcal{D}' \mathcal{G}$, which is then further mapped to τ with a neural network. The mapping $\mathcal{Q} \mapsto \tau$ is frame-independent and permutational invariant as $\mathcal{D}$ is invariant. More details of the workflow will be presented in Section 2 and further illustrated in Fig. 2.

2. Problem statement and proposed methodology

In the context of turbulence modeling for incompressible flows of constant density ρ, the mean flow fields (velocity $\mathbf{u}$ and pressure p) are described by the following RANS equations:

$$\mathbf{u}\cdot\nabla\mathbf{u}-\nu\nabla^2\mathbf{u}=-\frac{1}{\rho}\nabla p+\nabla\cdot\boldsymbol{\tau}, \tag{1}$$

where ν is molecular viscosity and the Reynolds stress term $\boldsymbol{\tau}$ needs to be modeled by a transport equation of the form:

$$\mathbf{u}\cdot\nabla\boldsymbol{\tau}-\nabla\cdot(\nu\nabla\boldsymbol{\tau})=\mathsf{P}-\mathsf{E}, \tag{2}$$

where E indicates dissipation and P includes both production, turbulent transport, and pressure–strain-rate terms. The constitutive PDE (2) implies a nonlocal mapping $g:\mathbf{u}(\boldsymbol{x})\mapsto\boldsymbol{\tau}(\boldsymbol{x})$ from the mean velocity field to the Reynolds stress. Our overarching goal of the data-driven constitutive modeling is to build a neural-network-based surrogate for Eq. (2) that can be trained and used more flexibly with diverse sets of data. In this preliminary work we prove the concept on a problem that is closely related yet simplified in two aspects. First, we build a constitutive neural network for a scalar τ as opposed to the Reynolds stress tensor $\boldsymbol{\tau}$. Examples of scalar quantities in turbulence modeling include turbulent kinetic energy (TKE) and turbulent length scale. In this work, we consider a hypothetical, dimensionless concentration tracer (e.g., volume fraction in multiphase flows) as a scalar τ. Second, the scalar τ is transported by a velocity field $\mathbf{u}$ not affected by the scalar and obtained from laminar flow. Given these two simplifications, hereafter τ can be interpreted as the concentration field of a passive tracer, determined by the resolved variable (velocity $\mathbf{u}$). The transport equation otherwise mimics Eq. (2) and reads:

$$\begin{aligned}&\mathbf{u}\cdot\nabla\tau-\nabla\cdot(C_\nu\nabla\tau)=\mathsf{P}-\mathsf{E}\\ \text{with}\quad&\mathsf{P}=C_g\ell_m\sqrt{\tau}s^2\ \text{and}\ \mathsf{E}=C_\zeta\tau^2\\ &s=\|\mathbf{s}\|=\|\nabla\mathbf{u}+(\nabla\mathbf{u})^\top\|,\end{aligned} \tag{3}$$

where the production P depends on scalar τ, mixing length ℓ_m, and strain-rate magnitude $\|\mathbf{s}\|$. This transport equation resembles that for the TKE transport equation [59] (see Appendix A for detailed analogy between the two equations). The mixing length is proportional to wall distance $\hat{\eta}$ and capped by the boundary layer thickness δ, i.e.,

$$\ell_m=\min(\kappa\hat{\eta},C_\mu\delta) \tag{4}$$

with von Karman constant $\kappa=0.41$ and coefficient $C_\mu=0.09$. Finally, $C_g=200\ \mathrm{m}^{-1}\mathrm{s}$, $C_\nu=0.1\ \mathrm{m}^2/\mathrm{s}$, and $C_\zeta=3\ \mathrm{s}^{-1}$ are coefficients associated with production, diffusion, and dissipation, respectively. Although the PDE above is in a dimensional form with physical units, the inputs of the vector cloud neural network will exclusively use dimensionless quantities (see Table 1 later). As such, training and prediction can be performed in flows of vastly different scales, as long as they are dynamically similar. Such a dynamic similarity is achieved if the coefficients C_μ, C_g, and C_ζ are properly scaled according to the length- and velocity-scales of the system. Details of the similarity are discussed in Appendix B.

Given the problem statement, we construct a constitutive neural network to predict the tracer concentration τ at the point of interest $\boldsymbol{x}_0$ from the flow field information. Considering both the nonlocal physics embodied in the transport PDE (3) and feasibility for implementation in CFD solvers, the network should form a region-to-point mapping $\hat{g}:\langle\mathbf{q}(\boldsymbol{x}_0)\rangle\mapsto\boldsymbol{\tau}(\boldsymbol{x}_0)$, where $\mathbf{q}$ is the feature vector, and $\langle\mathbf{q}(\boldsymbol{x}_0)\rangle$ indicates the collection of features $\{\mathbf{q}_i\}_{i=1}^n$ on n points sampled from the region around $\boldsymbol{x}_0$ (e.g., cell centers in the mesh indicated in dots inside the ovals in Fig. 2a). In general, the number of points n in a cloud can vary from location to location. The feature vector $\mathbf{q}$ at each point is chosen to include the relative coordinate $\boldsymbol{x}'=\boldsymbol{x}-\boldsymbol{x}_0$, flow velocity $\mathbf{u}$, and additional seven scalar quantities $\boldsymbol{c}=[\theta,s,b,\eta,\mathsf{u},r,r']^\top$, including

(1) cell volume θ, which represents the weight for each point in grid-based methods;
(2) velocity magnitude u and strain rate magnitude s, the latter of which often appears in various turbulence and transition models;
(3) boundary cell indicator b and wall distance function η, which help the closure model to distinguish between PDE-described mapping and wall model (boundary condition);

(4) proximity r (inverse of relative distance) to the center point of the cloud and proximity r' defined based on shifting and normalization of projection $-\mathbf{u}^\top \boldsymbol{x}'$ accounting for the alignment between the convection and the relative position of the point in its cloud.
The former is motivated by the fact that features on points closer to $\boldsymbol{x}_0$ have more influences on the prediction $\tau(\boldsymbol{x}_0)$ than those further away for isotropic, diffusion-dominated physics. The latter is justified as the velocity-relative position alignment is important for convective physics: a cell with velocity pointing towards the cloud center $\boldsymbol{x}_0$ is more important than those with velocities perpendicular to or pointing away from the center. Hence we propose r' to be proportional to (i) the time scale $\frac{|\mathbf{u}|}{|\boldsymbol{x}'|}$ and (ii) the cosine of the angle (shifted approximately to the positive range [0, 2]) between the velocity $\mathbf{u}$ and the vector $-\boldsymbol{x}'(=\boldsymbol{x}_0-\boldsymbol{x})$ from the point to the cloud center, i.e., $\epsilon_1 - \frac{\mathbf{u}^\top \boldsymbol{x}'}{|\mathbf{u}||\boldsymbol{x}'|+\epsilon_2}$ with $\epsilon_1 = 1.05$ and $\epsilon_2 = 10^{-10}$ m^2/s.

Note that s and u are directly derived from the velocity $\mathbf{u}$, and the features r and r' can be derived from the relative coordinates (and velocities) of the points on the cloud. While including these derived quantities introduces some redundancy, it helps to guide the construction of the network-based closure model by embedding prior knowledge of physics into the input. We also include two scalar quantities in the input features to represent the boundary conditions: (1) boundary cell indicator b and (2) wall distance function η, both of which reflect the position relative to the wall boundaries. The feature engineering above leads to $\mathbf{q} = [\![\boldsymbol{x}', \mathbf{u}, \boldsymbol{c}]\!]$. The relative coordinates $\boldsymbol{x}' = [x', y']^\top$ and velocity $\mathbf{u} = [u, v]^\top$ are in two dimensional spaces ($d = 2$), but extension to three dimensions is straightforward. With $l' = 7$ scalar quantities as chosen above, the feature vector is $\mathbf{q} \in \mathbb{R}^l$ with $l = 2d + l' = 11$. The features and the associated pre-processing are presented in Table 1. Note that we introduce the characteristic length L_0, flow velocity U_0 and time $T_0 = L_0/U_0$ to non-dimensionalize the features such that all the features are dimensionless before being fed into the neural network, which enables the output of the neural network to be comparable even in systems with different scales. In this work, we take the height of the hill as the characteristic length $L_0 = H = 1$ m and the mean bulk velocity at the inlet patch as the characteristic velocity $U_0 = |\mathbf{u}_b| = 1$ m/s. The characteristic time is then calculated as $T_0 = L_0/U_0 = 1$ s. By the definition in Table 1, the relative coordinate features $\boldsymbol{x}' = [x', y']^\top$ are essentially the cosine and sine of the angle between the relative direction and the streamwise direction, and thus should be more precisely referred to as *relative direction*. However, hereafter it is referred to as (normalized) "relative coordinates" according to the raw features.

In light of the frame-independence requirements, our first attempt is to seek a mapping of the form $\tau = \hat{g}(\mathcal{Q}\mathcal{Q}^\top)$ with $\mathcal{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_n]^\top$. Such a function has both translational and rotational invariance, because it depends only on the relative coordinates $\boldsymbol{x}_i'$ and relative orientations of $\mathbf{q}_i$ in the form of pairwise projections $\mathcal{Q}\mathcal{Q}^\top$ among the feature vectors. However, it lacks permutational invariance as it depends on the numbering (ordering) of the n cloud points. This is more evident when we write the projection in index form $\mathcal{D}'_{ii'} = \sum_{j=1}^{11} \mathcal{Q}_{ij}\mathcal{Q}_{ji'}$, where i and i' are cloud point indices, and j is feature index. Matrix $\mathcal{D}'$ would switch rows and columns if the n points are permutated (i.e., the numbering is switched). Therefore, we need to define a function that depends only on the set of vectors $\{\mathbf{q}_i\}_{i=1}^n$ and not on its ordering. To this end, we introduce a set of m functions $\{\phi_k(\boldsymbol{c}_i)\}_{k=1}^m$ for the scalar quantities $\boldsymbol{c}_i$ (note the input $\boldsymbol{c}_i = [\theta_i, s_i, \ldots]^\top$ represents all the scalar quantities for the ith point, which is part of the feature vector $\mathbf{q}_i$ and already has translational and rotational invariance) and define $\mathcal{L}_{kj} = \frac{1}{n}\sum_{i=1}^n \phi_k(\boldsymbol{c}_i)\,\mathcal{Q}_{ij}$, where $k = 1, \ldots, m$ is the function index. The summation over the point index i removes the dependence of $\mathcal{L}$ on the ordering and make it permutational invariant. If we define a matrix $\mathcal{G}_{ik} = \phi_k(\boldsymbol{c}_i)$, the order-removing transformation above can be written in matrix form as $\mathcal{L} = \frac{1}{n}\mathcal{G}^\top\mathcal{Q}$, where the normalization by n allows training and prediction to use different numbers of sampled points in the cloud. This mapping $f_{\text{embed}} : \mathcal{C} \mapsto \mathcal{G}$ with $\mathcal{C} = [\![\boldsymbol{c}_1^\top, \boldsymbol{c}_2^\top, \ldots, \boldsymbol{c}_n^\top]\!]$ is implemented as an embedding neural network with m-dimensional output, interpreted as applying the m functions $\{\phi_k(\boldsymbol{c}_i)\}_{k=1}^m$ to the frame-independence part $\boldsymbol{c}_i$ of the feature vector $\{\mathbf{q}_i\}$ at point i. This is illustrated in Fig. 2b. Similarly, we can define $\mathcal{L}^\star = \frac{1}{n}\mathcal{G}^{\star\top}\mathcal{Q}$, with $\mathcal{G}^\star$ being the first m' columns of $\mathcal{G}$, and $\mathcal{L}^\star$ is also permutational invariant. Here we choose $\mathcal{G}^\star$ as a subset of $\mathcal{G}$ rather than $\mathcal{G}$ itself mainly for the purpose of saving the computational cost without sacrificing the accuracy. Next, we define a projection mapping $\mathcal{D} = \mathcal{L}\mathcal{L}^{\star\top} = \frac{1}{n^2}\mathcal{G}^\top\mathcal{Q}\mathcal{Q}^\top\mathcal{G}^\star$, then $\mathcal{D}$ has translational, rotational, and permutational invariance (since both $\mathcal{L}$ and $\mathcal{L}^{\star\top}$ have permutational invariance while the pairwise projection $\mathcal{L}\mathcal{L}^{\star\top}$ achieves rotational invariance); it retains all information from the feature vectors on the cloud via $\mathcal{Q}\mathcal{Q}^\top$ by projecting it onto the learned basis $\mathcal{G}$. Finally, we fit a function $f_{\text{fit}} : \mathcal{D} \mapsto \tau$ that maps $\mathcal{D}$ to the closure variable τ, which is achieved through a fitting network shown in Fig. 2c. The combination of the embedding network, the linear mapping, and the fitting network thus forms the complete constitutive network that serves as the closure for the primary equation.

In summary, the data-driven, frame-independent, nonlocal constitutive model $\hat{g}: \mathcal{Q} \mapsto \tau$ that maps the features for points on a cloud to the closure variable is achieved in the following four steps:

(1) Feature engineering to stack relative coordinate $\boldsymbol{x}'$, velocity $\mathbf{u}$, and scalar quantities $\boldsymbol{c}$ (each row for a point in cloud):

$$\mathcal{Q} = \begin{bmatrix} \mathbf{q}_1^\top \\ \mathbf{q}_2^\top \\ \vdots \\ \mathbf{q}_n^\top \end{bmatrix} = \left[\begin{array}{cc|cc|ccccccc} x_1' & y_1' & u_1 & v_1 & \theta_1 & s_1 & b_1 & \mathsf{u}_1 & \eta_1 & r_1 & r_1' \\ x_2' & y_2' & u_2 & v_2 & \theta_2 & s_2 & b_2 & \mathsf{u}_2 & \eta_2 & r_2 & r_2' \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n' & y_n' & u_n & v_n & \theta_n & s_n & b_n & \mathsf{u}_n & \eta_n & r_n & r_n' \end{array}\right] = [\mathcal{X}, \mathcal{U}, \mathcal{C}] \in \mathbb{R}^{n\times 11}, \tag{5}$$

where $\mathbf{q}^\top = [\boldsymbol{x}'^\top, \mathbf{u}^\top, \boldsymbol{c}^\top]$, and $\mathcal{X}, \mathcal{U}, \mathcal{C}$ correspond to the three submatrices of $\mathcal{Q}$ indicated by separators.

(2) Embedding via a neural network $f_{\text{embed}}: \mathcal{C} \mapsto \mathcal{G}$ (using only scalar quantities of the feature vector) to remove the dependence on point numbering:

$$\mathcal{G} = \left[\begin{array}{ccccc|ccc} \phi_1(\boldsymbol{c}_1) & \phi_2(\boldsymbol{c}_1) & \dots & \phi_{m'}(\boldsymbol{c}_1) & \phi_{m'+1}(\boldsymbol{c}_1) & \dots & \phi_m(\boldsymbol{c}_1) \\ \phi_1(\boldsymbol{c}_2) & \phi_2(\boldsymbol{c}_2) & \dots & \phi_{m'}(\boldsymbol{c}_2) & \phi_{m'+1}(\boldsymbol{c}_2) & \dots & \phi_m(\boldsymbol{c}_2) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{c}_n) & \phi_2(\boldsymbol{c}_n) & \dots & \phi_{m'}(\boldsymbol{c}_n) & \phi_{m'+1}(\boldsymbol{c}_n) & \dots & \phi_m(\boldsymbol{c}_n) \end{array}\right] \in \mathbb{R}^{n\times m}, \tag{6}$$

and $\mathcal{G}^\star \in \mathbb{R}^{n\times m'}$ is the first m' ($\leq m$) columns of $\mathcal{G}$ (left of the separator line of $\mathcal{G}$).

(3) Combining $\mathcal{Q}\mathcal{Q}^\top$ with $\mathcal{G}$ and $\mathcal{G}^\star$ in a projection mapping to obtain a feature matrix with translational, rotational, and permutational invariance:

$$\mathcal{D} = \frac{1}{n^2}\mathcal{G}^\top \mathcal{Q}\mathcal{Q}^\top \mathcal{G}^\star \in \mathbb{R}^{m\times m'}. \tag{7}$$

(4) Fitting a neural network f_{fit} to map the frame-independent input $\mathcal{D}$ to closure variable τ:

$$\tau(\boldsymbol{x}_0) = f_{\text{fit}}(\mathcal{D}). \tag{8}$$

In this work we choose $m = 64$ embedding functions and a subset of $m' = 4$ in the extraction of feature matrix $\mathcal{D}$. The choice of m, m' is similar to that in Deep Potential [55], which gives good performance in the prediction of the potential energy surface in practice. In our problem, we find the prediction performance is insensitive to m' when m' is much less than m. We set $m' = 4$ because more features are preserved in the invariant feature matrix but the computational cost for the fitting network is still relatively small. The problem statement of constructing the data-driven constitutive model and the complete procedure is summarized in Fig. 2. A more general presentation of the framework and the proof of its invariance properties are presented in Appendix C. Furthermore, a minimal example illustrating all the invariance properties is provided in Appendix D.

3. Results

In this study, we consider the flow over periodic hills. This case is representative as a benchmark for RANS simulations of flow separation due to the wall curvature [60]. The baseline geometry is shown in Fig. 3a with the hill shape described as piecewise polynomials. In the baseline geometry, the width and height of the hill are $w = 1.93$ m and $H = 1$ m, while the length and the height of the domain are $L_x = 9$ m and $L_y = 3.035$ m. The slope parameter α is varied to generate a family of geometries for training and testing [61], which is presented in Fig. 3b. The parameter α describes the steepness of the hill profile, which equals the ratio of the width w of the parameterized hill to that of the baseline hill ($w|_{\alpha=1}$). The hill height is kept the same for all geometries and thus the hill becomes more gentle with increasing α. For training flows, we choose 21 configurations with $\alpha = 1.0, 1.05, \dots, 2$. For testing flows, the parameters α range from 0.5 to 4.

3.1. Generation of training data

We calculate the concentration field $\tau(\boldsymbol{x})$ by solving the transport PDE (3) with a steady, laminar flow field $\mathbf{u}(\boldsymbol{x})$. The flow is driven by a constant pressure gradient such that the Reynolds number based on bulk velocity at the

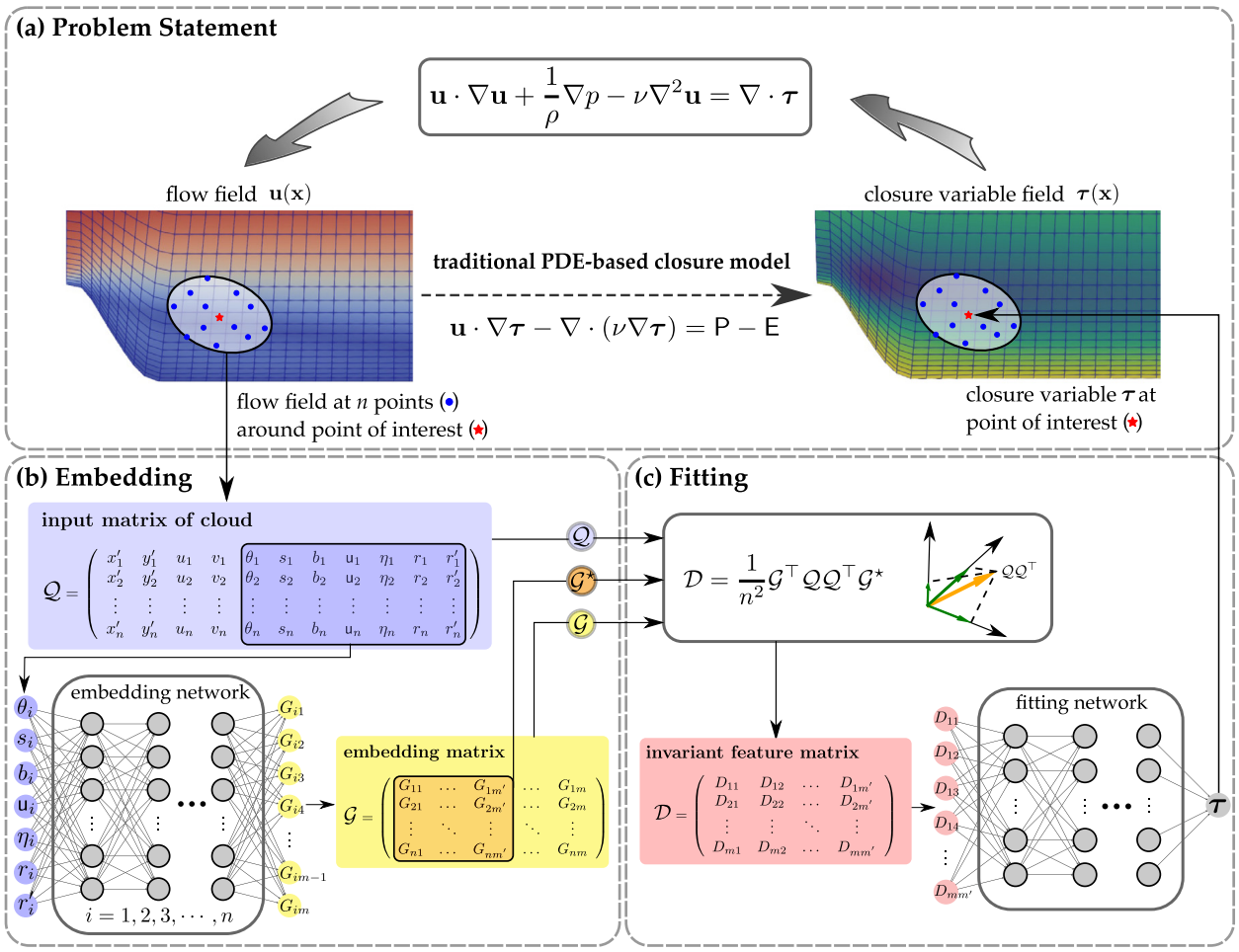
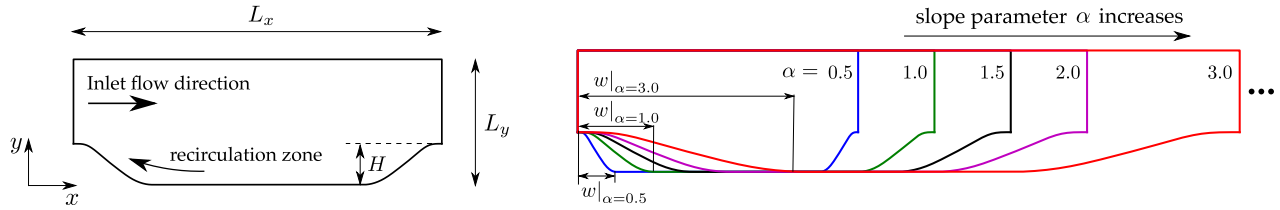


Fig. 2. Detailed schematic of the vector-cloud neural network to provide nonlocal closure model $\boldsymbol{\tau}(\mathbf{u})$ for the primary equation (the Reynolds-averaged Navier–Stokes equation is used here for illustration): (a) generate labeled training data $(\mathcal{Q},\boldsymbol{\tau})$, where the input matrix $\mathcal{Q}\in\mathbb{R}^{n\times l}$ consists of n vectors, each attached to a point (•) in the cloud (⬭) surrounding the point (★) where the closure variable $\boldsymbol{\tau}$ is to be predicted; each vector (a row in $\mathcal{Q}$) encodes its relative coordinate $\boldsymbol{x}'$, velocity $\mathbf{u}$, and other scalar features (Table 1); (b) map the scalar features in $\mathcal{Q}$ through an embedding network to a set of invariant bases $\mathcal{G}\in\mathbb{R}^{n\times m}$, of which $\mathcal{G}^\star\in\mathbb{R}^{n\times m'}$ is the first m' (≤m) columns; then, project the pairwise inner-product matrix $\mathcal{Q}\mathcal{Q}^\top$ to the learned embedding matrix $\mathcal{G}^\top$ and its submatrix $\mathcal{G}^\star$ to yield an invariant feature matrix $\mathcal{D}\in\mathbb{R}^{m\times m'}$; and (c) flatten and feed the feature matrix $\mathcal{D}$ into the fitting network to predict the closure variable $\boldsymbol{\tau}$. The constitutive mapping $\mathbf{u}(\mathbf{x})\mapsto\boldsymbol{\tau}$ based on the vector-cloud neural network is invariant to both frame translation and rotation and to the ordering of points in the cloud. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) baseline geometry (b) geometries of varying slopes

Fig. 3. Geometries of flow domains used for training and testing, showing (a) baseline geometry and (b) geometries with varying slopes, parameterized by the ratio α of the width w of hill to that of the baseline hill, while the hill height $H=1$ m is the same for all geometries. The x- and y-coordinates are aligned in the streamwise and wall-normal directions, respectively. By definition the baseline geometry has $\alpha=1$. The slope parameters α for training and testing are within the ranges [1, 2] and [0.5, 4], respectively.

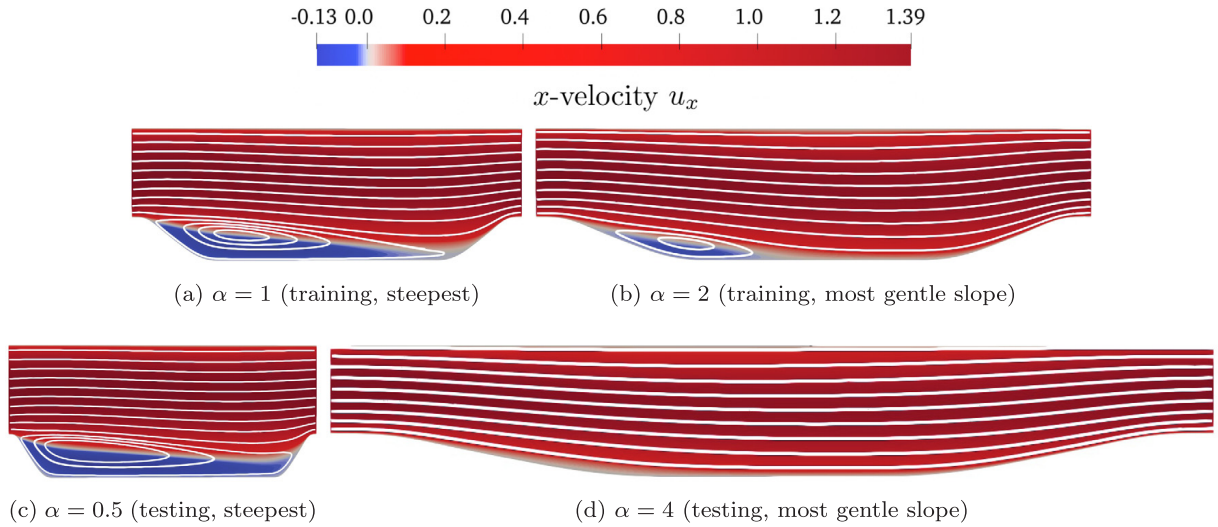


Fig. 4. Flow velocity fields used to solve the transport equation for generating data, showing streamlines from representative flow fields in the training and testing datasets. Top: (a) the steepest slope $\alpha = 1$ and (b) the most gentle slope ($\alpha = 2$) in **training** flows. Bottom: (c) the steepest slope $\alpha = 0.5$ and (d) the most gentle slope ($\alpha = 4$) in **testing** flows, which are steeper and more gentle, respectively, than (a) and (b), the slope range of training flows. The color contours of streamwise velocity u_x highlight the recirculation zones with blue/dark gray, showing the regions with flow reversal (present only in panels a–c).

crest $|\mathbf{u}_b| = 1$ m/s and the hill height $H = 1$ m reaches a specified value $Re = 100$. All the numerical simulations are performed in the open-source CFD platform OpenFOAM [62]. We first simulate the steady-state flows over the periodic hill by solving the incompressible Navier–Stokes equations with the built-in solver simpleFoam. The coupled momentum and pressure equations are solved by using the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) algorithm [63]. Then, Eq. (3) is solved to simulate the steady-state concentration field $\tau(\boldsymbol{x})$ with the obtained stationary velocity field to provide data for training and testing as detailed below. When solving the fluid flow and concentration equations, at both walls nonslip boundary conditions ($\mathbf{u} = 0$) are used for the velocities and Dirichlet boundary conditions ($\tau = 0$) for the concentration, while cyclic boundary conditions are applied at the inlet and the outlet. The equations are numerically discretized on an unstructured mesh of quadrilateral cells with second-order spatial discretization schemes. In the wall normal direction all configuration has 200 cells that are refined towards both the upper and bottom walls. In the streamwise direction the number of cells is approximately proportional to the length L_x of the domain (200 cells in the baseline geometry with $\alpha = 1$ and $L_x = 9$ m). The boundary layer thickness length scale is set to $\ell_\delta = 1.5$ m for the calculation of mixing length in Eq. (3).

Four representative flow fields obtained from simpleFoam are displayed in Fig. 4, showing the streamlines for the cases with limiting slopes in the training flows (Fig. 4a and b for $\alpha = 1$ and 2, respectively) and testing flows (Fig. 4c and d for $\alpha = 0.5$ and 4, respectively). The backgrounds of the plots highlight the recirculation zones with contours of the streamwise velocities u_x with blue (darker gray) near the bottom wall showing the regions with flow reversal. The testing flows are intentionally chosen to span a wide range of flow patterns ranging from massive separation ($\alpha = 0.5$ with the recirculation zone extending to windward of the hill) to mild separation ($\alpha = 2$ with recirculation bubble at the leeward foot of the hill) and fully attached flow ($\alpha = 4$). Such a wide range of flow patterns is expected to pose challenges to the closure modeling from flow field to concentration.

We aim to learn a nonlocal mapping from a patch of flow field $\langle \mathbf{u}(\boldsymbol{x}) \rangle$ to the concentration τ at the point of interest. The procedure of generating such pairs of data $(\mathcal{Q}, \tau)$ is illustrated in Fig. 5, where $\mathcal{Q}$ is the input feature matrix derived from the patch of flow field $\langle \mathbf{u}(\boldsymbol{x}) \rangle$. In Fig. 5, the cell centers on which the flow fields are defined are indicated as gray dots (•), showing only every seventh row and third column for clarity; the highlighted point (⋆) indicates the location where the concentration is predicted, and the ellipse (⬭) denotes the surrounding vector cloud; n data points (•) are randomly sampled within the cloud for the prediction of concentration τ. The extent of the cloud is determined by the velocity $\mathbf{u}$ at the point of interest according to region of physical influence. The major axis of the ellipse aligns with the direction of the velocity $\mathbf{u}$. The half-lengths ℓ_1 and ℓ_2 of the major and

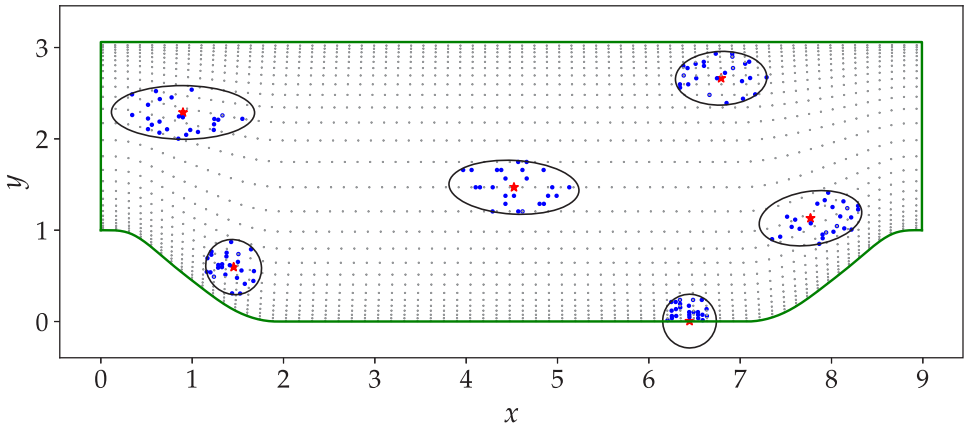


Fig. 5. Method of sampling data points to generate labeled training data mapping a vector cloud $\mathcal{Q}$ to the closure variable τ. The gray dots (•) indicate cell centers, showing only every seventh row and third column for clarity. The surrounding cloud is depicted as ellipses (⬭), whose size and orientation are determined by the velocity at the cloud center (⋆) according to the region of physical influence. The cloud centers (⋆) indicate the locations where the concentration τ is to be predicted. The blue/darker gray dots (•) are randomly sampled from the cell centers (•) within the cloud, and the feature vectors attached to them are used as input matrix $\mathcal{Q}$ to predict τ. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

minor axis are determined based on the specified relative error tolerance ϵ according to [22]:

$$\ell_1 = \left| \frac{2\nu \log \epsilon}{\sqrt{|\mathbf{u}|^2 + 4\nu\zeta} - |\mathbf{u}|} \right| \quad \text{and} \quad \ell_2 = \left| \sqrt{\frac{\nu}{\zeta}} \log \epsilon \right|, \tag{9}$$

where ν and ζ are the coefficients for diffusion and dissipation terms in the one-dimensional steady-state convection–diffusion–reaction equation $-\nu \frac{\mathrm{d}^2}{\mathrm{d}x^2}\tau(x) + u\frac{\mathrm{d}}{\mathrm{d}x}\tau(x) + \zeta\tau(x) = P(x)$, and the detailed derivation can be found in [22]. In principle, the actual dissipation coefficient ζ in Eq. (9) varies in space as $C_\zeta \tau$, considering the dissipation term $\mathsf{E} = C_\zeta \tau^2$. Here, we assume it to be constant $\zeta \approx C_\zeta \tau_{\max}$ to control the size of clouds, in which $\tau_{\max}$ is obtained by taking the maximum value from the steady-state concentration field. The diffusion coefficient ν in Eq. (9) is chosen as the coefficient C_ν in Eq. (3).

For convenience, we set n, the number of sampled data points in a cloud, to 150 in all training data, which gives good predictions in our problems. It should be noted that, the constructed region-to-point mapping is applicable to data with different numbers of sampled data points within the cloud. We set it to be constant in the training data to process them in batch with better computation efficiency. Given a fixed number of sampled data points within a cloud, the sampling method influences the performance of the trained model. Although the points closer to the center usually have a larger impact on the center due to the underlying convection–diffusion process, all the points in the elliptical region may have non-negligible contributions to the final prediction. Always choosing the closest data points will indirectly reduce the size of the influence region, which may lose the effective nonlocal mean flow information and undermine the prediction performance [22]. Instead, we use the uniform random sampling method to sample n data points from the cloud. In other words, we treat each data point equally a priori and let the embedding network learn the importance of each point implicitly. In testing, we showcase the flexibility of our trained networks to the input data with different numbers of sampled points in the cloud by using all the data points within the cloud, whose number varies at different locations, to predict the concentration at the points of interest. In total, we generated 1.26×10^6 pairs of $(\mathcal{Q}, \tau)$ as training data, which takes around 6 h on a single CPU. Note that we can use fewer pairs for training since the adjacent data pairs from the same geometry appear similar. In this study, we choose all the pairs to make full use of the generated data. The proposed neural network is implemented and trained by using the machine learning library PyTorch [64], which takes around 16.7 h on an NVIDIA GeForce RTX 3090 GPU. We verified that the constructed constitutive neural network does ensure all the expected (translational, rotational, and permutational) invariance. The detailed architecture for the embedding and fitting networks as well as the parameters used for training are presented in Appendix E. The code developed and data generated for this work are made publicly available on GitHub [65].

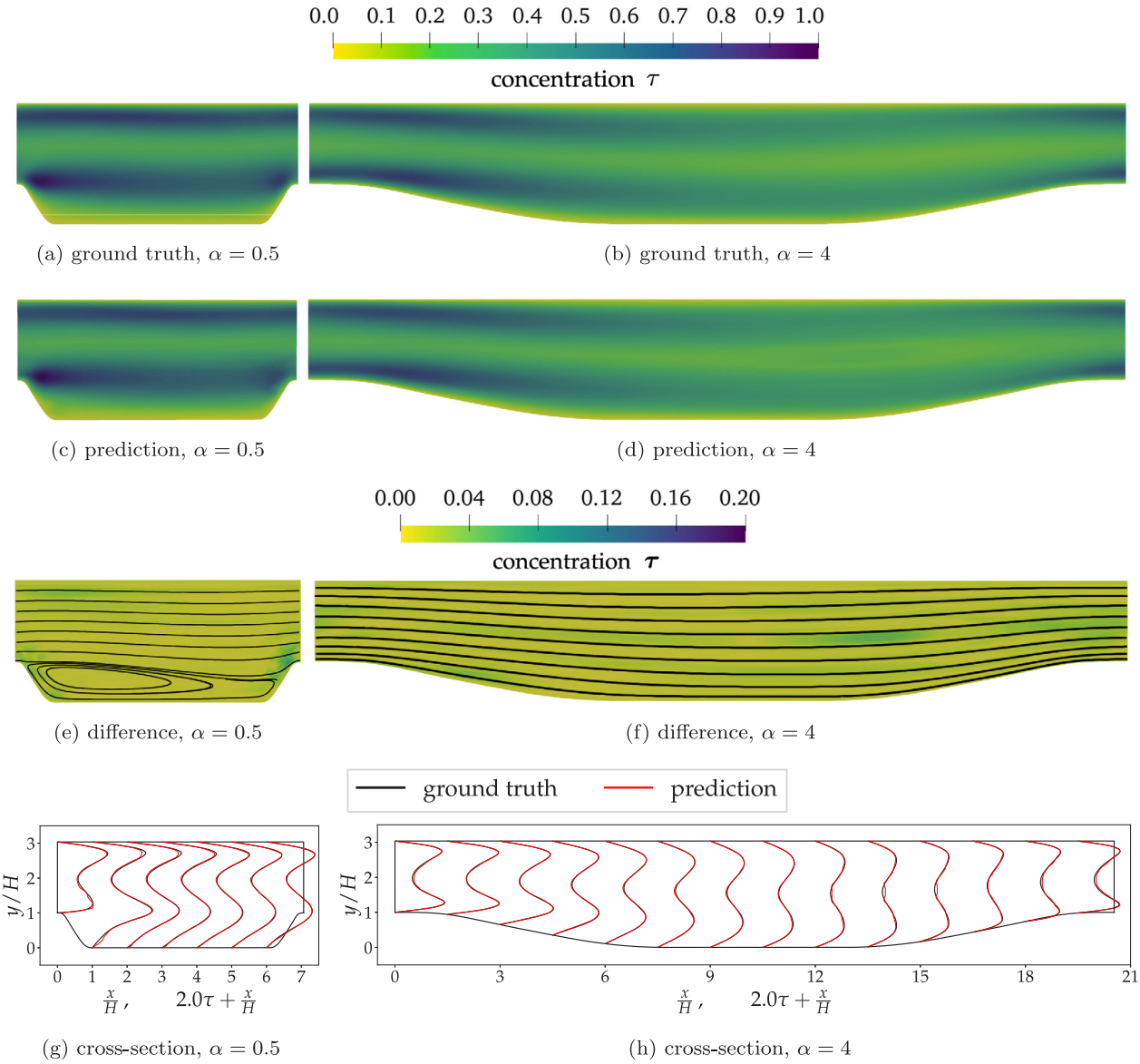


Fig. 6. Comparison of the ground truths of the concentration field τ (top row), the corresponding predictions with the trained constitutive neural network (second row) and the differences between the ground truths and the predictions (third row) along with the concentration plots in seven and fourteen cross-sections (bottom row) for two configurations with slope parameters $\alpha = 0.5$ (left panels) and $\alpha = 4.0$ (right panels). The neural network is trained with data from 21 configurations with $\alpha = 1, 1.05, 1.1, \ldots, 2$.

3.2. Neural-network-based prediction of concentrations

After training, the predicted concentration fields provided by the trained constitutive network are quite close to the ground truths. The comparison of the predicted concentration fields and the corresponding ground truths in two extreme configurations with $\alpha = 0.5$ and 4 is shown in Fig. 6. The two flow fields have distinctive patterns in that there is a large recirculation zone for $\alpha = 0.5$ as shown in Fig. 4. According to Fig. 6a–e, we can see that although neither field is present in the training data, both predicted concentration fields are similar to the corresponding ground truths. The accuracy is visualized more clearly in Fig. 6g, the comparison of the predicted and true concentration on seven vertical cross-sections at $x/H = 0, 1, \ldots, 6$ when $\alpha = 0.5$. The same comparison is presented in Fig. 6h for the case $\alpha = 4$ on fourteen cross-sections. From Fig. 6e and g, we can see that the predicted concentrations near both the inlet and outlet show slight inconsistency with the ground truths in the case $\alpha = 0.5$, which may be caused by the flow separation due to the sharp hill. According to Fig. 6f and h, however, the inconsistency is not

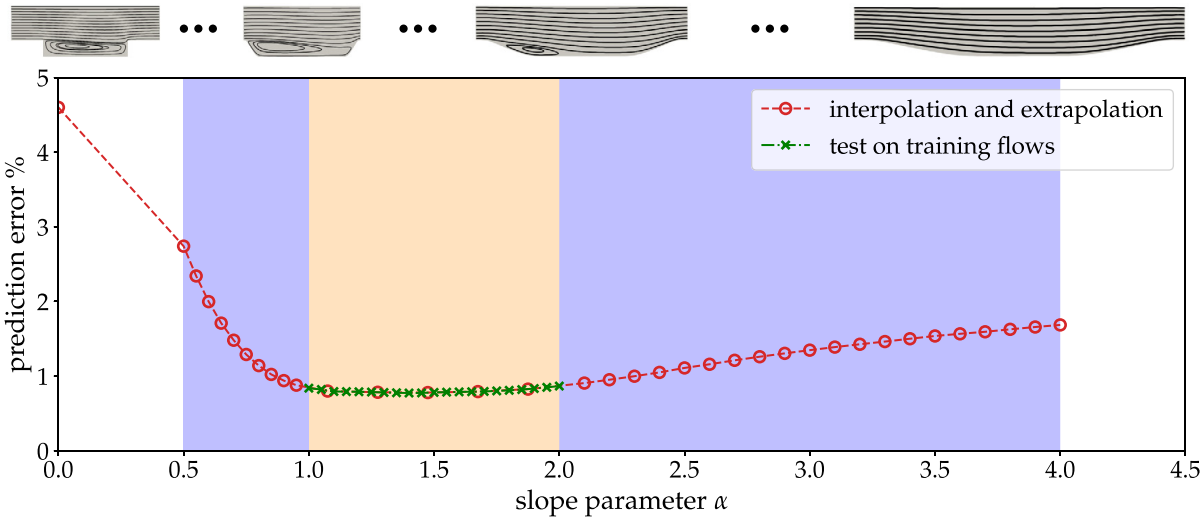


Fig. 7. General predictive errors at various slope parameters α. The yellow/lighter and blue/darker backgrounds represent the regimes of the slope parameters α of training and testing flows, respectively. The neural network is trained with data from 21 configurations with $\alpha = 1, 1.05, 1.1, \ldots, 2$. The trained network is then tested on five configurations with $\alpha = 1.075, 1.275, \ldots, 1.875$ in the training regime and 30 configurations with $\alpha = 0.5, 0.55, \ldots, 0.95$ and $\alpha = 2.1, 2.2, \ldots, 4$ in the testing regime. The thumbnails on the top indicate the variation of geometry and flow field with slope parameter α increasing from 0.5 to 4. We also tested a configuration with a sudden expansion and contraction, roughly corresponding to $\alpha = 0$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

concentrated near the hills in the case $\alpha = 4$ since the slope is very gentle. The slight inconsistency in the middle channel near $x/H \approx 15$ is mainly caused by the difference from the training flows, i.e., the main stream here is being contracted along a longer hill while it is less contracted in the training flows due to the shorter hills and the existence of the recirculation zones.

We use 35 configurations with varying slope parameters α to evaluate the prediction performance of the trained neural network. The prediction errors of testing flows are shown in Fig. 7. The normalized prediction error is defined as the normalized pointwise discrepancy between the predicted concentration $\hat{\tau}$ and the corresponding ground truth τ^*:

$$\text{error} = \frac{\sqrt{\sum_{i=1}^{N} |\hat{\tau}_i - \tau_i^*|^2}}{\sqrt{\sum_{i=1}^{N} |\tau_i^*|^2}}, \tag{10}$$

where the summation is performed over all of the N training or testing data points. The in-between (yellow/lighter gray) region and the whole region represent the regimes of slope parameter α of training flows and testing flows, respectively. We can see that the prediction errors of interpolated testing flows with $\alpha = 1.075, 1.275, \ldots, 1.875$ are the lowest, which is expected because the flow fields in these configurations are very close to that of the training flows. For extrapolated testing flows, the prediction error grows as α departs from the range 1–2 in the training data (yellow/lighter gray). Specifically, as α approaches 4 the recirculation zone gradually shrinks and eventually disappears, and the prediction error grows. In contrast, when α decreases to 0.5, the whole bottom region becomes the recirculation zone, and the prediction error increases relatively fast due to the massive flow separation that is dramatically different from those in the training flows. Further, we tested the trained network on a geometry with a sudden expansion at $x/H = 1.93$ and contraction at $x/H = 7.07$ (roughly corresponding to $\alpha = 0$). The prediction error was 4.76%, which was higher than that for $\alpha = 0.5$ but still reasonable. In addition, we also performed testing on 21 training flows. Note that in this testing step all the data points in a cloud (approximately 300 to 1500 points) are used for prediction, although only 150 points within each cloud were used in training. The prediction errors on 21 training flows are all around 1%, which is close to the training error 0.62%. This shows that the proposed nonlocal constitutive neural network not only guarantees permutational invariance *formally*, but also approximates the underlying PDE mapping with adequate accuracy and great flexibility in terms of the number and locations of the used points.

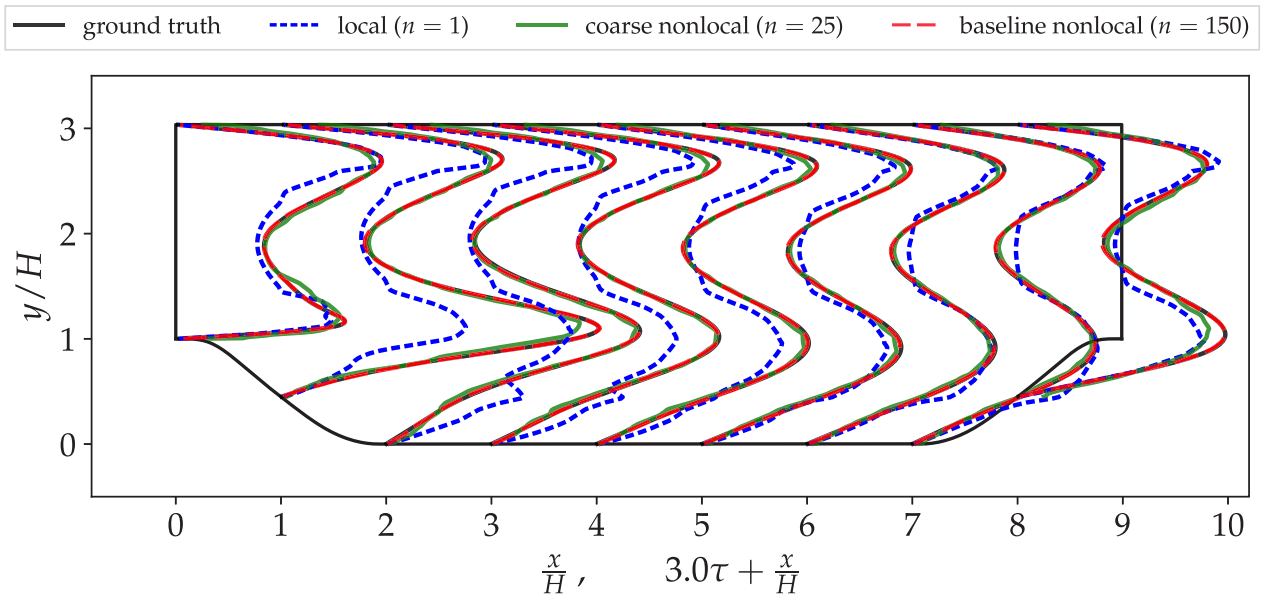
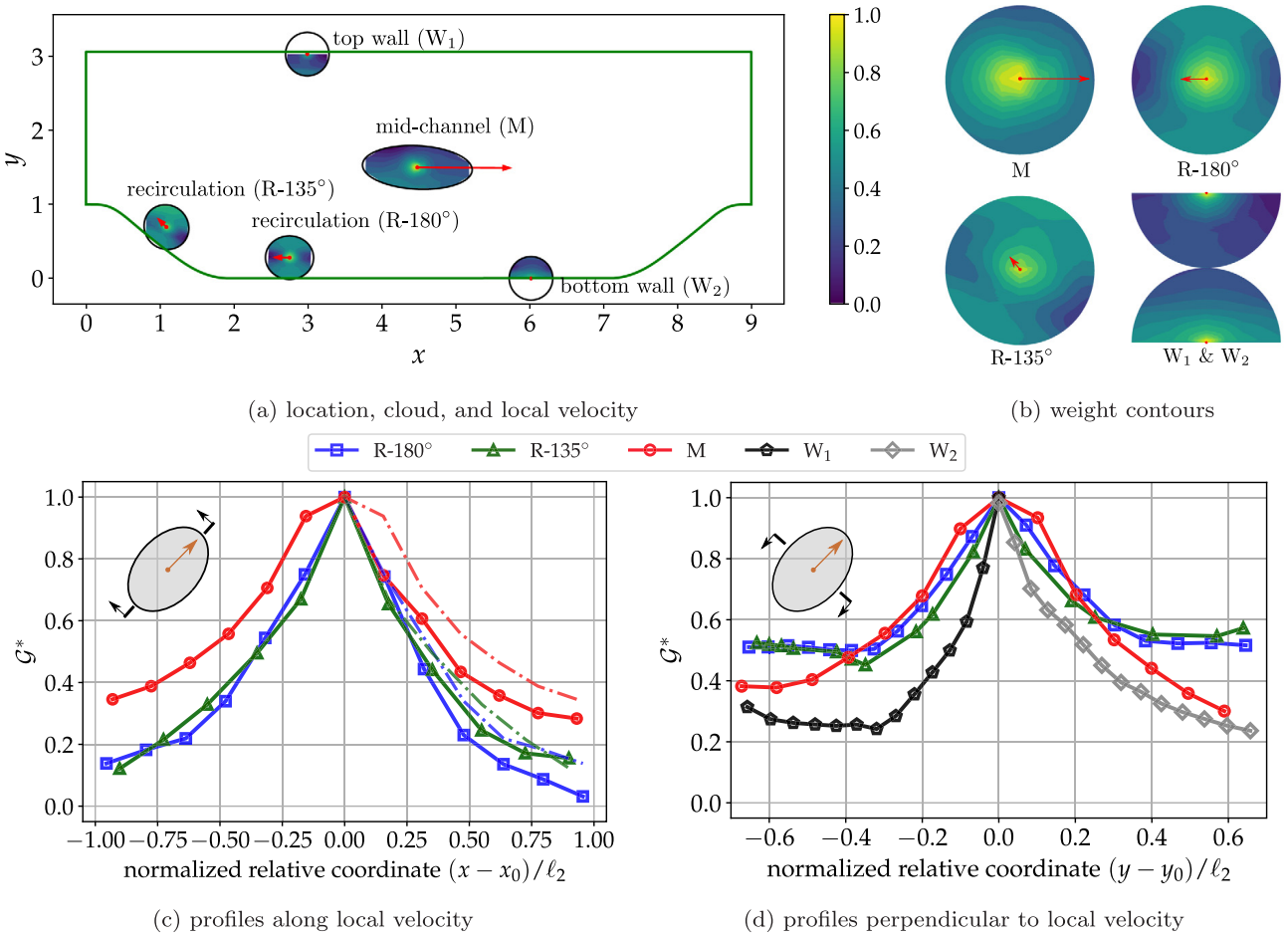


Fig. 8. Ablation study results showing a comparison of the predicted concentrations on nine cross-sections in the flow with $\alpha = 1$ by using different models, including a local model ($n = 1$; blue dotted lines) and nonlocal models with a coarse sampling ($n = 25$ points per cloud; green/light gray solid lines) and a baseline sampling ($n = 150$ points; red dashed lines). The results from $n = 150$ are visually indistinguishable from the ground truth. The neural networks are trained with data from 21 configurations with $\alpha = 1, 1.05, 1.1, \ldots, 2$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

While we use $n = 150$ points per cloud to achieve optimal results, our ablation studies show that (1) using a nonlocal model with coarser cloud sampling causes deteriorated performance due to the sampling error, and (2) using a local model would diminish the predictive capability of the model as it completely misses the nonlocal physics in the data. To this end, we investigate two levels of ablated models: a coarse nonlocal model and a local model. First, we study a coarse nonlocal model by randomly choosing $n = 25$ data points within the clouds and train the neural network with such generated data. The trained neural network is then tested based on all the data points within the clouds. In a further ablation, we evaluate a local model with $n = 1$ point, i.e., only the data point at the center is used for training (without information from any neighboring points). In the latter case, the local constitutive model as represented by the neural network no longer has pairwise inner productions. As a result, only the scalar quantities $\boldsymbol{c} = [\theta, s, b, \mathsf{u}, \eta]^\top$ attached to the center point are used for training and they are directly fed into the fitting network (Fig. 1c). An embedding network (Fig. 1b) is thus omitted in the local model. Accordingly, the trained neural network is tested by feeding the scalar quantities $\boldsymbol{c}$ associated with the data point itself and not the cloud. The comparison of the three different models is shown in Fig. 8. It can be seen that the predicted profiles from the local model ($n = 1$, blue/dark gray dotted lines) deviate significantly from the ground truth in most regions. In contrast, the prediction of nonlocal models (green and red dashed lines) are almost indistinguishable from the ground truth, although the sparsely sampled model (with $n = 25$ points per cloud) deviates from the truth in a few places. For a quantitative comparison, the local model does not perform well and the prediction error rate is much larger (19.2%) even when compared to the error rate 7.65% of the sparsely sampled nonlocal model. This is expected as the local model completely misses the nonlocal physics due to convection and diffusion. The prediction error is further reduced to 0.84% when a denser sampling $n = 150$ is used in the nonlocal model.

3.3. Physical interpretation of learned model

The learned network implies a weight (contribution) for each point in any given cloud, and it is desirable to shed light on such a weight and its dependence on the location of the point in the cloud. Physical intuition dictates that the weight shall be skewed towards in the upwind direction, and that such a skewness shall increase (diminish) as the local velocity increases (diminishes). To verify such a trend, we chose the embedded basis $\mathcal{G}^\star \in \mathbb{R}^{n \times m'}$ as a proxy of the weight for each point. Furthermore, we set $m' = 1$ so that there is exactly one number associated



(a) location, cloud, and local velocity

(b) weight contours

(c) profiles along local velocity

(d) profiles perpendicular to local velocity

Fig. 9. Weight $\mathcal{G}^\star$ for each point in the cloud as learned by the neural network in the case of $m' = 1$, showing skewness towards the upwind direction as expected from physical intuition. Weight contours are shown for five points in three representative regions: M in the mid-channel, R-135° and R-180° in the recirculation zone, and W_1 and W_2 near the walls. (a) Locations, clouds (indicated in ovals), and local velocities (indicated in red/gray arrows) for each point; (b) Weight contours zoomed to the circle of radius ℓ_2; (c) Profiles along the local velocity direction, with the upstream mirrored to downstream as dash-dotted lines; (d) Profiles perpendicular to local velocity direction. Note that the weight $\mathcal{G}^\star$ is normalized by the maximum $\mathcal{G}^\star$ in the cloud such that the plotted weight has a maximum of 1. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

with each of the n points. We chose five points in three representative regions for visualization: (i) a point M in the mid-channel, (ii) two points R-135° and R-180° in the recirculation zone (named according to their respective velocity angles in reference to the streamwise direction), (iii) two points W_1 and W_2 near the top and bottom walls, respectively.

The contours of the weights showed a trend that is consistent with the physical intuition above. The locations, clouds and local velocities of these points are shown in Fig. 9a. The weight contours (showing only a circle of radius ℓ_2 around the point) are presented in Fig. 9b, where the weights in the upwind regions are clearly larger for the cloud of the mid-channel point M, while for other points it is less prominent due to their smaller velocities. To show the trend more clearly, we present two cross-sections in Fig. 9c (along the local velocity) and Fig. 9d (perpendicular to the local velocity direction). In Fig. 9c, the upwind region (negative relative coordinates $x - x_0$) is mirrored to the downwind region and denoted in dash-dotted lines of the same color. This comparison clearly shows that the upwind region has larger weights. Our parametric studies further showed that such a trend of upwind skewness was even more evident when the diffusion coefficient C_ν was reduced when generating the training data (figures omitted here for brevity). In contrast, the skewness is not present in the profiles perpendicular to the local

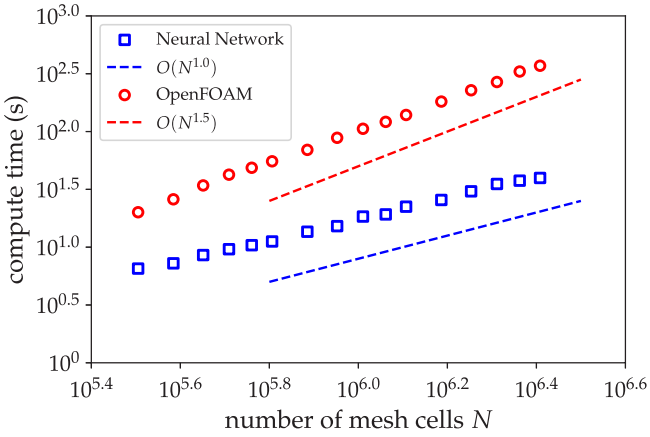


Fig. 10. Comparison of computational time and its scaling with problem size (number of cells N in the mesh) between vector-cloud neural network and traditional PDE solver (finite volume method in OpenFOAM). Different cases correspond to periodic hills with the slope parameter α ranging from 0.5 to 4.

velocity as shown in Fig. 9d, which is expected since there is no convection in this direction. The general trend of the weights for the near wall points W_1 and W_2 are similar to those for the in-stream points (M, R-135°, and R-180°) because they must emulate the same differential operator. On the other hand, the weights for the near wall points W_1 and W_2 decrease more rapidly from center to periphery than those for the in-stream points (M, R-135°, and R-180°), meaning that the learned weights for points near the boundaries are significantly larger than those far from the boundaries. Such a difference in the trend is desirable: the embedding network needs to also emulate a wall-function implicitly embedded in the mixing length model in Eq. (4) for the near-wall points. In summary, the neural network correctly learned from the data the contribution of each point in the cloud.

3.4. Discussion

Besides the physical interpretability, by design the proposed vector-cloud neural network has some other merits that make it suitable for nonlocal closure modeling. On the training side, it is flexible with a wide range of data, which facilitates the process of data collection. Training samples are well defined if we only have data in a region rather than the whole field or the spatial meshes are different in different regions. On the inference side, the computational cost of predicting the tracer concentration of the entire field is proportional to the total number of mesh cells N, given a fixed number of sampled points in the cloud. Due to the lack of well-established machine-learning-based models for the closure modeling, we compare the computational complexity of the vector-cloud neural network with traditional PDE solver (finite volume method in OpenFOAM) on the periodic hills with the slope parameter α ranging from 0.5 to 4. The results are shown in Fig. 10. The linear scaling $O(N)$ of the vector-cloud neural networks is confirmed. In contrast, although the scaling of the traditional finite volume method inevitably depends on the specific linear solver used (e.g., direct solver, conjugate gradient, or multigrid method), it can never achieve linear scaling. In our test, the scaling of the OpenFOAM solver is approximately $O(N^{1.5})$. Such scaling, however, is never constant. Solving constitutive PDEs can be accelerated when coupled with RANS equations since the constitutive PDEs have better initial conditions from the last iteration. But for real industrial flows, solving the PDEs in constitutive models is very likely to be expensive and time-consuming while using the neural network based constitutive models can be much faster. Note that this experiment is performed on a single CPU, but in practice the proposed network can be massively parallel since the computation of each point is independent, which would further increase the computational superiority of the neural-network-based method.

4. Conclusion

Constitutive closure models based on transport partial differential equations (PDEs) are commonly used in computational fluid dynamics. Such closure models often lack robustness and are too rigid to calibrate with diverse training data. Drawing inspirations from a previous neural-network-based molecular potential energy model, in

this work we propose a vector-cloud neural network to represent the nonlocal physics in the transport PDEs as a region-to-point mapping. Such a vector-cloud network is invariant to coordinate translation, rotation, and ordering of the points. It also accounts for the transport physics through corresponding scalar features. The vector-cloud network works on arbitrary grids and is suitable for finite-element and finite-volume solvers that are common in fluid dynamics. The proposed network has been used as surrogate models for scalar transport PDEs on a family of parameterized periodic hill geometries and it has been demonstrated to have desired invariance and satisfactory accuracy. Therefore, it is a promising tool not only as nonlocal constitutive models and but also as general surrogate models for PDEs on irregular domains.

Despite the preliminary successes demonstrated in the present work, there are several directions of future research that should be conducted. First and foremost, the performance of using the learned nonlocal constitutive models as the closure models for solving the primary PDEs (e.g., the RANS equations in turbulence modeling for mean velocities and pressure) should be methodically examined in future work. Moreover, the proposed neural network model is now applied to a passive scalar, still different from the full Reynolds stress tensor in the RANS equations. Learning a constitutive model for a general anisotropic tensor is not straightforward. We need to guarantee the rotational equivariance of the tensor instead of the rotational invariance of the scalar. The work of predicting the polarizability tensor of liquid water [66] based on Deep Potential provides a possible approach to achieve this goal. Nevertheless, we still need to explore more options to accurately model the complicated dynamics of the Reynolds stress in turbulence with neural networks. Last but not least, we have demonstrated the invariance of the proposed vector cloud neural network but not shown the advantages of such invariance, especially the rotational invariance. The superiority should be studied systematically by comparing with the neural network without such invariance in future work.

Acknowledgment

H. Xiao is supported by the U.S. Air Force under agreement number FA865019-2-2204. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Construction of constitutive scalar transport equation

The transport PDE in Eq. (3) is used in this work as target for the vector-cloud neural network to emulate. Its construction is inspired by the transport PDEs encountered in turbulence modeling. Among the most commonly used turbulence models are the eddy viscosity models. The transport of turbulent kinetic energy (TKE) k for incompressible flows is described by the following PDE [59]:

$$\mathbf{u}\cdot\nabla k-\nabla\cdot\big((\nu+\nu_{\mathrm{t}})\nabla k\big)=\mathsf{P}-\mathsf{E}$$

where $k=\frac{1}{2}\,\mathrm{tr}(\boldsymbol{\tau})$ is the trace of the Reynolds stress tensor $\boldsymbol{\tau}$, $\mathbf{u}$ is the mean (Reynolds-averaged) velocity, ν_{t} is the turbulent eddy viscosity, $\mathsf{P}=\boldsymbol{\tau}:\nabla\mathbf{u}$ is the production term, and E is the dissipation term.

In order to provide closure for $\boldsymbol{\tau}$ in the production term, we will utilize the Boussinesq assumption $\boldsymbol{\tau}=2\nu_t\boldsymbol{s}-\frac{2}{3}kI$ along with the following facts:

$$\nabla\mathbf{u}=\boldsymbol{s}+\Omega,\quad \boldsymbol{s}:\Omega=0,\quad \text{and}\quad I:\nabla\mathbf{u}=\delta_{ij}\frac{\partial u_i}{\partial x_j}=\frac{\partial u_i}{\partial x_i}=0,$$

where $\boldsymbol{s}$ and Ω are strain-rate and vorticity tensors, respectively, I (or Kronecker delta δ_{ij}) is second-rank identity tensor, : indicates double dot product (tensor contraction). The production of TKE can thus be written as:

$$\mathsf{P}=\boldsymbol{\tau}:\nabla\mathbf{u}=2\nu_t\boldsymbol{s}:\nabla\mathbf{u}=2\nu_t\boldsymbol{s}:(\boldsymbol{s}+\Omega)=2\nu_t\,\boldsymbol{s}:\boldsymbol{s}\equiv\sqrt{k}\ell_m s^2$$

where in the last step the tensor norm is defined as $s=\|\boldsymbol{s}\|=\sqrt{2\boldsymbol{s}:\boldsymbol{s}}$, and the turbulent eddy viscosity is approximated as $\nu_t=\sqrt{k}\ell_m$ based on Prandtl's mixing length assumption, with mixing length ℓ_m defined in Eq. (4).

Evoking again the analogy between concentration τ and the TKE k, the boundary condition for τ is set to zero at the walls. The dissipation of TKE in Prantdl's one-equation model follows the form $\mathsf{E} = C_\zeta \frac{k^{\frac{3}{2}}}{\ell_m}$. However, for simplicity, we set $\mathsf{E} = C_\zeta k^2$ (where the coefficient C_ζ shall be chosen to make the dimension consistent) by following the other transport equations [59], e.g., the dissipation ε, the dissipation rate ω, and the turbulent viscosity-like variable $\tilde{\nu}$, where the destruction terms are as ε^2, ω^2, and $\tilde{\nu}^2$ in the respective equations. With $\mathsf{P} = \sqrt{k}\ell_m s^2$ and $\mathsf{E} = C_\zeta k^2$, the analogy between the TKE transport equation and Eq. (3) then becomes evident.

Appendix B. Non-dimensionalization of the passive tracer transport equation

In this work, the neural network is used to emulate the transport equation for the hypothetical volumetric concentration τ (dimensionless):

$$\mathbf{u} \cdot \nabla\tau - \nabla \cdot (C_\nu \nabla\tau) = C_g \ell_m \sqrt{\tau} s^2 - C_\zeta \tau^2,$$

where C_g (unit $[\mathrm{m}^{-1}\mathrm{s}]$), C_ν (unit $[\mathrm{m}^2/\mathrm{s}]$), and C_ζ (unit $[\mathrm{s}^{-1}]$) are coefficients associated with production, diffusion, and dissipation, respectively. Although the PDE above is in a dimensional form with physical units, the inputs of the vector cloud neural network are all dimensionless quantities (see Table 1). Therefore, the network can be trained on data from transport equations from vastly different length- and velocity-scales, as long as they are dynamically similar. Furthermore, the similarity argument also allows the network to predict convection–diffusion–reaction flows that have different scales from the training flows if they are dynamically similar. In the following, we show how dynamic similarity can be achieved for the convection–diffusion–reaction PDEs above.

We introduce the characteristic length L_0 and the characteristic flow velocity U_0 to normalize the variables as follows:

$$\mathbf{u}^* = \frac{\mathbf{u}}{U_0}, \quad \ell_m^* = \frac{\ell_m}{L_0}, \quad \nabla^* = L_0 \nabla, \tag{B.1}$$

where the dimensionless variables are indicated in superscript $*$, and accordingly the strain rate magnitude and coefficients are normalized as follows:

$$s^* = \frac{L_0}{U_0} s, \quad C_\nu^* = \frac{C_\nu}{U_0 L_0}, \quad C_g^* = U_0 C_g, \quad C_\zeta^* = \frac{L_0}{U_0} C_\zeta. \tag{B.2}$$

As such, we can rewrite the transport equation in the following dimensionless form:

$$\mathbf{u}^* \cdot \nabla^* \tau - \nabla^* \cdot \left(C_\nu^* \nabla^* \tau\right) = C_g^* \ell_m^* \sqrt{\tau} s^{*2} - C_\zeta^* \tau^2. \tag{B.3}$$

From the non-dimensionlization scheme in Eq. (B.2) and the resulting dimensionless equation (B.3), we can see the solutions of two convection–diffusion–reaction equations at different scales are dynamically similar if they have the same non-dimensional diffusion, production, and dissipation coefficients (C_ν^*, C_g^*, and C_ζ^*) and the same dimensionless strain rate magnitude (i.e., the same time scale $T_0 \equiv U_0/L_0$). For example, the system studied in this work has length and velocity scales $L_0 = 1$ m and $U_0 = 1$ m/s. Consider system with a characteristic length λL_0, it must have a characteristic velocity of λU_0 and would further need to have the diffusivity C_ν scaled by a factor of λ^2, the production rate C_g by $1/\lambda$, and the dissipation C_ζ by 1 in order to preserve dynamic similarity. Note that the characteristic length L_0 and the characteristic velocity U_0 must scale at the same ratio to achieve the dynamic similarity, because the production model P depends on the strain rate $\mathbf{s}$.

Appendix C. Invariance of the proposed constitutive neural network

In this work, we are interested in predicting the scalar quantity $\tau(\boldsymbol{x}_0)$ at the location $\boldsymbol{x}_0$ from data collected at n neighboring locations $\boldsymbol{x}_i$, $i = 1, \ldots, n$. We introduce our methodology in two dimensional space in consistency with our computation examples. Extending it to three dimensional case is straightforward following the same procedure. At each point $\boldsymbol{x}_i$ in the collection we have velocity $\mathbf{u}_i = \mathbf{u}(\boldsymbol{x}_i)$ and a number of other scalar features, denoted together by $\boldsymbol{c}_i = \boldsymbol{c}(\boldsymbol{x}_i) \in \mathbb{R}^{l'}$. We denote the region-to-point mapping by g, i.e.,

$$\tau(\boldsymbol{x}_0) = g\Big(\left\{\boldsymbol{x}_i, \mathbf{u}(\boldsymbol{x}_i), \boldsymbol{c}(\boldsymbol{x}_i)\right\}_{i=1}^n\Big). \tag{C.1}$$

Given the underlying physical knowledge, we want this mapping to possess three types of invariance: translational, rotational, and permutational invariances. We first give a formal definition of these properties.

- **Translational invariance**. If we shift the coordinate system with a constant vector $\Delta \boldsymbol{x}$, the mapping in the new system should be the same

$$\tau(\boldsymbol{x}_0 + \Delta \boldsymbol{x}) = g\Big(\big\{\boldsymbol{x}_i + \Delta \boldsymbol{x}, \mathbf{u}(\boldsymbol{x}_i + \Delta \boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x}_i + \Delta \boldsymbol{x})\big\}_{i=1}^{n}\Big). \tag{C.2}$$

- **Rotational invariance**. If we rotate the coordinate system through an orthogonal matrix $\mathcal{R} \in \mathbb{R}^{2\times 2}$, the mapping in the new system should be the same

$$\tau(\mathcal{R}\boldsymbol{x}_0) = g\Big(\big\{\mathcal{R}\boldsymbol{x}_i, \mathcal{R}\mathbf{u}(\mathcal{R}\boldsymbol{x}_i), \boldsymbol{c}(\mathcal{R}\boldsymbol{x}_i)\big\}_{i=1}^{n}\Big). \tag{C.3}$$

- **Permutational invariance**. The mapping should be independent of the indexing of collection points, i.e., if σ denotes an arbitrary permutation of the set $\{1, 2, \ldots, n\}$, we have

$$\tau(\boldsymbol{x}_0) = g\Big(\big\{\boldsymbol{x}_{\sigma(i)}, \mathbf{u}(\boldsymbol{x}_{\sigma(i)}), \boldsymbol{c}(\boldsymbol{x}_{\sigma(i)})\big\}_{i=1}^{n}\Big). \tag{C.4}$$

In addition, the data we collected at the neighboring points can be interpreted as a discretized sampling of the continuous fields $\mathbf{u}$ and $\boldsymbol{c}$, and we are interested in the general case where n varies in a certain range among different regions. So we also want the mapping g to be applicable to different sampling numbers n.

Granted the fitting ability of neural networks, the key to a achieve a general representation satisfying the above properties is an *embedding* procedure that maps the original input to invariance preserving components. We draw inspiration from the following two observations.

- **Translation and rotation**. If we define a relative coordinate matrix

$$\mathcal{X} = \begin{bmatrix} (\boldsymbol{x}_1 - \boldsymbol{x}_0)^\top \\ (\boldsymbol{x}_2 - \boldsymbol{x}_0)^\top \\ \vdots \\ (\boldsymbol{x}_n - \boldsymbol{x}_0)^\top \end{bmatrix} \in \mathbb{R}^{n\times 2} \tag{C.5}$$

  then the symmetric matrix $\mathcal{X}\mathcal{X}^\top \in \mathbb{R}^{n\times n}$ is an over-complete array of invariants with respect to translation and rotation, i.e., it contains the complete information of the pattern of neighboring points' locations [43,44]. However, this symmetric matrix depends on the indexing of points. It switches rows and columns under a permutational operation of the neighboring points.
- **Permutation and flexibility with sampling number**. It is proven [67,68] that if f is a function taking the set $\{\boldsymbol{z}_i\}_{i=1}^{n}$ as input, i.e., f is permutational invariant with respect to the set's elements, then it can be represented as

$$\Phi\Big(\frac{1}{n}\sum_{i=1}^{n}\phi(\boldsymbol{z}_i)\Big) \tag{C.6}$$

  where $\phi(\boldsymbol{z}_i)$ is a multidimensional function, and $\Phi(\cdot)$ is another general function. On the other hand, we can see such a representation intrinsically preserves the permutational invariance and flexibility with the number of point n.

  This result is similar in flavor to the Kolmogorov–Arnold representation theorem [69] but specific to permutational invariant functions. Here we give two examples to provide readers some intuition. ${z_1}^2+{z_2}^2+z_1z_2$ is permutational invariant with two scalar input variables z_1, z_2. This function can be written in the form of Eq. (C.6) when we define $\phi(z) = [z, z^2]^\top$ and $\Phi([a, b]^\top) = 2a^2 + b$:

$$\Phi\Big(\frac{1}{2}\sum_{i=1}^{2}\phi(\boldsymbol{z}_i)\Big) = \Phi\left(\frac{1}{2}\begin{bmatrix} z_1+z_2 \\ {z_1}^2+{z_2}^2 \end{bmatrix}\right) = \frac{1}{2}(z_1+z_2)^2 + \frac{1}{2}({z_1}^2+{z_2}^2) = {z_1}^2 + {z_2}^2 + z_1z_2. \tag{C.7}$$

  Similarly, $z_1+z_2+z_3+2z_1z_2z_3$ is permutational invariant with three scalar input variables z_1, z_2, z_3. Assuming $\phi(z) = [z, z^2, z^3]^\top$ and $\Phi([a, b, c]^\top) = 9a^3 - 9ab + 3a + 2c$, we can verify that the composition in the form of Eq. (C.6) gives us the desired function.

Inspired by the above observations, we construct our nonlocal mapping through neural networks in the following four steps. As explained below, the final output inherits all the invariant properties intrinsically.

(1) Given the input data $\left(\boldsymbol{x}_0, \left\{\boldsymbol{x}_i, \mathbf{u}(\boldsymbol{x}_i), \boldsymbol{c}(\boldsymbol{x}_i)\right\}_{i=1}^n\right)$, we define the input data matrix:

$$\mathcal{Q} = [\mathcal{X}, \mathcal{U}, \mathcal{C}] \in \mathbb{R}^{n\times(4+l')}, \tag{C.8}$$

where

$$\mathcal{U} = \begin{bmatrix} \mathbf{u}(\boldsymbol{x}_1)^\top \\ \mathbf{u}(\boldsymbol{x}_2)^\top \\ \vdots \\ \mathbf{u}(\boldsymbol{x}_n)^\top \end{bmatrix} \in \mathbb{R}^{n\times 2}, \quad \mathcal{C} = \begin{bmatrix} \boldsymbol{c}(\boldsymbol{x}_1)^\top \\ \boldsymbol{c}(\boldsymbol{x}_2)^\top \\ \vdots \\ \boldsymbol{c}(\boldsymbol{x}_n)^\top \end{bmatrix} \in \mathbb{R}^{n\times l'}, \tag{C.9}$$

and $\mathcal{X} \in \mathbb{R}^{n\times 2}$ is defined in Eq. (C.5). Note that the elements in $\mathcal{C}$ are all translational and rotational invariant.

(2) We define m embedding functions $\{\phi_k(\boldsymbol{c}_i)\}_{k=1}^m$ for the scalar features $\boldsymbol{c}_i$ of each point i and compute the embedding matrix as

$$\mathcal{G} = \begin{bmatrix} \phi_1(\boldsymbol{c}_1) & \phi_2(\boldsymbol{c}_1) & \dots & \phi_m(\boldsymbol{c}_1) \\ \phi_1(\boldsymbol{c}_2) & \phi_2(\boldsymbol{c}_2) & \dots & \phi_m(\boldsymbol{c}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{c}_n) & \phi_2(\boldsymbol{c}_n) & \dots & \phi_m(\boldsymbol{c}_n) \end{bmatrix} \in \mathbb{R}^{n\times m}. \tag{C.10}$$

We can interpret matrix $\mathcal{G}$ as a collection of basis, as seen in the last step below. We also select the first m' ($\leq m$) columns of $\mathcal{G}$ as another embedding matrix $\mathcal{G}^\star \in \mathbb{R}^{n\times m'}$. Since all the features in $\boldsymbol{c}$ are translational and rotational invariant, so are the elements in $\mathcal{G}, \mathcal{G}^\star$. In implementation we use an embedding network with m-dimensional output to instantiate the m functions $\{\phi_k(\boldsymbol{c}_i)\}_{k=1}^m$.

(3) Next we consider

$$\mathcal{Q}\mathcal{Q}^\top = \mathcal{X}\mathcal{X}^\top + \mathcal{U}\mathcal{U}^\top + \mathcal{C}\mathcal{C}^\top \in \mathbb{R}^{n\times n}, \tag{C.11}$$

which is also translational and rotational invariant. In particular, to see its rotational invariance, we check that the new matrix under a rotation $\mathcal{R}$ becomes

$$\mathcal{X}\mathcal{R}^\top\mathcal{R}\mathcal{X}^\top + \mathcal{U}\mathcal{R}^\top\mathcal{R}\mathcal{U}^\top + \mathcal{C}\mathcal{C}^\top, \tag{C.12}$$

which remains the same.

(4) Finally, we consider the encoded feature matrix

$$\mathcal{D} = \frac{1}{n^2}\mathcal{G}^\top\mathcal{Q}\mathcal{Q}^\top\mathcal{G}^\star \in \mathbb{R}^{m\times m'}.$$

Given that all the elements in $\mathcal{G}, \mathcal{G}^\star, \mathcal{Q}\mathcal{Q}^\top$ are translational and rotational invariant, so are the elements of $\mathcal{D}$. By the second observation above, we know each element in $\mathcal{G}^\top\mathcal{Q}$ is permutational invariant, and so is $\mathcal{G}^{\star\top}\mathcal{Q}$. So all the elements in $\mathcal{D}$ possess all the desired properties. Now we reshape $\mathcal{D}$ into a vector to form the input of the *fitting* network f_{fit}, which outputs the predicted concentration $\hat{\tau}(\boldsymbol{x}_0)$.

In the above procedures, essentially we need to train the parameters associated with two networks, the embedding network f_{embed} and fitting network f_{fit}. We also remark that by definition the evaluation of the predicted τ is flexible with the sampling number n and has linear computational complexity with respect to n.

Appendix D. Minimal example of invariance of the proposed constitutive neural network

In this example, we consider a minimal example where the scalar quantity τ is predicted based on the data collected at three neighboring locations and the input matrix $\mathcal{Q}$ has only three input features:

$$\mathcal{Q} = \begin{bmatrix} x_1 & y_1 & c_1 \\ x_2 & y_2 & c_2 \\ x_3 & y_3 & c_3 \end{bmatrix} \in \mathbb{R}^{3\times 3}, \tag{D.1}$$

where x, y are the relative coordinates and c is the only scalar feature. The scalar feature c attached to each point is fed into the embedding network to obtain the embedding matrix $\mathcal{G}$, which is processed individually and identically.

Table E.2
Detailed architectures of the embedding neural network and the fitting network. The architecture denotes the numbers of neurons in each layer in sequence, from the input layer to the hidden layers (if any) and the output layer. The numbers of neurons in the input and output layers are highlighted in bold. Note that the embedding network operate identically on the scalar features $\boldsymbol{c} \in \mathbb{R}^{l'}$ associated with each of the point in the cloud and output a row of m elements in matrix $\mathcal{G}$.

	Embedding network	Fitting network ($\mathcal{D} \mapsto \tau$)
No. of input neurons	$l' = 7$	$m \times m' = 256\,(\mathcal{D} \in \mathbb{R}^{64\times 4})$
No. of hidden layers	2	1
Architecture	(**7**, 32, 64, **64**)	(**256**, 128, **1**)
No. of output neurons	$m = 64$	1 ($\tau \in \mathbb{R}$)
Activation functions	ReLU	ReLU, Linear (last layer)
No. of trainable parameters	6528	33 025

For convenience, we set $m = 3$ and $m' = 1$. The embedding matrix and its submatrix are then computed as:

$$\mathcal{G} = \begin{bmatrix} \phi_1(c_1) & \phi_2(c_1) & \phi_3(c_1) \\ \phi_1(c_2) & \phi_2(c_2) & \phi_3(c_2) \\ \phi_1(c_3) & \phi_2(c_3) & \phi_3(c_3) \end{bmatrix} \in \mathbb{R}^{3\times 3}, \quad \mathcal{G}^{\star} = \begin{bmatrix} \phi_1(c_1) \\ \phi_1(c_2) \\ \phi_1(c_3) \end{bmatrix} \in \mathbb{R}^{3\times 1}, \tag{D.2}$$

where $\{\phi_k(c)\}_{k=1}^{3}$ are embedding functions. Finally, the invariant feature matrix is computed as:

$$\mathcal{D} = \frac{1}{3^2}\mathcal{G}^{\top}\mathcal{Q}\mathcal{Q}^{\top}\mathcal{G}^{\star} \in \mathbb{R}^{3\times 1}, \tag{D.3}$$

where the feature matrix $\mathcal{D}$ has translational, rotational and permutational invariances.

- **Translational invariance**. Given that the input x, y are the relative coordinates and c is the passive scalar, the input matrix $\mathcal{Q}$ will not change if the coordinate system is translated.
- **Rotational invariance**. Given a rotation matrix $\mathcal{R}$ defined as:

  $$\mathcal{R} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \in \mathbb{R}^{2\times 2}, \tag{D.4}$$

  where θ is the rotation angle of the coordinate system. In the new rotated coordinate system, $\mathcal{G}$ and $\mathcal{G}^{\star}$ remain constant but the input matrix becomes

  $$\mathcal{Q}_r = \begin{bmatrix} x_1\cos\theta + y_1\sin\theta & -x_1\sin\theta + y_1\cos\theta & c_1 \\ x_2\cos\theta + y_2\sin\theta & -x_2\sin\theta + y_2\cos\theta & c_2 \\ x_3\cos\theta + y_3\sin\theta & -x_3\sin\theta + y_3\cos\theta & c_3 \end{bmatrix} \in \mathbb{R}^{3\times 3}. \tag{D.5}$$

  In this way, the pairwise projection is computed as:

  $$\begin{aligned} \mathcal{Q}_r\mathcal{Q}_r^{\top} &= \begin{bmatrix} c_1^2 + x_1^2 + y_1^2 & c_1c_2 + x_1x_2 + y_1y_2 & c_1c_3 + x_1x_3 + y_1y_3 \\ c_1c_2 + x_1x_2 + y_1y_2 & c_2^2 + x_2^2 + y_2^2 & c_2c_3 + x_2x_3 + y_2y_3 \\ c_1c_3 + x_1x_3 + y_1y_3 & c_2c_3 + x_2x_3 + y_2y_3 & c_3^2 + x_3^2 + y_3^2 \end{bmatrix} \\ &= \mathcal{Q}\mathcal{Q}^{\top}, \end{aligned} \tag{D.6}$$

  which means that the pairwise projection stays the same after rotating the coordinate system. Therefore the feature matrix $\mathcal{D}$ has rotational invariance.
- **Permutational invariance**. The obtained feature matrix $\mathcal{D}$ should be independent of the indexing of three collected points. As defined above, $\mathcal{L} = \frac{1}{3}\mathcal{G}^{\top}\mathcal{Q}$, which is computed as:

  $$\mathcal{L} = \frac{1}{3}\begin{bmatrix} \phi_1(c_1)x_1 + \phi_1(c_2)x_2 + \phi_1(c_3)x_3 & \dots & \phi_1(c_1)c_1 + \phi_1(c_2)c_2 + \phi_1(c_3)c_3 \\ \phi_2(c_1)x_1 + \phi_2(c_2)x_2 + \phi_2(c_3)x_3 & \dots & \phi_2(c_1)c_1 + \phi_2(c_2)c_2 + \phi_2(c_3)c_3 \\ \phi_3(c_1)x_1 + \phi_3(c_2)x_2 + \phi_3(c_3)x_3 & \dots & \phi_3(c_1)c_1 + \phi_3(c_2)c_2 + \phi_3(c_3)c_3 \end{bmatrix} \in \mathbb{R}^{3\times 3}. \tag{D.7}$$

  With regard to the first element $\mathcal{L}_{11} = \frac{1}{3}\left[\phi_1(c_1)x_1 + \phi_1(c_2)x_2 + \phi_1(c_3)x_3\right] = \frac{1}{3}\sum_{i=1}^{3}\phi_1(c_i)x_i$, it is obviously independent of the indexing of collected points and so are other elements, which means the matrix $\mathcal{L}$ is permutational invariant and so is $\mathcal{L}^{\star} = \frac{1}{3}\mathcal{G}^{\star\top}\mathcal{Q}$. Therefore the feature matrix $\mathcal{D}$ has permutational invariance.

Appendix E. Network architectures and training parameters

Detailed architectures of the embedding neural network and the fitting network are presented in Table E.2. The Adam optimizer [70] is adopted to train the neural networks. The training process takes 2000 epochs with a batch size of 1024. The training is scheduled such that the learning rate is initialized with 0.001 and is reduced by multiplying a factor of 0.7 every 600 epochs.

References

[1] A.P. Singh, S. Medida, K. Duraisamy, Machine learning-augmented predictive modeling of turbulent separated flows over airfoils, AIAA J. 55 (7) (2017) 2215–2227.
[2] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. 807 (2016) 155–166.
[3] J.-X. Wang, J.-L. Wu, H. Xiao, Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data, Phys. Rev. Fluids 2 (3) (2017) 034603.
[4] J.-L. Wu, H. Xiao, E.G. Paterson, Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework, Phys. Rev. Fluids 3 (2018) 074602.
[5] X.I.A. Yang, S. Zafar, J.-X. Wang, H. Xiao, Predictive large-eddy-simulation wall modeling via physics-informed neural networks, Phys. Rev. Fluids 4 (2019) 034602.
[6] M. Schmelzer, R.P. Dwight, P. Cinnella, Discovery of algebraic Reynolds-stress models using sparse symbolic regression, Flow Turbul. Combust. 104 (2) (2020) 579–603.
[7] D. Stefanos, P. Gyan, On neural network constitutive models for geomaterials, J. Civ. Eng. Res. 5 (5) (2015) 106–113.
[8] T. Kirchdoerfer, M. Ortiz, Data-driven computational mechanics, Comput. Methods Appl. Mech. Engrg. 304 (2016) 81–101.
[9] F.E. Bock, R.C. Aydin, C.J. Cyron, N. Huber, S.R. Kalidindi, B. Klusemann, A review of the application of machine learning and data mining approaches in continuum materials mechanics, Front. Mater. 6 (2019) 110.
[10] J. Han, C. Ma, Z. Ma, W. E, Uniformly accurate machine learning-based hydrodynamic models for kinetic equations, Proc. Natl. Acad. Sci. 116 (44) (2019) 21983–21991.
[11] D.Z. Huang, K. Xu, C. Farhat, E. Darve, Learning constitutive relations from indirect observations using deep neural networks, J. Comput. Phys. 416 (2020) 109491.
[12] K. Xu, D.Z. Huang, E. Darve, Learning constitutive relations using symmetric positive definite neural networks, J. Comput. Phys. 428 (2021) 110072.
[13] F. Masi, I. Stefanou, P. Vannucci, V. Maffi-Berthier, Thermodynamics-based artificial neural networks for constitutive modeling, 2020, arXiv preprint arXiv:2005.12183.
[14] C.A.M. Ströfer, H. Xiao, End-to-end differentiable learning of turbulence models from indirect observations, Theor. Appl. Mech. Lett. 11 (4) (2021) 100280.
[15] T.B. Gatski, M.Y. Hussaini, J.L. Lumley, Simulation and Modeling of Turbulent Flows, Oxford University Press, 1996.
[16] B. Basara, S. Jakirlic, A new hybrid turbulence modelling strategy for industrial CFD, Internat. J. Numer. Methods Fluids 42 (1) (2003) 89–116.
[17] F.R. Menter, P.E. Smirnov, T. Liu, R. Avancha, A one-equation local correlation-based transition model, Flow Turbul. Combust. 95 (4) (2015) 583–619.
[18] J.G. Coder, M.D. Maughmer, Computational fluid dynamics compatible transition modeling using an amplification factor transport equation, AIAA J. 52 (11) (2014) 2506–2512.
[19] P.R. Spalart, S.R. Allmaras, A one equation turbulence model for aerodynamic flows., AIAA J. 94 (1992).
[20] B. Launder, B. Sharma, Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc, Lett. Heat Mass Transfer 1 (2) (1974) 131–137.
[21] D.C. Wilcox, Reassessment of the scale-determining equation for advanced turbulence models, AIAA J. 26 (11) (1988) 1299–1310.
[22] X.-H. Zhou, J. Han, H. Xiao, Learning nonlocal constitutive models with neural networks, Comput. Methods Appl. Mech. Engrg. 384 (2021) 113927.
[23] C.R. Gin, D.E. Shea, S.L. Brunton, J.N. Kutz, DeepGreen: Deep learning of Green's functions for Nonlinear Boundary Value Problems, 2020, arXiv preprint arXiv:2101.07206.
[24] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, 2020, arXiv preprint arXiv:2003.03485.
[25] C.G. Speziale, A review of material frame-indifference in mechanics, Appl. Mech. Rev. 51 (8) (1998) 489–504.
[26] P.R. Spalart, Philosophies and fallacies in turbulence modeling, Prog. Aerosp. Sci. 74 (2015) 1–15.
[27] S.B. Pope, A more general effective-viscosity hypothesis, J. Fluid Mech. 72 (2) (1975) 331–340.
[28] T. Gatski, C. Speziale, On explicit algebraic stress models for complex turbulent flows, J. Fluid Mech. 254 (1993) 59–79.
[29] C.G. Speziale, S. Sarkar, T.B. Gatski, Modelling the pressure–strain correlation of turbulence: an invariant dynamical systems approach, J. Fluid Mech. 227 (1991) 245–272.
[30] Z. Long, Y. Lu, X. Ma, B. Dong, PDE-Net: Learning PDEs from data, in: International Conference on Machine Learning, PMLR, 2018, pp. 3208–3216.
[31] Z. Long, Y. Lu, B. Dong, PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network, J. Comput. Phys. 399 (2019) 108925.

[32] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Engrg. 361 (2020) 112732.
[33] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, B. Solenthaler, DeepFluids: A generative network for parameterized fluid simulations, Comput. Graph. Forum (Proc. Eurograph.) 38 (2) (2019).
[34] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 481–490.
[35] L. Lu, P. Jin, G.E. Karniadakis, DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators, 2019, arXiv preprint arXiv:1910.03193.
[36] C. Ma, B. Zhu, X.-Q. Xu, W. Wang, Machine learning surrogate models for Landau fluid closure, Phys. Plasmas 27 (4) (2020) 042502.
[37] M.D. Ribeiro, A. Rehman, S. Ahmed, A. Dengel, DeepCFD: Efficient steady-state laminar flow approximation with deep convolutional neural networks, 2020, arXiv preprint arXiv:2004.08826.
[38] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Multipole graph neural operator for parametric partial differential equations, in: Advances in Neural Information Processing Systems, Vol. 33, 2020, pp. 6755–6766.
[39] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, in: International Conference on Learning Representations, 2021.
[40] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 25 (2012) 1097–1105.
[41] A. Sharif Razavian, H. Azizpour, J. Sullivan, S. Carlsson, CNN features off-the-shelf: an astounding baseline for recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014, pp. 806–813.
[42] J. Ling, R. Jones, J. Templeton, Machine learning strategies for systems with invariance properties, J. Comput. Phys. 318 (2016) 22–35.
[43] A.P. Bartók, R. Kondor, G. Csányi, On representing chemical environments, Phys. Rev. B 87 (18) (2013) 184115.
[44] H. Weyl, The Classical Groups: Their Invariants and Representations, Vol. 45, Princeton University Press, 1946.
[45] W. E, J. Han, L. Zhang, Machine-learning-assisted modeling, Phys. Today 74 (7) (2021) 36–41.
[46] J.-L. Wu, R. Sun, S. Laizet, H. Xiao, Representation of stress tensor perturbations with application in machine-learning-assisted turbulence modeling, Comput. Methods Appl. Mech. Engrg. 346 (2019) 707–726.
[47] M.I. Zafar, H. Xiao, M.M. Choudhari, F. Li, C.-L. Chang, P. Paredes, B. Venkatachari, Convolutional neural network for transition modeling based on linear stability theory, Phys. Rev. Fluids 5 (2020) 113903.
[48] S. Taghizadeh, F.D. Witherden, S.S. Girimaji, Turbulence closure modeling with data-driven techniques: physical compatibility and consistency considerations, New J. Phys. 22 (9) (2020) 093023.
[49] N.A.K. Doan, W. Polifke, L. Magri, Auto-encoded reservoir computing for turbulence learning, 2020, arXiv preprint arXiv:2012.10968.
[50] J.-L. Wu, K. Kashinath, A. Albert, D. Chirila, M. Prabhat, H. Xiao, Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems, J. Comput. Phys. 406 (2020) 109209.
[51] N. Doan, W. Polifke, L. Magri, Physics-informed echo state networks, J. Comput. Sci. 47 (2020) 101237.
[52] N.A.K. Doan, W. Polifke, L. Magri, Short- and long-term prediction of a chaotic flow: A physics-constrained reservoir computing approach, 2021, arXiv preprint arXiv:2102.07514.
[53] H. Yu, M.P. Juniper, L. Magri, A data-driven kinematic model of a ducted premixed flame, Proc. Combust. Inst. (2020).
[54] J. Han, L. Zhang, R. Car, W. E, Deep potential: a general representation of a many-body potential energy surface, Commun. Comput. Phys. 23 (3) (2018) 629–639.
[55] L. Zhang, J. Han, H. Wang, W. Saidi, R. Car, W. E, End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems, in: Advances in Neural Information Processing Systems, 2018, pp. 4436–4446.
[56] L. Zhang, J. Han, H. Wang, R. Car, W. E, Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics, Phys. Rev. Lett. 120 (14) (2018) 143001.
[57] L. Zhang, J. Han, H. Wang, R. Car, W. E, DeePCG: Constructing coarse-grained models via deep neural networks, J. Chem. Phys. 149 (3) (2018) 034101.
[58] W. Jia, H. Wang, M. Chen, D. Lu, L. Lin, R. Car, W. E, L. Zhang, Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1–14.
[59] S.B. Pope, Turbulent Flows, Cambridge University Press, Cambridge, 2000.
[60] M. Breuer, N. Peller, C. Rapp, M. Manhart, Flow over periodic hills–numerical and experimental study in a wide range of Reynolds numbers, Comput. & Fluids 38 (2) (2009) 433–457.
[61] H. Xiao, J.-L. Wu, S. Laizet, L. Duan, Flows over periodic hills of parameterized geometries: A dataset for data-driven turbulence modeling from direct simulations, Comput. & Fluids 200 (2020) 104431.
[62] The OpenFOAM Foundation, OpenFOAM user guide, 2020, URL https://cfd.direct/openfoam/user-guide.
[63] R.I. Issa, Solution of the implicitly discretised fluid flow equations by operator-splitting, J. Comput. Phys. 62 (1986) 40–65.
[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019, pp. 8026–8037.
[65] X.-H. Zhou, J. Han, H. Xiao, Learning nonlocal constitutive models with vector cloud neural networks, https://www.github.com/xuhuizhou-vt/VCNN-nonlocal-constitutive-model.
[66] G.M. Sommers, M.F.C. Andrade, L. Zhang, H. Wang, R. Car, Raman spectrum and polarizability of liquid water from deep neural networks, Phys. Chem. Chem. Phys. 22 (19) (2020) 10592–10602.
[67] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R.R. Salakhutdinov, A.J. Smola, Deep sets, in: Advances in Neural Information Processing Systems, 2017, pp. 3394–3404.

[68] J. Han, Y. Li, L. Lin, J. Lu, J. Zhang, L. Zhang, Universal approximation of symmetric and anti-symmetric functions, 2019, arXiv preprint arXiv:1912.01765.

[69] A.N. Kolmogorov, On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition, in: Doklady Akademii Nauk, Vol. 114, Russian Academy of Sciences, 1957, pp. 953–956.

[70] D. Kingma, J. Ba, Adam: a method for stochastic optimization, in: Proceedings of the International Conference on Learning Representations, 2015.

Chapter 3

An equivariant neural operator for developing nonlocal tensorial constitutive models

Journal of Computational Physics 488 (2023) 112243

Contents lists available at ScienceDirect

Journal of Computational Physics

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

An equivariant neural operator for developing nonlocal tensorial constitutive models

Jiequn Han [a,1], Xu-Hui Zhou [b,1], Heng Xiao [c,*]

[a] Center for Computational Mathematics, Flatiron Institute, New York, NY, USA
[b] Kevin T. Crofton Department of Aerospace and Ocean Engineering, Virginia Tech, Blacksburg, VA, USA
[c] Stuttgart Center for Simulation Science, University of Stuttgart, Stuttgart, BW, Germany

ARTICLE INFO

Article history:
Received 5 December 2022
Received in revised form 28 April 2023
Accepted 18 May 2023
Available online 24 May 2023

Keywords:
Neural operator
Nonlocal closure model
Constitutive modeling
Invariance and equivariance
Deep learning

ABSTRACT

Developing robust constitutive models is a fundamental and longstanding problem for accelerating the simulation of multiscale physics. Machine learning provides promising tools to construct constitutive models based on various calibration data. However, with either traditional or machine learning techniques, the constitutive model for tensorial quantities has been much less studied than scalar quantities due to more complicated physics, even though the former plays an important role in many scientific and engineering applications. In this work, we propose a neural operator to develop nonlocal constitutive models for tensorial quantities through a vector-cloud neural network with equivariance (VCNN-e). The VCNN-e respects all the invariance properties desired by constitutive models, faithfully reflects the region of influence in physics, and is applicable to different spatial resolutions. By design, the model guarantees that the predicted tensor is invariant to the frame translation and ordering (permutation) of the neighboring points. Furthermore, it is equivariant to the frame rotation, i.e., the output tensor co-rotates with the coordinate frame. We evaluate the VCNN-e by using it to emulate the Reynolds stress transport model for turbulent flows, which directly computes the Reynolds stress tensor to close the Reynolds-averaged Navier–Stokes (RANS) equations. The evaluation is performed in two situations: (1) emulating the Reynolds stress model through synthetic data generated from the Reynolds stress transport equations with closure models, and (2) predicting the Reynolds stress by learning from data generated from direct numerical simulations. Such a priori evaluations of the proposed network on realistic and challenging datasets pave the way for developing and calibrating robust and nonlocal, non-equilibrium constitutive models for the RANS equations and other mechanical problems.

1. Introduction

The first principle-based simulations of many practical problems arising from scientific and engineering applications are prohibitively expensive due to their multiscale nature. The constitutive relationship builds an effective bridge between different scales to mitigate this difficulty by simplifying and approximating the unresolved process at the microscopic scale

* Corresponding author.
E-mail address: heng.xiao@simtech.uni-stuttgart.de (H. Xiao).
[1] Contributed equally.

https://doi.org/10.1016/j.jcp.2023.112243
0021-9991/

with the resolved process in order to accelerate the simulations at the macroscopic scale. Taking industrial computational fluid dynamics (CFD) for instance, simulating the Reynolds averaged Navier–Stokes (RANS) equations for the mean flows fields are of ultimate interest in many engineering design tasks. To this end, constitutive models such as eddy viscosity models [1,2] and Reynolds stress models [3] are often introduced to describe unresolved turbulence and close the RANS equations. As such, they are also referred to as "closure models", a term that is used interchangeably with "constitutive models" in this paper. Other examples include developing nonlinear stress-strain constitutive models to describe nonlinear elasticity [4] and developing hydrodynamic models to approximate the motion of particles described by the kinetic equations or non-Newtonian fluids [5,6]. Despite the enormous growth of available computational resources in the past decades, even to this day, such closure models are still the backbone of practical engineering computations.

1.1. Invariance properties of constitutive models

In order to capture the essence of the physical phenomenon, the closure model should ideally express fundamental principles obeying physical constraints and meanwhile be as universally accurate as possible in the considered conditions. Traditional constitutive modeling usually relies on simple parametric models in combination with physical insight. However, as the number of involved state variables increases, it becomes more and more difficult to specify the form of the closure model and calibrate the parameters systematically, an essential obstacle known as the curse of dimensionality. The development of machine learning techniques in recent years, especially deep learning, has provided new tools to extract complicated relationships from massive data in an efficient and flexible way, thus a new possibility to data-driven constitutive modeling [7]. In light of such strengths and promises, many machine learning-based constitutive models have been developed in the past few years in a wide range of application areas such as turbulence modeling [8–11] and computational mechanics [12–16]. In particular, data-driven turbulence models based on invariant tensorial formulations have attracted significant attention [8,17].

In some cases of physical modeling, it has been observed that machine learning-based models tend to achieve the best performance when it respects exactly the physical constraints and symmetries [7,18–21]. However, unlike the traditional constitutive modeling in which conventional wisdom can help restrict the set of admissible functions to obey physical constraints, such as frame independence [22–24], it is not straightforward to restrict flexible machine learning models like neural networks that take the raw state variables as input to be always genuinely physical. The challenge becomes more demanding when the constitutive relationship is nonlocal, i.e., the closure variable at location $\boldsymbol{x}_i$ depends on the values of the resolved variables in a neighborhood region of $\boldsymbol{x}_i$ rather than the value at $\boldsymbol{x}_i$ solely. This situation is most evidently seen in turbulence models. The unresolved turbulent velocity fluctuation (which can be partly characterized by its second-order statistics, i.e., the Reynolds stress) depends on an upstream neighborhood due to the turbulent transport [25]. In such circumstances, an ideal closure model should be indifferent to the choice of material frame and the representation of resolved variables in the nonlocal region. In other words, the constitutive relationship should remain invariant when the coordinate system is translated or the index order of the discretized variable in the neighborhood is permuted. Depending on the type of the closure variable, the constitutive relationship should also remain invariant or equivariant if the coordinate system is rotated. A recent work of Gao et al. [26] considers specifically the rotational invariance/equivariance of a local relationship between a pair of tensors in fluid dynamics. However, the technique developed therein is insufficient to handle the nonlocal relationship. In a different vein, many recent works [see, e.g. 27–34] used machine learning models to approximate global operators or surrogate models defined by the PDEs, which also hold the promise to nonlocal constitutive modeling. However, the objectivity of these modeling approaches, such as frame independence and permutational invariance mentioned above, has rarely been discussed. How to ensure that the machine learning models obey various physical constraints in the context of nonlocal constitutive modeling remains largely an open question.

1.2. Vector-cloud neural network with equivariance (VCNN-e)

One line of efforts to impose all the aforementioned symmetries in data-driven modeling is Deep Potential [35,36] for interatomic potential and the vector-cloud neural network for nonlocal constitutive modeling [37]. At a high level, the original model consists of two modules, an embedding network and a fitting network. The former extracts features that are translational, rotational, and permutational invariant from the raw input, and the latter predicts the output with the desired symmetry. This work focuses on generalizing the vector-cloud neural network (VCNN) [37] to develop a nonlocal closure model for *tensorial* output quantities. Compared to VCNN, in which the closure variable is a rotational invariant scalar [37], the main innovation of the present work is to tackle the closure problem in which the closure variable is a rotational equivariant tensor. To this end, we adopt the idea of Sommers et al. [38] to equip the VCNN with equivariance for tensorial outputs while preserving its general approximation capability. Furthermore, we extend the VCNN equipped with equivariance to correctly represent the function space that is uniquely required for turbulence modeling, as illustrated in Fig. 2(c). In so doing, we obtain a vector-cloud neural network with equivariance (VCNN-e) that satisfies all the symmetries and function space requirements for modeling Reynolds stresses. To demonstrate the merits of the proposed VCNN-e framework, we test it for (1) emulating a Reynolds stress transport equation via synthetic data generated by solving the equation with closure models, and (2) predicting the Reynolds stress through data from direct numerical simulations. The latter test differs from the former in that the Reynolds stress from DNS data is not described by an explicit tensor transport equation and is a more

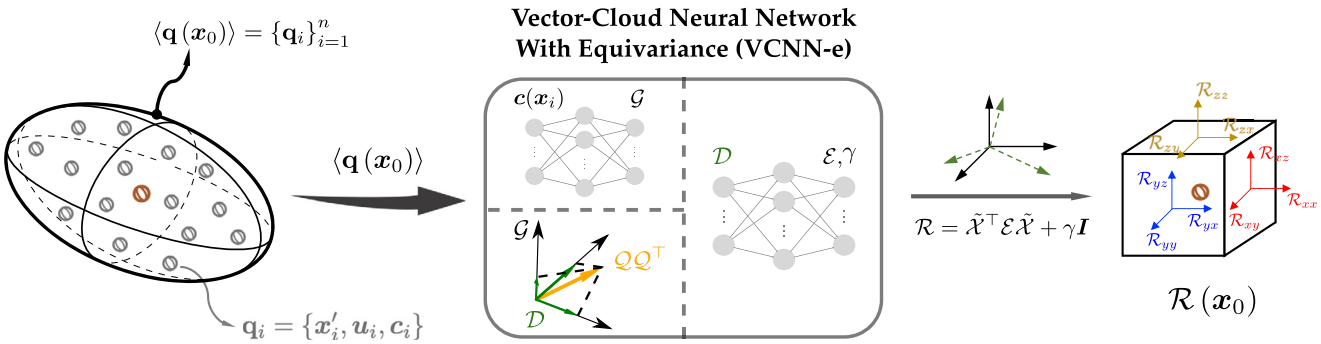


Fig. 1. Schematic of the vector-cloud neural network with equivariance (VCNN-e) for nonlocal tensorial constitutive modeling, showing a region-to-point mapping from a cloud (ellipsoid) of feature vectors $\langle \mathbf{q}(\boldsymbol{x}_0)\rangle$ to the tensorial closure quantity $\mathcal{R}(\boldsymbol{x}_0)$ at center of the ellipsoid. Details on the architecture of VCNN-e are shown subsequently in Fig. 2.

realistic and challenging test. Through the validation on a family of parameterized periodic hill geometries, we showcase that the proposed vector-cloud neural network with equivariance can be trained flexibly from data and meanwhile capture the nonlocal physics of tensor transport accurately. Among other potential applications, the work paves the way for further development towards frame-independent, nonlocal, data-driven Reynolds stress closure models for the RANS equations.

2. Problem statement and methodology

In this study, we intend to learn the nonlocal turbulence model for closing the Reynolds-averaged Navier–Stokes (RANS) equations. For industrial applications, the RANS models are still the workhorse as they do not resolve all scales and are much less expensive. The RANS equations are derived by performing Reynolds decomposition and ensemble averaging on the Navier–Stokes (N–S) equations, and are very similar to the original equations. However, the additional nonlinear terms in the momentum equations, namely Reynolds stress, are unknown and must be modeled. Here we propose using an equivariant neural operator to model the Reynolds stress term as a functional of the nonlocal mean flow fields instead of solving a transport PDE. We call it an operator because the proposed neural network model takes an discretized function as input and can adapt to different discretizations. The turbulence model in this work merely serves as an example of tensorial constitutive models described by transport PDEs, while the method is generally applicable for any nonlocal tensorial constitutive models.

For turbulent flows, the mean flow fields (velocity $\mathbf{u}$ and pressure p) are described by the following RANS equations:

$$\mathbf{u}\cdot\nabla\mathbf{u} - \nu\nabla^2\mathbf{u} = -\frac{1}{\rho}\nabla p + \nabla\cdot\mathcal{R}, \tag{1}$$

where ρ and ν is density and kinematic viscosity, respectively, and $\mathcal{R}$ is the Reynolds stress. To close the RANS equations, we model the Reynolds stress $\mathcal{R}$ by a constitutive transport equation of the following form such that $\mathcal{R}$ can be determined from the mean flow fields:

$$\mathbf{u}\cdot\nabla\mathcal{R} - \nabla\cdot[(\nu+\nu_t)\nabla\mathcal{R}] = \mathcal{P} + \Phi - E, \tag{2}$$

where ν_t denotes turbulent eddy viscosity, and $\mathcal{P}$, Φ, E denote production, pressure–strain-rate, and dissipation, respectively. The terms ν_t, Φ, E need to be additionally modeled (see examples in Section 3).

Considering both the nonlocal physics embodied in the transport PDE (2) and feasibility for implementation in CFD solvers, the network should form a region-to-point mapping $\langle \mathbf{q}(\boldsymbol{x}_0)\rangle \mapsto \mathcal{R}(\boldsymbol{x}_0)$ shown in Fig. 1, where $\mathbf{q}$ is the feature vector (we always assume column vectors in this paper), and $\langle \mathbf{q}(\boldsymbol{x}_0)\rangle$ indicates the collection of features $\{\mathbf{q}_i\}_{i=1}^n$ on n points sampled from the region around $\boldsymbol{x}_0$ (referred to as *cloud*). The extent of the cloud is determined by the velocity $\mathbf{u}$ at the point of interest according to the region of physical influence (see the detailed expression in [37]). The number of points n in a cloud can vary from location to location, which requires our model to adapt to different discretizations. The feature vector $\mathbf{q}$ attached to each point is chosen to include the relative coordinate $\boldsymbol{x}' = \boldsymbol{x} - \boldsymbol{x}_0$, flow velocity $\mathbf{u}$, and additional seven scalar quantities $\boldsymbol{c} = [\theta, \mathrm{u}, s, b, \eta, r, r']^\top$. These scalar features are the same as those used in [37], and we briefly recall their definitions for completeness.

(1) cell volume θ, which represents the weight for each point in mesh-based methods;
(2) velocity magnitude $\mathrm{u} = |\boldsymbol{u}|$ and strain rate magnitude $s = \|\boldsymbol{S}\| = \|\nabla\mathbf{u} + (\nabla\mathbf{u})^\top\|$, the latter of which often appears in various turbulence and transition models;
(3) boundary cell indicator b and wall distance η after normalization by boundary layer thickness scale δ^* and capped by 1. These two variables help the closure model to distinguish between the mapping described by the differential PDE and wall model (boundary condition);

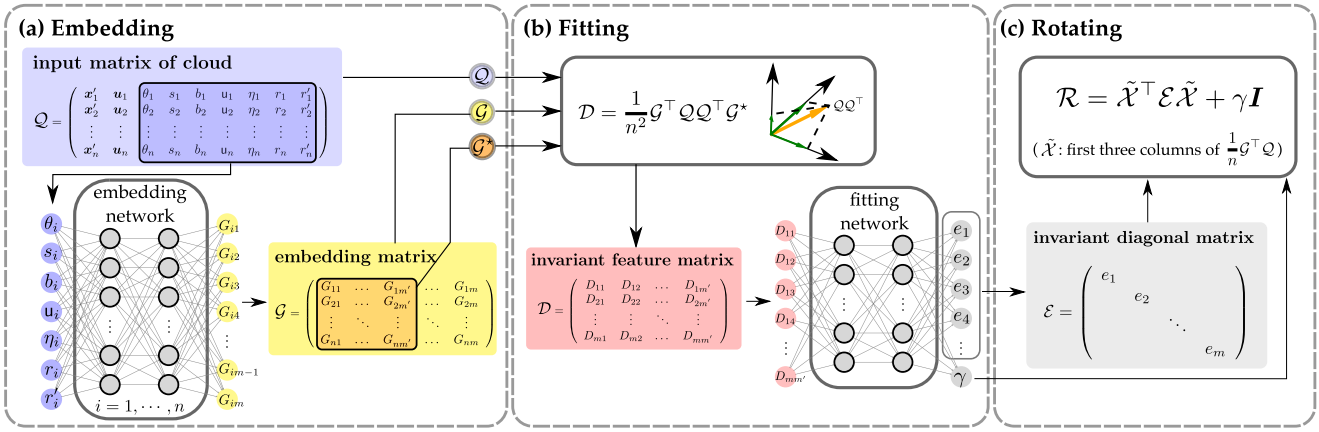


Fig. 2. Detailed architecture of the vector-cloud neural network with equivariance (VCNN-e) for predicting a targeting tensor $\mathcal{R}$: (a) embed the frame-independent features $\{\boldsymbol{c}_i\}_{i=1}^n$ to form the embedding matrix $\mathcal{G} \in \mathbb{R}^{n\times m}$; (b) project the pairwise inner-product matrix $\mathcal{Q}\mathcal{Q}^\top$ to the learned embedding matrix $\mathcal{G}$ and its submatrix $\mathcal{G}^\star$ to yield an invariant feature matrix $\mathcal{D} \in \mathbb{R}^{m\times m'}$; flatten and feed the feature matrix $\mathcal{D}$ into the fitting network to predict the diagonal matrix $\mathcal{E}$ and scalar γ; (c) rotate $\mathcal{E}$ through the embedded coordinates $\tilde{X}$ and get the final prediction of $\mathcal{R}$. The constitutive mapping $\mathbf{u}(\boldsymbol{x}) \mapsto \mathcal{R}$ based on VCNN-e is invariant to both the frame translation and the ordering of points in the cloud, and equivariant to the frame rotation. The VCNN-e differs from its predecessor VCNN [37] in that the developed rotating panel (c) generalizes scalar output to tensor and ensures its equivariance.

(4) proximity r (inverse of relative distance) to the center point of the cloud and proximity r' defined based on the angle between $-\mathbf{u}^\top$ and $\boldsymbol{x}'$ accounting for the alignment between the convection and the relative position of the point in its cloud.

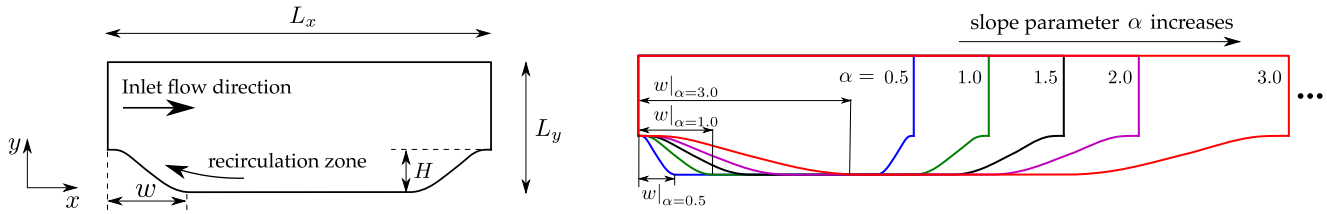
According to the above definition, the feature vector $\mathbf{q}$ is $3+3+7=13$ dimensional, and we define the input data matrix as $\mathcal{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_n]^\top \in \mathbb{R}^{n\times 13}$. Note that all the input features are first non-dimensionalized by the characteristic scales and then fed into the neural network. As such the closure model can handle dynamically similar systems, e.g., training and predicting systems at similar Reynolds numbers despite with vastly different length and time scales.

The vector-cloud neural network with equivariance (VCNN-e) for modeling Reynolds stress tensor consists of three modules at a high level: an embedding module, a fitting module, and a rotating module. The goal of the first module (Fig. 2a) is to find a representation of the input data that is translational, rotational, and permutational invariant. To this end, we introduce a set of m basis functions $\{\phi_k(\boldsymbol{c}_i)\}_{k=1}^m$, implemented by an embedding neural network with m-dimensional output, for the scalar quantities $\boldsymbol{c}_i$ (note the input $\boldsymbol{c}_i$ is the frame-independent part of the feature vector $\mathbf{q}_i$) and define $\mathcal{L}_{kj} = \frac{1}{n}\sum_{i=1}^n \phi_k(\boldsymbol{c}_i)\,\mathcal{Q}_{ij}$, $k = 1, \ldots, m$, $j = 1, \ldots, 13$. The summation over the point index i removes the dependence of $\mathcal{L}$ on the ordering and makes it permutational invariant. The normalization by n allows a different number of sampled points in the cloud in the training or testing data and guarantees the resulting VCNN-e is resolution adaptive. If we define a matrix $\mathcal{G}_{ik} = \phi_k(\boldsymbol{c}_i)$, the permutation invariant transformation above can be written as $\mathcal{L} = \frac{1}{n}\mathcal{G}^\top\mathcal{Q}$. Similarly, we define $\mathcal{L}^\star = \frac{1}{n}\mathcal{G}^{\star\top}\mathcal{Q}$, with $\mathcal{G}^\star$ being the first m' columns of $\mathcal{G}$, and $\mathcal{L}^*$ is also permutational invariant. Here we choose $\mathcal{G}^\star$ as a subset of $\mathcal{G}$ instead of $\mathcal{G}$ itself mainly in order to save the computational cost in the next step without sacrificing the accuracy. Next, we define $\mathcal{D} = \mathcal{L}\mathcal{L}^{\star\top} = \frac{1}{n^2}\mathcal{G}^\top\mathcal{Q}\mathcal{Q}^\top\mathcal{G}^\star$, which is translational, rotational, and permutational invariant since both $\mathcal{L}$ and $\mathcal{L}^{\star\top}$ are. We can view $\mathcal{D}$ as a faithful extraction of information from the feature vectors in the cloud via projecting $\mathcal{Q}\mathcal{Q}^\top$ onto the learned basis $\mathcal{G}$. Then the second module (Fig. 2b) takes $\mathcal{D}$ as the input and fits an invariant diagonal matrix $\mathcal{E}$. We construct a fitting network $f_{\text{fit}} : \mathcal{D} \mapsto [e_1, \cdots, e_m, \gamma]^\top$ that maps $\mathcal{D}$ to $(m+1)$-dimensional output, and define a diagonal matrix $\mathcal{E} \in \mathbb{R}^{m\times m}$ taking $e_1, \cdots, e_m$ as the diagonal elements. Finally, in the third module (Fig. 2c), we re-incorporate the embedded coordinates $\tilde{\mathcal{X}} = \frac{1}{n}\mathcal{G}^\top\mathcal{X} \in \mathbb{R}^{m\times 3}$ from the first module where $\mathcal{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n\times 3}$ and rotate properly to get the final tensor prediction through

$$\mathcal{R} = \tilde{\mathcal{X}}^\top\mathcal{E}\tilde{\mathcal{X}} + \gamma\boldsymbol{I}, \tag{3}$$

where $\boldsymbol{I}$ is the identity tensor. Since $\tilde{\mathcal{X}}$ rotates in the same way as the original coordinates under rotation and $\boldsymbol{I}$ is rotational equivariant, the final output $\mathcal{R}$ preserves the desired equivariance. The optimal parameters in the embedding and fitting networks are trained from the data. The detailed proof of the equivariance property of the formulation in Eq. (3) is shown in Appendix A.

The formulation in Eq. (3) is inspired by the work of [38] in the context of molecular dynamics, which only has the first term $\tilde{\mathcal{X}}^\top\mathcal{E}\tilde{\mathcal{X}}$ satisfying the equivariance requirements. However, here we need to add an extra term $\gamma\boldsymbol{I}$ to ensure that the network correctly represents the function space and symmetries that are unique for turbulence modeling. Specifically, these requirements are:



(a) baseline configuration (b) configurations of varying slopes

Fig. 3. Configurations of computational domains used for training and testing, showing (a) the baseline configuration in the xy-plane and (b) parameterized configurations with varying slopes parameters α, defined as the ratio of the width w of the hill to that of the baseline hill with $\alpha = 1$ (i.e., $w|_{\alpha=1}$). The x- and y- coordinates are aligned with streamwise and wall-normal directions, respectively, and the z-coordinate is determined based on the right-hand rule (perpendicular to the paper and pointing outward). The slope parameters α are selected within the ranges [1, 2] and [0.5, 4] for training and testing flows, respectively.

(1) The Reynolds stress tensor has a full rank of three even for statistically two-dimensional (or even one- or zero-dimensional) mean flows, with all three diagonal components being nonzero in general. In contrast, the first part $\tilde{\mathcal{X}}^\top \mathcal{E} \tilde{\mathcal{X}}$ alone degenerates for such flows: it only has a rank of two for two-dimensional mean flows.
(2) For two-dimensional mean flows, the out-of-plane stresses should be zero. This requirement is correctly satisfied by the first part $\tilde{\mathcal{X}}^\top \mathcal{E} \tilde{\mathcal{X}}$.

The additional term $\gamma \boldsymbol{I}$ enables the network to satisfy the first requirement of function space without breaking its equivariance property or the symmetries implied in the second requirement. Moreover, despite the apparent formal similarity of the formulation to the deviatoric decomposition of the Reynolds stress tensor, we note that our formulation is motivated mathematically rather than physically. Further details on the motivation and interpretation of the VCNN-e model are presented in Appendix B.

With the three modules of the VCNN-e introduced above, the model provides us with a flexible representation of the Reynolds stress tensor based on the nonlocal information, and all the physical symmetries are faithfully guaranteed. The trainable parameters of the VCNN-e model include the network parameters in the embedding network and fitting network. We use the squared difference between the predicted stress and the ground truth as the loss function to optimize the parameters in the VCNN-e model. The details of data generation and training procedure will be introduced in the following sections, and the detailed architecture for the embedding and fitting networks are provided in Appendix C. Specifically, we choose $m = 64$ embedding functions and their subset of size $m' = 4$ in $\mathcal{G}^\star$ to extract feature matrix $\mathcal{D}$. Similarly to the results reported in [36,37], we find the prediction accuracy is insensitive to m' when m' is much less than m. We set $m = 64, m' = 4$ to preserve more features (mainly determined by m) in the invariant feature matrix and reduce the computational cost (mainly determined by $m \times m'$ as the input dimension) of the fitting network.

3. Emulating a Reynolds stress transport equation with closure models

In this section, we emulate a Reynolds stress transport equation with a frozen velocity field on the parameterized periodic hills, and take the associated solutions as the ground truth. Mathematically, the ground truth model defines an exact mapping from the mean velocity field to the Reynolds stress (if we do not consider any domain truncation error or discretization error). Such a setting enables us to focus on studying the learning performance of the proposed nonlocal closure model.

3.1. Generation of training data

We consider predicting the full Reynolds stress tensor in the flows over periodic hills. The baseline configuration of the computational domain in the xy-plane (normal to z-axis) has the length $L_x/H = 9$ and the height $L_y/H = 3.035$, both of which are normalized by the crest height H, which is shown in Fig. 3a. The profile of the hill is described as piecewise polynomials with width $w/H = 1.93$. We systematically vary the slope parameter α of the hill profile to generate a family of geometry configurations for training and testing [39], as is shown in Fig. 3b. The slope parameter α is defined as the ratio of the width w of the parameterized hill to that of the baseline hill $w|_{\alpha=1.0}$. The height of the hill remains the same for all configurations, and thus the hill gets more gentle with increasing slope parameter α. For training flows, we select 11 configurations with $\alpha = 1.0, 1.1, \ldots, 2.0$. For testing flows, the slope parameters α are within the range from 0.5 to 4.

We generate data by solving a Reynolds stress transport equation with the LRR-IP model (a second-moment closure model of Launder, Reece, and Rodi with isotropization of production [40]):

$$\mathbf{u} \cdot \nabla \mathcal{R} - \nabla \cdot [(\nu + \nu_t) \nabla \mathcal{R}] = \mathcal{P} + \Phi - E, \tag{4a}$$

where the left-hand side includes convection and diffusion, and the right-hand side consists of three tensor-based terms: production $\mathcal{P}$, pressure–strain-rate Φ, and dissipation tensor E. The production term is closed, while the pressure–strain-

rate tensor Φ consists of a slow component modeled by Rotta's return-to-isotropy model and a rapid component modeled by the isotropization of production model [41]. These two terms can be written as follows:

$$\mathcal{P} = -\left[\mathcal{R}\cdot\nabla\mathbf{u} + (\mathcal{R}\cdot\nabla\mathbf{u})^{\top}\right], \tag{4b}$$

$$\Phi = \frac{C_1}{\tau}\,\mathrm{dev}(\mathcal{R}) + C_2\,\mathrm{dev}(\mathcal{P}), \tag{4c}$$

where $\mathrm{dev}(\cdot)$ indicates the deviatoric component of a tensor, "$\cdot$" denotes inner product between tensors, and $\tau = k/\varepsilon$ is the turbulence time scale. The dissipation tensor E is assumed to be isotropic, i.e.,

$$E = \frac{2}{3}\varepsilon\boldsymbol{I} \quad \text{with} \quad \varepsilon = C_D\frac{k^{3/2}}{\ell_m}, \tag{4d}$$

where $\boldsymbol{I}$ is the second order identity tensor, and ε is the dissipation rate scalar estimated from the turbulence kinetic energy (TKE) k and mixing length ℓ_m. k is defined as half the trace of the Reynolds stress: $k \equiv \frac{1}{2}\mathrm{tr}(\mathcal{R})$, and ℓ_m is assumed to be proportional to wall distance η and capped by the boundary layer thickness δ^*, i.e.,

$$\ell_m = \min(\kappa\eta, C_\mu\delta^*) \tag{4e}$$

with von Karman constant $\kappa = 0.41$. The boundary layer thickness is set to be $\delta^*/H = 0.5$ for all cases. Other model coefficients are chosen as $C_\mu = 0.09$, $C_D = C_\mu^{3/4}$, $C_1 = 1.8$, $C_2 = 0.6$. The boundary layer thickness is set based on the geometry for calculating the mixing length ℓ_m in equation (4e). Finally, the turbulent eddy viscosity is modeled by $\nu_t = C_\mu k^2/\varepsilon$. Although the Reynolds stress transport equation defined above is in a dimensional form with physical units, the inputs of the VCNN-e are dimensionless. As such, the VCNN-e can be trained with and applied to flows with different scales, as long as they have dynamical similarity. A detailed discussion is provided in Appendix D.

We note that equations (4a)–(4e) by no means represent a complete differential Reynolds stress model since the closure terms like the diffusion and the dissipation are obtained with massively simplified models (comparable to those in incomplete one-equation models [41]). However, the equations capture much of the mathematical features of the Reynolds stress transport equation, including convection, diffusion, production, and dissipation. Most importantly, it includes a model for the pressure–strain-rate tensor, which is responsible for energy transfer among different components of the tensor $\mathcal{R}$. This tensor is notoriously difficult to model and is a pacing item in developing differential Reynolds stress models. So we treat equations (4a)–(4e) as a prototype closure model for the Reynolds stress $\mathcal{R}$. Instead of modeling and solving (4a)–(4e) to get the steady solution of $\mathcal{R}$, we use VCNN-e to construct and learn a nonlocal closure model to map from the velocity field to the stress tensor directly.

We calculate the steady Reynolds stress $\mathcal{R}(\boldsymbol{x})$ by solving the transport equation PDE (4a)–(4e) with a steady, incompressible flow field $\mathbf{u}(\boldsymbol{x})$. The steady-state flow is driven by a constant pressure gradient such that the Reynolds number based on the volume averaged bulk velocity magnitude u_b, the hill height H, and the kinematic viscosity ν attains a specified value $Re = u_b H/\nu = 10595$. We assume the flow is statistically homogeneous in the z-direction so that the z component of the mean velocity is 0. However, note that the instantaneous velocity fluctuations and thus the Reynolds normal stress $\mathcal{R}_{zz}$ in the z-direction is nonzero. The simulations of flow fields and Reynolds stress transport are performed with an open-source CFD platform OpenFOAM [42]. Firstly, we simulate the flows over the periodic hills by solving the RANS equations with k–ε turbulence model using the built-in steady-state incompressible flow solver simpleFoam [43]. Then, the simulated flow field and the corresponding Reynolds stress field are treated as the frozen flow field and initial condition, respectively, for solving the transport equation PDE (4a). When solving the RANS equations and Reynolds stress transport equation, no-slip boundary conditions are applied for the velocity at walls, wall functions are used for Reynolds stress and its associated turbulent quantities in the near-wall region, and periodic boundary conditions are applied in the streamwise direction. The numerical discretization based on the finite volume method and second-order spatial discretization scheme is utilized to solve the equations. Specifically, we make the number of cells approximately proportional to the length L_x of the configuration in the x-direction (e.g., 200 cells for the baseline configuration with $L_x/H = 9$), while in the y-direction the cells are refined towards the walls and the number is fixed at 200 for all configurations.

Our aim is to learn a region-to-point mapping from a patch of nonlocal mean flow field $\langle\mathbf{u}(\boldsymbol{x})\rangle$ to the steady Reynolds stress tensor $\mathcal{R}$ at the point of interest. As mentioned above, the nonlocal flow field is described by the input feature matrix $\mathcal{Q}$ based on n points sampled from the cloud. The generation of such pairs of data $(\mathcal{Q}, \mathcal{R})$ in different locations is illustrated in Fig. 4. The grey dots (•) indicate all the discrete points (i.e., the cell centers) within the computational domain, which are shown every seventh row and every third column for clarity; the star (⋆) indicates the point of interest at which the Reynolds stress tensor $\mathcal{R}$ is to be predicted; n data points (•) are randomly sampled within the vector cloud (⬭) to represent the nonlocal flow field used for prediction of the Reynolds stress tensor $\mathcal{R}$. The extent and orientation of the elliptical cloud vary at different locations: The extent of the ellipse is determined by the velocity magnitude $|\mathbf{u}|$ at the point of interest; the orientation (i.e., the major axis) of the ellipse aligns with the direction of the velocity $\mathbf{u}$. Specifically, the length ℓ_1 of the semi-major axis and ℓ_2 of the semi-minor axis are defined according to the previous work [44]:

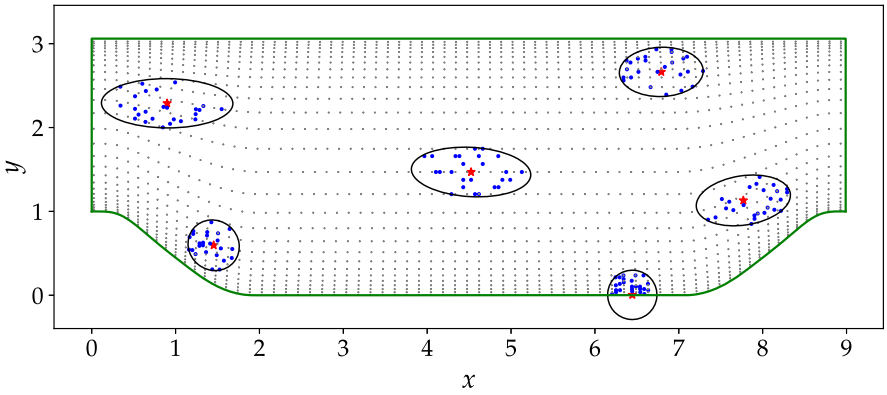


Fig. 4. Method of sampling data points within the cloud to generate pairs of labeled training data $(\mathcal{Q}, \mathcal{R})$. The gray dots (•) indicate all the discrete points (i.e., the cell centers) within the computational domain, showing only every seventh row and every third column for clarity. The star (⋆) indicates the point of interest where the Reynolds stress tensor $\mathcal{R}$ is to be predicted. The surrounding ellipse denotes the region of cloud, whose extent and orientation are determined by the velocity at the cloud center (⋆). The dots (•) are randomly sampled within the cloud, and the set of all feature vectors attached to them is taken as the input matrix $\mathcal{Q}$ to predict $\mathcal{R}$.

$$\ell_1 = \left| \frac{2C_\nu \log \epsilon}{\sqrt{|\mathbf{u}|^2 + 4C_\nu C_\zeta} - |\mathbf{u}|} \right| \quad \text{and} \quad \ell_2 = \left| \sqrt{\frac{C_\nu}{C_\zeta}} \log \epsilon \right|, \tag{5}$$

where C_ν and C_ζ are diffusion and dissipation coefficients in the 1-D steady-state convection-diffusion-reaction equation $-C_\nu \frac{\mathrm{d}^2}{\mathrm{d}x^2} c(x) + \boldsymbol{u} \frac{\mathrm{d}}{\mathrm{d}x} c(x) + C_\zeta c(x) = P(x)$, and $\epsilon (= 0.2)$ denotes relative error tolerance for truncating the associated Green's function. For the transport equation (4a) in this study, the actual dissipation coefficient C_ζ in Eq. (5) should be approximately $\frac{2}{3} C_D \frac{k^{1/2}}{\ell_m}$, considering the dissipation term $\mathcal{E}$. Here, it is assumed as a constant $C_\zeta \approx 2$ to determine the extent of the clouds, where the maximum ℓ_m and k are used for the approximation. Similarly, the actual diffusion coefficient C_ν in Eq. (5) is chosen as $(\nu_t + \nu)_{\min} \approx 0.01$.

After determining the extent of clouds, we randomly sample $n = 300$ data points within each cloud to generate training data. The available data points are repeatedly sampled for the locations where the number of data points in the ellipse is smaller than 300. It should be noted that the proposed neural network is flexible with an arbitrary number of sampled points in the cloud. Here we set n as a constant for training in order to conveniently store the data and process them in batches for better computational efficiency during training. There are 6.6×10^5 pairs of $(\mathcal{Q}, \mathcal{R})$ in total for 11 training flows with $\alpha = 1.0, 1.1, \ldots, 2.0$. Considering that the adjacent data pairs from the same flow may appear similar, we take one from every two adjacent pairs to reduce the size of the training dataset and use the reduced 3.3×10^5 pairs of $(\mathcal{Q}, \mathcal{R})$ as the training data. As mentioned above, the data are non-dimensionalized by the characteristic length H, flow velocity u_b, and time H/u_b, and then fed into the neural network.

3.2. Neural-network-based prediction of Reynolds stress tensors

With the generated training data described above, one can directly train the VCNN-e model in a straightforward way. However, we employ a different progressive training strategy here to boost training efficiency. The network is initially trained using sparse data (i.e., a small stencil size), followed by dense data. This training method achieves better performance due to the accelerated optimization process compared to the conventional training method and can serve as a general training method for neural operators with resolution adaptivity, which is detailed in Appendix E. The progressive training processes are performed on an NVIDIA RTX 3090 GPU using the open-source machine learning framework PyTorch [45], which takes about 15.4 hours for 5000 epochs. The code for data generation and model training are available on GitHub [46], which can be used by the readers for reproducing the results and further development.

After training, the prediction capabilities of the trained neural-network-based model are investigated in different configurations with slope parameters α between 0.5 and 4. In the testing step, we take all the data points within the clouds (i.e., full stencil size varying from 50 to 2000 points) to predict the Reynolds stress tensor at the cloud centers, which is different from using only 300 sampled points in each cloud for prediction in the training step. In this sense, testing on the same flow used for training is still compelling because the input are different. We examine how well the learned model performs toward capturing two turbulent quantities: (1) turbulence kinetic energy (TKE) k and (2) Reynolds stress anisotropy invariant A_2 (detailed below), both of which are invariants of Reynolds stress tensor. Note that the loss function for training the network only consists of the differences in Reynolds stress components, while the invariant quantities, such as the TKE and the anisotropy scalar, are not included. Accurate predictions of these invariant quantities rely on the equivariant property of the proposed VCNN-e model.

The predicted TKE k fields are nearly identical to the corresponding ground truths in two extreme interpolated flows with $\alpha = 1$ and 2, which is illustrated in Fig. 5. The TKE is calculated based on the predicted Reynolds stress and then compared

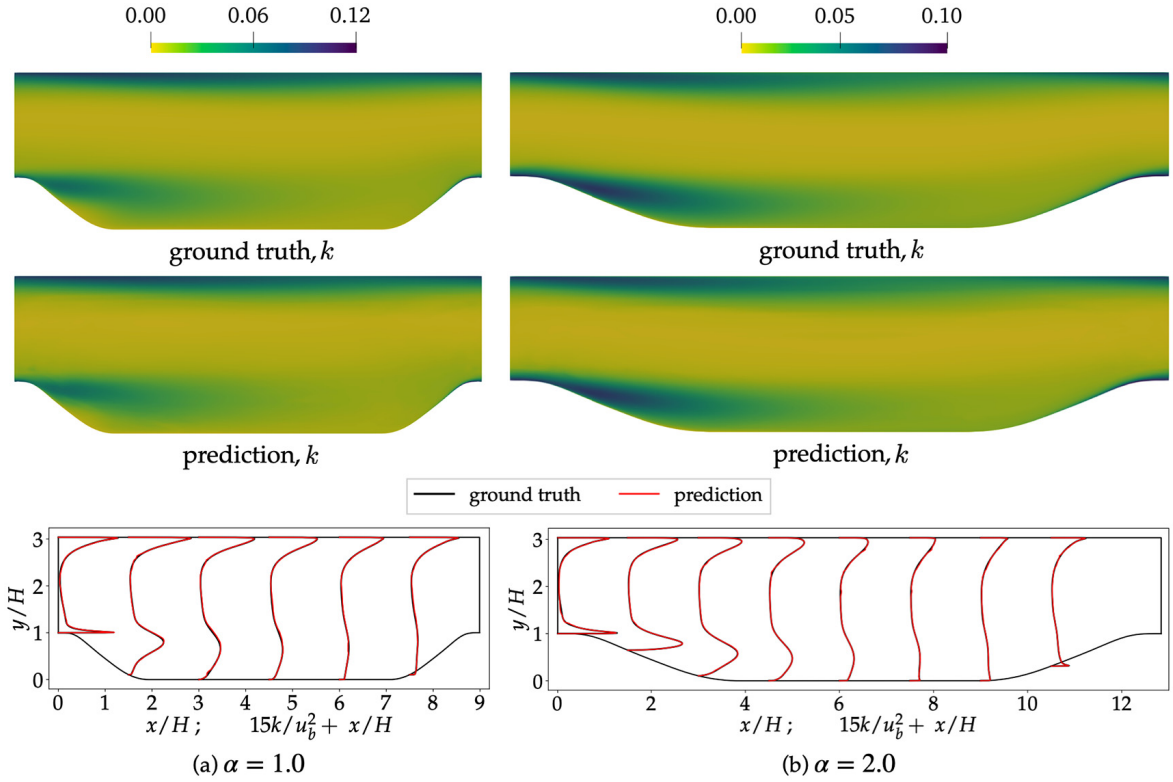


Fig. 5. Comparison of the ground truths of the **turbulence kinetic energy** (TKE) k (top row) and the corresponding predictions based on the predicted Reynolds stress from the trained neural network (middle row), along with the TKE profiles at six and eight cross-sections (bottom row) for two **interpolated** configurations with slope parameters $\alpha = 1$ (left panels) and $\alpha = 2$ (right panels). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

with the ground truth. We can observe similar patterns between the predicted TKE and the ground truths (top two rows). The similarity is shown more clearly at the vertical cross-sections (bottom row): The predicted TKE profiles coincide with those of the corresponding ground truths. Such good prediction performance in the interpolated flows lies in the fact that the method respects the transport physics of turbulent quantities by considering the mean flow field in a nonlocal region. Moreover, 300 randomly sampled points in each cloud are sufficient to describe the flow, allowing the network to learn the nonlocal mapping and make accurate predictions.

The prediction of TKE for two extreme extrapolated flows with $\alpha = 0.5$ and 4 is not as accurate as that for the interpolated flows, but they are still reasonable with expected features being captured. As illustrated in Fig. 6, the nonlocal model is able to predict the TKE in the boundary layer at the top wall for both flows. However, despite some similarity with the ground truth in the near-hill region, the flow with $\alpha = 0.5$ exhibits distinct inconsistency and artifacts due to the steepest hill slope and the large flow separation. In contrast, the flow with $\alpha = 4.0$ shows a better prediction near the bottom wall because of the gentlest hill slope. The comparison is more clearly illustrated in the profile plots at cross-sections (bottom row). The predicted TKE in the flow with $\alpha = 0.5$ is overestimated on the leeside of the hill, whereas the predicted TKE in the flow with $\alpha = 4.0$ is nearly identical to the ground truth despite some unsmoothness.

Another evaluation criterion of predictive capability is the Reynolds stress anisotropy invariant A_2 as in practical flows the Reynolds stress tensor is anisotropic due to deformation of large eddies by mean strain, inhomogeneities and boundaries. The invariant A_2 is defined as half the square of the tensor magnitude of normalized Reynolds stress anisotropy $\boldsymbol{b}$ given by [47]:

$$A_2 = b_{ij} b_{ji}/2, \quad \text{with} \quad b_{ij} = \frac{\mathcal{R}_{ij}}{2k} - \frac{1}{3}\delta_{ij}, \tag{6}$$

where k is the TKE and δ_{ij} is the Kronecker delta. Here we scale up the invariant A_2 by multiplying the TKE k, considering that A_2 of anisotropy tensor is no longer important when k is sufficiently small. The predicted kA_2 in two extreme interpolated flows with $\alpha = 1$ and 2 are quite accurate with nearly no deviation from the ground truths, as is shown in Fig. 7. The nonlocal model is able to capture a more rapid change of A_2 near walls compared to that of TKE. For extrapolated flows, however, we find more deviations than interpolated flows, with the similar reasons discussed above for the results of the predicted TKE.

From the above results, we can see that the hill slope significantly impacts the prediction performance of the trained model. For further investigation, we evaluate the generalizability of the trained model on 21 configurations with varying

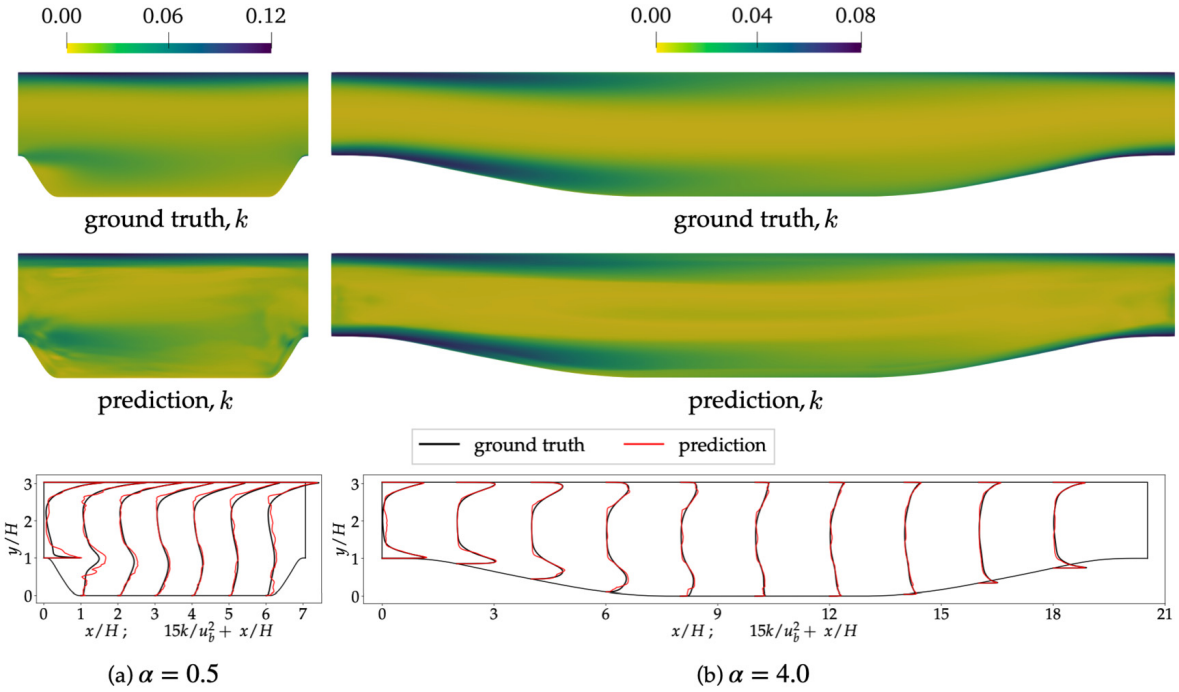


Fig. 6. Comparison of the ground truths of the **turbulence kinetic energy** k (top row) and the corresponding predictions based on the predicted Reynolds stress from the trained neural network (middle row), along with the TKE profiles at seven and ten cross-sections (bottom row) for two extreme **extrapolated** configurations with slope parameters $\alpha = 0.5$ (left panels) and $\alpha = 4.0$ (right panels).

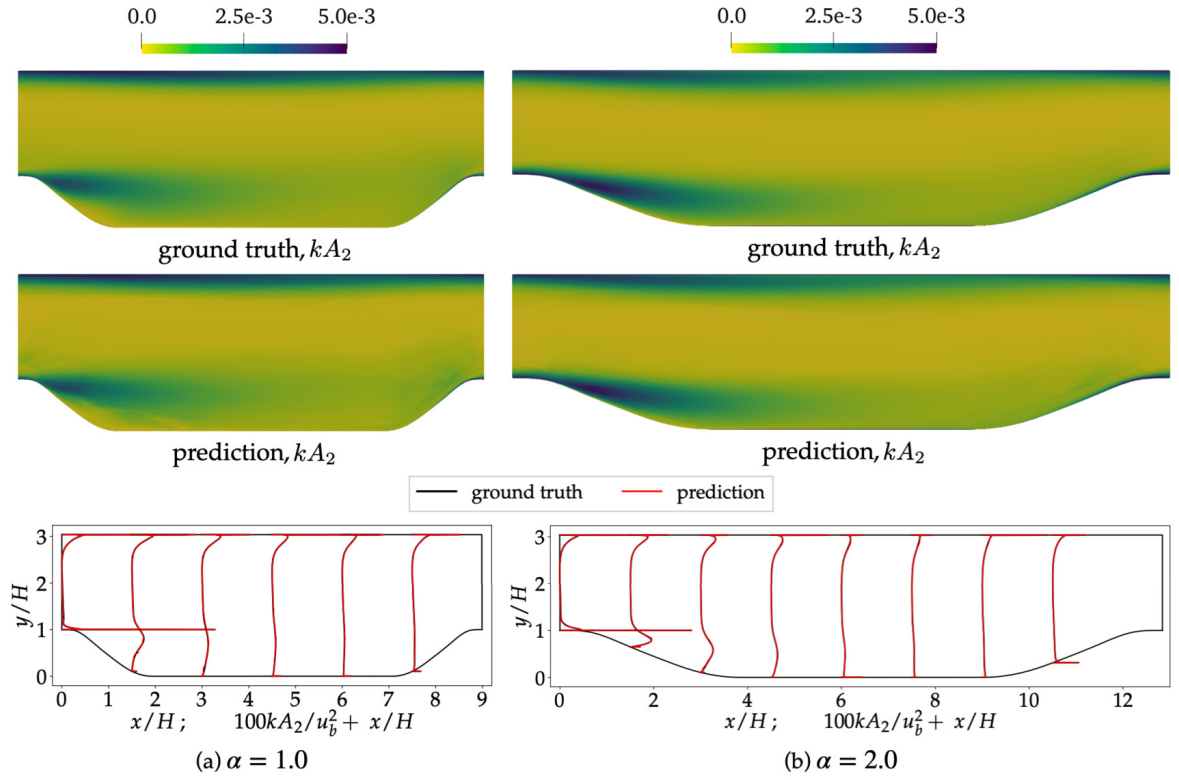


Fig. 7. Comparison of the ground truths of kA_2 (top row), the **Reynolds stress anisotropy invariant** up scaled by the turbulent kinetic energy, and the corresponding predictions (middle row) based on the predicted Reynolds stress from the trained neural network, along with the kA_2 profiles at several cross-sections (bottom row) for two **interpolated** configurations with slope parameters $\alpha = 1$ (left panels) and $\alpha = 2$ (right panels).

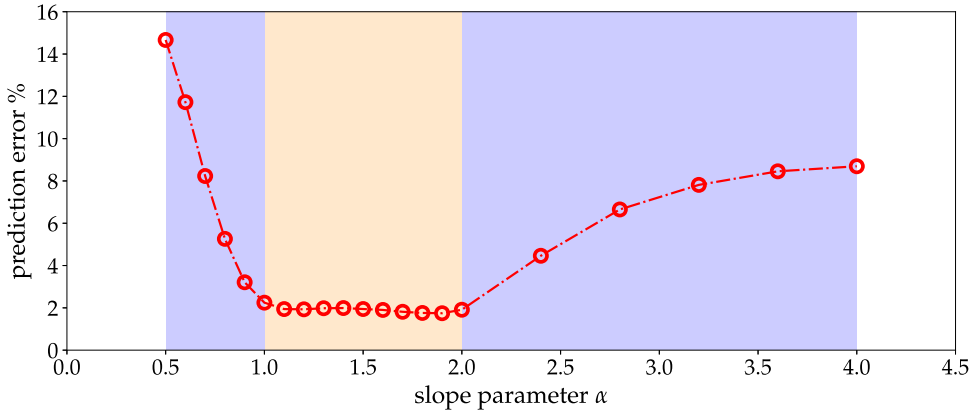


Fig. 8. Predictive errors for turbulence kinetic energy k at various slope parameters α. The yellow/lighter and blue/darker backgrounds represent the regimes of the slope parameters α of the interpolated and extrapolated testing flows, respectively. The neural network is trained with data of a fixed stencil size from 11 flows with $\alpha = 1, 1.1, \ldots, 2$. The trained network is then tested on the data of full stencil size from the same 11 interpolated flows and 10 extrapolated flows with $\alpha = 0.5, 0.6, \ldots, 0.9$ and $\alpha = 2.4, 2.8, \ldots, 4$.

slope parameters α from 0.5 to 4. The prediction performance is assessed using the prediction error, which is defined as the normalized ℓ^2-norm discrepancy between the calculated TKE $\hat{k}$ based on the predicted Reynolds stress and corresponding ground truth k^*:

$$\text{error} = \frac{\sqrt{\sum_{l=1}^{N} |\hat{k}_l - k_l^*|^2}}{\sqrt{\sum_{l=1}^{N} |k_l^*|^2}}, \tag{7}$$

where the summation is performed on all of the N training or testing data points (e.g., 40000 data points for baseline configuration with $\alpha = 1$). Again, the prediction is made using the full stencil size, which differs from using a fixed number of sampled points in the training step, so the flows with α between 1 and 2 can still serve as the interpolated testing flows. The prediction errors for all 21 testing flows are illustrated in Fig. 8. The middle (yellow/light gray) region from 1 to 2 represents the regime of slope parameters α of training flows while that on both sides (blue/dark) represents the regime of extrapolated testing flows. The prediction performance on 11 interpolated testing flows is the best, with prediction errors being 1.8%, showing the trained model can accurately predict the Reynolds stress tensor based on a nonlocal flow field. When extrapolating to flows with steeper ($\alpha < 1$) or more gentle ($\alpha > 2$) hill profiles, the prediction errors increase significantly to 14.7% for $\alpha = 0.5$ and 8.7% for $\alpha = 4$, which is reasonable and can be partially explained by the changes of input and output distributions displayed subsequently.

We visualize the input and output data distributions in two directions of extrapolation through the kernel density plot of the variable pairs with different α in Fig. 9: velocities (u/v) in x/y-directions in the first row and turbulence kinetic energy k/anisotropy invariant A_2 in the second row. We can see that in subfigure (a) (the first column), the interpolation regime, both the input data u, v and output-dependent data k, A_2 have similar distributions as α change values from 1 to 1.5, and to 2. In subfigure (c) (the last column), the data distribution corresponding to $\alpha = 4$ keeps the trend of change and deviates from the interpolation data a little bit. In contrast, in subfigure (b) (the middle column), the data distribution corresponding to $\alpha = 0.5$ presents some data points that are more significantly away from the training data.

4. Results on a dataset from direct numerical simulations

In this section, we consider using the mean flow fields collected from direct numerical simulations to predict the corresponding Reynolds stress. Compared to the setting in Section 3, this setting is much closer to the practice of developing and calibrating closure models and hence provides a more realistic testbed for the proposed nonlocal closure model. However, it should be noted, due to the information loss in the Reynolds averaging, there may not exist an exact mapping from the mean flow fields to the Reynolds stress. This is evident from the exact transport equations for the Reynolds stresses [3], where the turbulent diffusion depends on the triple correlation of the turbulent fluctuations while the pressure-strain-rate term also depends on the pressure fluctuations. However, in RANS models such information is not available and the corresponding term must be modeled as functions of the mean field. Using only the mean flow fields as the input without turbulent fluctuation data can result in modeling inaccuracy. So the objective here is to investigate the prediction performance of the proposed nonlocal model in the presence of such a modeling error.

4.1. Dataset for nonlocal turbulence modeling from direct numerical simulations

A new dataset of flows over parameterized periodic hills from direct numerical simulations (DNS) is employed for the further evaluation of VCNN-e in modeling the Reynolds stress. Direct numerical simulations are performed on 29 param-

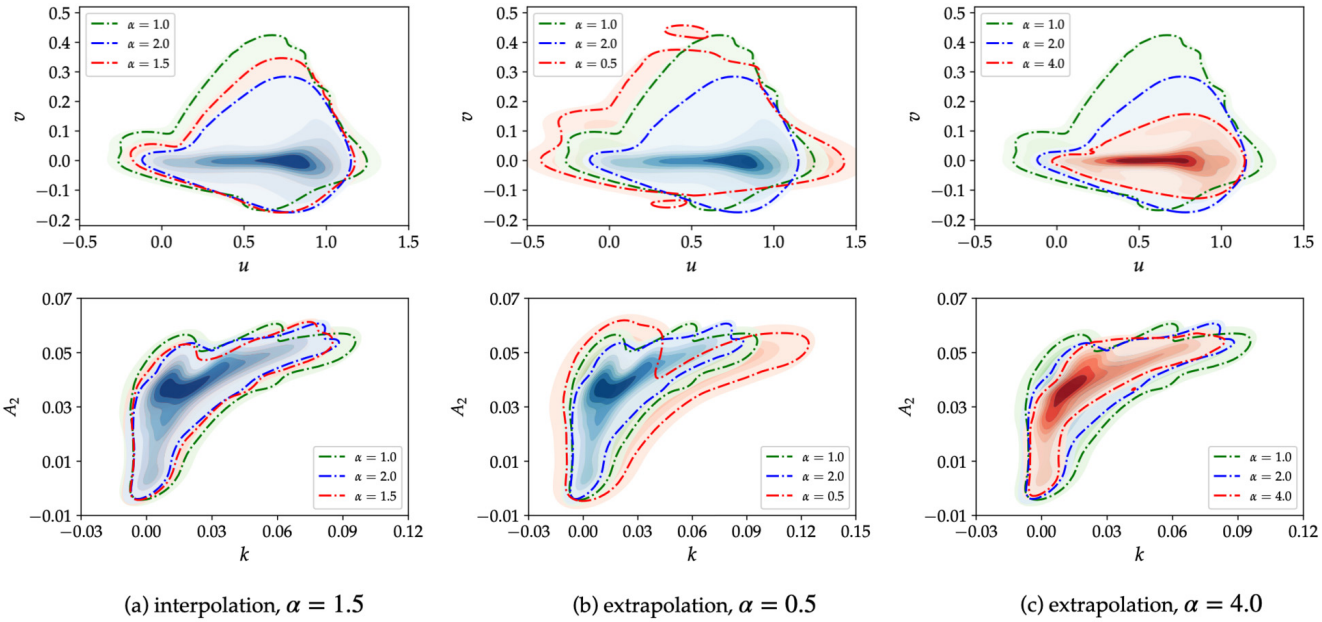


Fig. 9. First row: kernel density estimate (KDE) plot of the joint data distribution of the velocities in x/y directions (u/v) with different slope parameters α, in the regimes of interpolation and extrapolation on two sides. Second row: KDE of the joint data distribution of the turbulence kinetic energy k and anisotropy invariant A_2 with different slope parameters α, in the regimes of interpolation and extrapolation on two sides. The dash-dotted line represents the contour line which envelops 99% of the probability mass of the data.

eterized periodic hills with five different hill slopes ($\alpha = 0.5, 0.75, 1.0, 1.25, 1.5$), three different domain lengths ($L_x/H = 3.858\alpha + 2.142, 3.858\alpha + 5.142, 3.858\alpha + 8.142$) and heights ($L_y/H = 2.024, 3.036, 4.048$). In this study, we select five flows with varying hill slopes for training and testing, in which their domain lengths and height are $L_x/H = 3.858\alpha + 5.142$ and $L_y/H = 3.036$, respectively. The DNS data is generated by Sylvain Laizet and his coworkers using the high-order flow solver `Incompact3d` [39,48,49], which consists of the mean velocity, pressure, and Reynolds stress fields. The DNS data is then mapped to their counterparts on a coarser mesh to generate the training and testing data in this situation. For example, in flow with $\alpha = 0.5$, the DNS data on fine mesh ($640 \times 385 \times 192$) is first averaged in the spanwise direction (z) and then interpolated to the data on a coarse mesh (157×150).

After the data preprocessing, the generation of the training and testing data follows a similar method to that in Section 3. Specifically, the cloud extent is based on the local velocity and determined by Eq. (5) with the numerical values of ϵ, C_v, and C_ζ being 0.2, 0.02, and 2, respectively; the cloud orientation (i.e., long axis) aligns with the local velocity. A constant stencil size of 300 is applied for sampling in each cloud. In this section, four different flows with $\alpha = 0.5, 0.75, 1.25, 1.5$ are used for training and the flow with $\alpha = 1.0$ for testing. Finally, we have 1.2×10^5 pairs of $(\mathcal{Q}, \mathcal{R})$ as the training data and 3×10^4 pairs as the testing data. Considering the significant reduction in the amount of training data compared to the synthetic situation, the VCNN-e is trained straightforwardly rather than in a progressive way.

4.2. Neural-network-based prediction of Reynolds stress tensors

The predicted Reynolds stress in the testing flow exhibits a high degree of agreement with the ground truth. Comparison of the predicted Reynolds stress components ($\mathcal{R}_{xx}$, $\mathcal{R}_{yy}$, $\mathcal{R}_{zz}$, and $\mathcal{R}_{xy}$) and their corresponding ground truths is shown in Fig. 10. The predictions (middle row) of four components exhibit similar patterns to the ground truths (top row). The similarity is further illustrated by the consistency of their profiles on six cross-sections ($x/H = 0, 1.5, \cdots, 7.5$). Such consistency further demonstrates the capability of VCNN-e to model the Reynolds stress in a more realistic and complex scenario in which the transport is driven by the original Navier-Stokes equations rather than an explicit transport PDE. Note that, when working with a synthetic PDE in the former section, there is an exact mapping between mean flow fields and Reynolds stress, but this may not be the case for the DNS situation. The consistency is further demonstrated by the predicted turbulent anisotropy states in Appendix F. The prediction over three cross-sections exhibits a much more similar distribution to the DNS data in the Barycentric triangle than the RANS result. In addition to achieving high modeling accuracy, it is essential for the prediction from mean flow field to Reynolds stress field to be efficient, as it must be seamlessly coupled with the RANS equations [50]. Notably, in this case, one-time prediction takes approximately 0.6 seconds for the testing flow with $\alpha = 1.0$, indicating that the method has computationally efficiency and can be further optimized by implementing parallel operations on GPUs. This enables the approach to be used for large-scale simulations and facilitates the exploration of complex fluid flow phenomena in engineering applications.

Given the success of VCNN-e in predicting the Reynolds stress in the DNS dataset, we further examine the role of nonlocality in the modeling. To this end, we train a network using only local input features and then compare the local

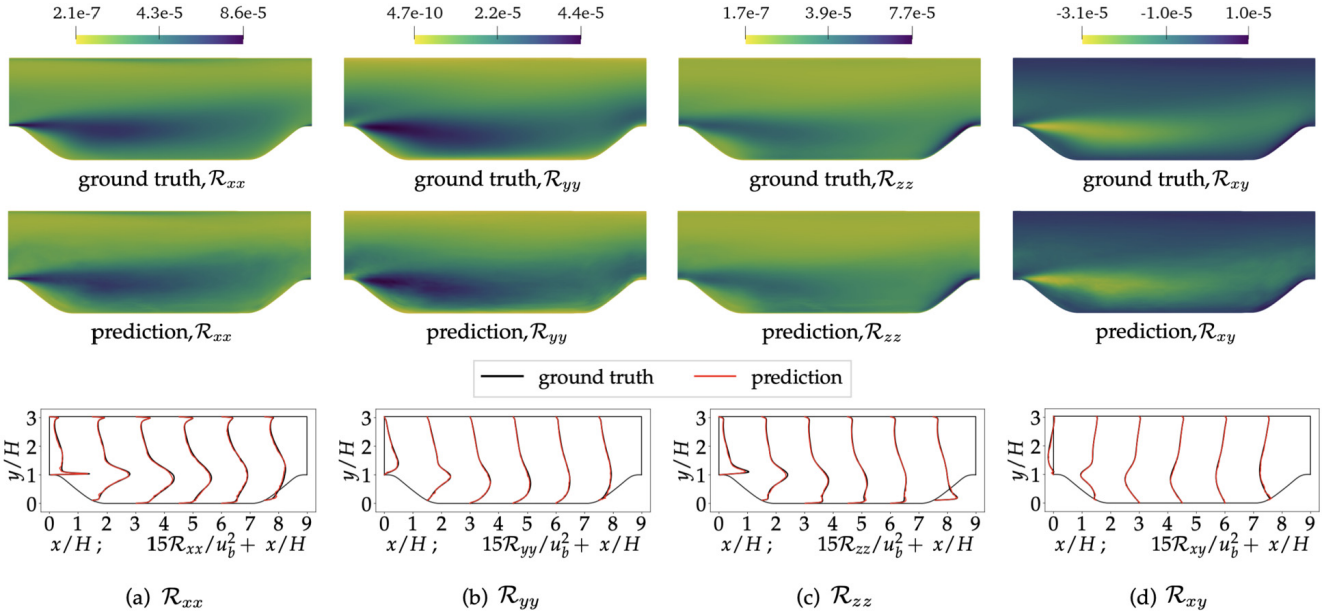


Fig. 10. Comparison of the ground truths (DNS) of four Reynolds stress components (top row) and the corresponding predictions provided by the trained neural network (middle row), along with their scaled profiles at six cross-sections (bottom row) for the testing configuration with the slope parameter $\alpha = 1.0$.

model with the nonlocal counterpart. In particular, the local model is trained with five scalar features $\boldsymbol{c} = [\mathrm{u}, s, b, d, p]^\top$, whereas features associated with neighboring points are removed. The training settings are kept the same to ensure a fair comparison. The prediction error from the local model (14.9%) is significantly larger than that from the nonlocal model (7.7%), indicating a greater deviation from the ground truth compared to the nonlocal model. Note that the calculation of the prediction error here follows Eq. (7) but targets at the total squared error of all the components of the Reynolds stress tensor instead of the TKE. The comparison of the predictions from the local and nonlocal models is shown in Fig. 11. Specifically, the profiles of four Reynolds stress components from the two models are compared at six cross-sections (top row), with the green/dashed box designating regions with clear differences. We can observe that the regions with differences are primarily found close to the hills, whereas in other areas the predictions from the two models are almost similar. The error plots (bottom two rows) illustrate this phenomenon more clearly, which show the pointwise relative error compared to the ground truth, with the darker area indicating a larger prediction error. For all Reynolds stress components, it is evident that the nonlocal model provides more accurate predictions than the local model near the inlet/outlet and in the recirculation zone. This is because the contraction and expansion of the flow channel induce rapid changes in strain and rotation, the history of which influences the Reynolds stress anisotropy downstream. Consequently, the Reynolds stress anisotropy in the windward and leeward sides near the hill cannot be determined by using only local flow features; the upstream flow field must also be considered. By incorporating the spatial nonlocality to the model, we can better account for the effects of flow history. As a result, the trained model is better equipped to capture the physics of the Reynolds stress transport than the local model and provides more accurate predictions. The VCNN's capability to learn the nonlocal effects was also demonstrated in our previous work for scalar transport PDEs [37]: The upwind region has larger learned weights than the downstream weights when the flow convection is nontrivial.

Furthermore, the capability of VCNN-e to handle different stencil sizes demonstrates its resolution adaptivity and that it can serve as a neural operator for developing nonlocal tensorial constitutive models, which is illustrated in Fig. 12. To be specific, the network is trained using a constant stencil size of 300 and tested with different stencil sizes ranging from 50, 100, 200, 300, and 400 up to a full stencil size that varies by location. Notably, we specify separate seed states in the code to perform different random sampling for different stencil sizes to ensure that the testing data are unique from the training data and that the testing data for larger stencil sizes do not contain those for smaller stencil sizes. The result indicates that the prediction errors gradually converge as the stencil size increases. Comparatively, the prediction from the stencil size of 50 is inferior to that from other stencil sizes because 50 randomly sampled points are insufficient to describe the nonlocal flow field in the cloud, resulting in a considerable loss of mean flow information. In contrast, the full stencil size achieves the best prediction performance since all of the data points inside the cloud are used to represent the nonlocal mean flow fields without sacrificing any flow information. The distribution of the full stencil size over the entire domain is visualized in Fig. 12(b) to highlight its variation at different locations, with the dark area indicating a small stencil size. The prediction errors for the intermediate stencil sizes, from 200 to 400, are fairly close to those for the full stencil size. The above result demonstrates that the data with varying stencil sizes (i.e., neural network input dimension) can share the same network parameters, and all of them provide Reynolds stress predictions close to the DNS data. In other words, the VCNN-e is capable of learning the operator that maps an arbitrarily discretized nonlocal mean flow field to the Reynolds stress.

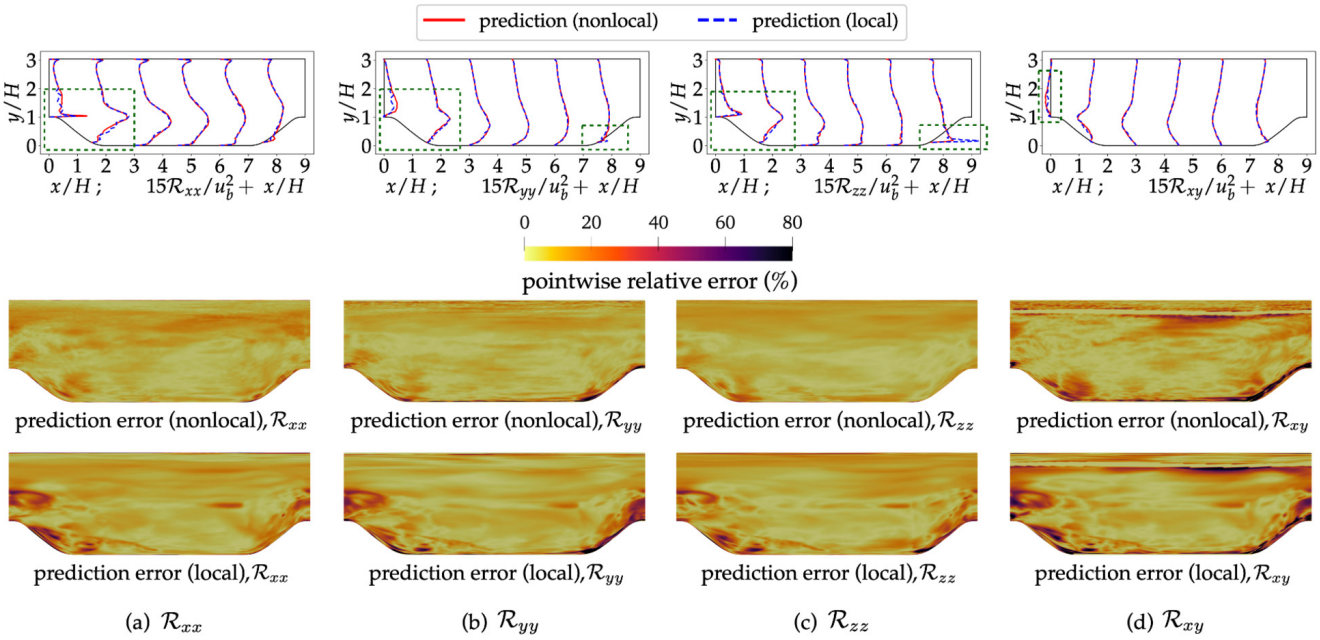


Fig. 11. Comparison of the predicted Reynolds stress components ($\mathcal{R}_{xx}$, $\mathcal{R}_{yy}$, $\mathcal{R}_{zz}$, and $\mathcal{R}_{xy}$) from the trained nonlocal and local models: Scaled profiles of the predicted Reynolds stress components from the nonlocal (solid/red) and local (dashed/blue) models are compared at six cross-sections (top row), with dashed boxes indicating locations with distinct differences; pointwise prediction errors of Reynolds stress components from the nonlocal (middle row) and local (bottom row) models, with the dark area representing a larger deviation from the ground truths.

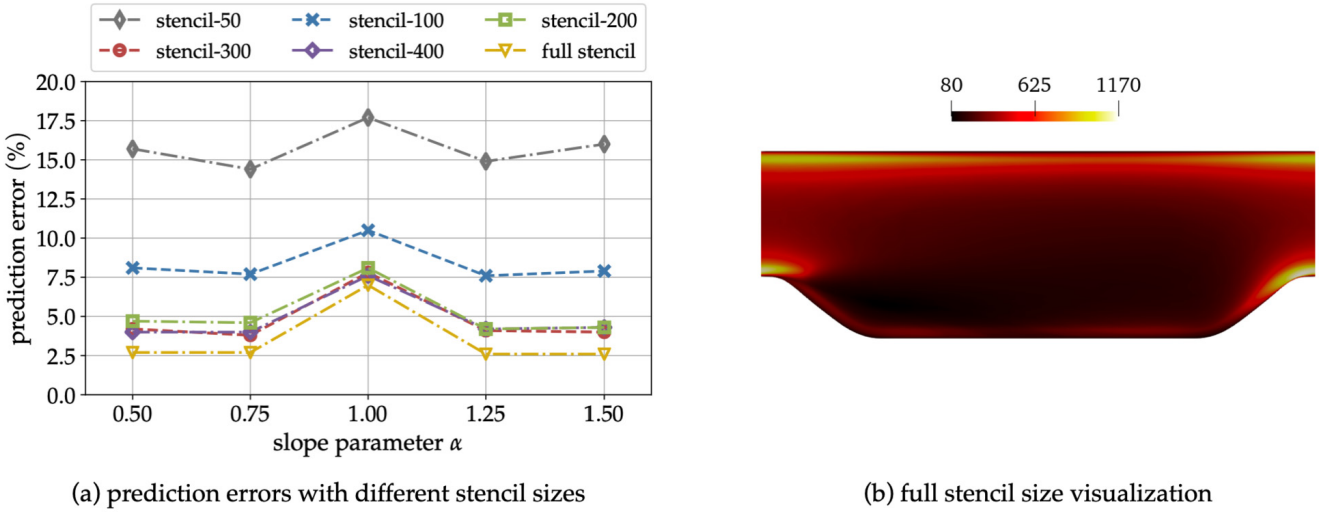


Fig. 12. Resolution adaptivity property of VCNN-e: (a) prediction errors for five periodic-hill flows with different stencil sizes and (b) Visualization of full stencil size at different locations in the testing flow with slope parameter $\alpha = 1.0$. Full stencil indicates that all the data points in the cloud are used for predicting the Reynolds stress at the central location, which differs from other testings with fixed stencil sizes (varying from 50 to 400) in that the stencil size varies from location to location.

5. Discussions

So far we have mainly demonstrated the capability of VCNN-e in modeling the full Reynolds stress. The same methodology can also be straightforwardly applied to model the normalized Reynolds stress anisotropy, which can be derived from the Reynolds stress transport equation (2) and implies a similar region-to-point mapping as the latter does. It is well known that turbulence modeling consists of determining the length scale, velocity scale, and anisotropy of the turbulence. The former two quantities amount to eddy viscosity ν_t, which cannot be determined by the local mean flow (i.e., the mean velocity or its gradient). Therefore, turbulent length- and velocity-scales must be solved for, i.e., via transport PDEs as in k–ε [51] and Spalart–Allmaras models [1]. These models account for the nonlocal dependence of eddy viscosity on the mean flow fields. However, they do not account for the dependency of the Reynolds stress *anisotropy* on the mean flow upstream. Rather, the normalized anisotropy is modeled based on local mean velocity gradient (or equivalently $\nabla\mathbf{u} = \boldsymbol{S} + \boldsymbol{\Omega}$, with $\boldsymbol{S}$ and $\boldsymbol{\Omega}$ denoting strain-rate and rotation-rate tensors, respectively). As such, eddy viscosity models (linear or nonlinear)

are referred to as *weak equilibrium* models [22]. Modeling the normalized Reynolds stress anisotropy through the proposed VCNN-e can complement the existing eddy viscosity models by providing a better description of the Reynolds stresses.

Additionally, it is important to compare new methods with existing methods and to highlight actual improvement and innovation. We note that one machine learning model potentially competitive for predicting Reynolds stress or other tasks in CFD is the graph kernel network (GKN) [33]. GKN is based on graph neural networks, a specialized class of neural networks designed to process graph structures and properties. By treating the mesh cells in the nonlocal region as graph nodes and constructing edges using pairwise properties, graph neural networks can effectively model the nonlocal constitutive relationships discussed in this paper. The GKN model proposed in [33] has permutational invariance built-in by consistently maintaining the graph structure and averaging across the associated nodes. The graph's construction allows the model to handle unstructured data, which is assumed in the current study and is very common in industrial CFD simulations. Note that GKN in itself does not have built-in rotational invariance, but one can equip GKN with rotational invariance by pre-processing the edge properties, i.e., by substituting the pair of neighboring three-dimensional vectors with their inner products. Therefore, it is valuable to compare VCNN to GKN, without and with rotational invariance. Our recent work [52] has performed a comprehensive comparison between VCNN and GKN, evaluating both the conceptual aspects, such as handling nonlocal mapping and preserving symmetry, and the numerical aspects, including the models' accuracy and efficiency. The findings can be summarized as follows.

1. GKN without rotational invariance performs poorly when predictions are made in rotated coordinates.
2. GKN equipped with rotational invariance has slightly better accuracy than VCNN.
3. The training time of GKN scales quadratically with the number of points in the cloud, i.e., $\mathcal{O}(n^2)$, where n is the number of points in the cloud. Such computation scaling makes it difficult for GKN to handle clouds with over 200 points. In contrast, VCNN has a constant training time (due to parallel processing of the GPU), i.e., $\mathcal{O}(1)$. It can handle clouds with at least a few thousand points.

The comparison between GKN and VCNN discussed above is performed on a scalar transport equation, but the conclusion for a tensor transport equation should be qualitatively similar.

We note that our emulation of the Reynolds stress transport model is currently limited to flows over periodic hills with a constant Reynolds number due to data availability. While the trained model may perform well for similar flows, such as flows over (curved) backward-facing steps with similar Reynolds numbers, as demonstrated in our previous work [37], its applicability to other flow scenarios requires further investigation. These scenarios include flows over different configurations, flows in boundary layers with favorable or adverse pressure gradients, flows with stagnation or strong rotation, or even flows with compressible effects. Scaling up the model to encompass a wider range of flows for joint training is a possible way to improve its generalizability. Also, the current training approach for VCNN-e involves using half of the available data pairs of $(\mathcal{Q}, \mathcal{R})$ across the entire flow domain, which could lead to high computational costs for both data storage and model training. However, we have used this approach merely as a proof-of-concept. It is possible that training the model with fewer data pairs from specific sub-domains may be sufficient because of the correlation between data points and the redundancy therein. Recent research has demonstrated that 3D turbulent flows can be accurately reconstructed by training a generative adversarial network (GAN)-based model using only 2D observation data from cross-sections [53]. This finding highlights the potential of deep learning-based methods to learn complex physical mappings, even with limited data.

6. Conclusion

In this work we generalize the vector-cloud neural network as a neural operator to model the constitutive tensor transport equations. By training on unstructured data points, the proposed neural operator can faithfully capture the underlying nonlocal physics through a region-to-point mapping; it is invariant to coordinate translation and ordering of the points and meanwhile equivariant to coordinate rotation. The demonstrated performance shows its promise for nonlocal constitutive models, especially the RANS momentum equation in turbulence modeling to solve for mean velocities and pressure.

There are a few directions worth exploring in future work. The immediate next step is to examine the performance of the learned nonlocal model as the closure model for the primary PDEs. The recent work [50] has demonstrated the robustness and stability of the neural operator-based eddy viscosity model when coupled with RANS equations. Similar works should be further done for the machine learning-based full turbulence model considering anisotropy as proposed in this paper. Furthermore, we will evaluate the proposed network on more complicated, three-dimensional flows. Finally, current input only used vectors; how to include tensor quantities (e.g., strain-rate and rotation-rate) of the mean flow in the input for nonlocal constitutive modeling is an interesting question to explore in future work.

CRediT authorship contribution statement

Jiequn Han: Conceptualization, Methodology, Writing - original draft preparation, Reviewing and editing.

Xu-Hui Zhou: Methodology, Code development, Verification and Validation, Writing - original draft preparation, Reviewing and editing.

Heng Xiao: Conceptualization, Methodology, Code development, Writing - original draft preparation, Reviewing and editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the link to my data/code in the paper.

Acknowledgements

X.-H. Zhou is supported by the U.S. Air Force under agreement number FA865019-2-2204. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The computational resources used for this project were provided by the Advanced Research Computing (ARC) of Virginia Tech, which is gratefully acknowledged.

Appendix A. Equivariance of the proposed vector-cloud neural network

We claim that the proposed neural operator has rotational equivariance. That is, given a function $\mathcal{R} = f(\mathbf{q})$ where $\mathbf{q}$ is the input of vector type and $\mathcal{R}$ is the output of tensor type, we say f is equivariant if the following equation holds for any rotation matrix Q:

$$\mathsf{Q}\mathcal{R}\mathsf{Q}^\top = f(\mathsf{Q}\mathbf{q}) \qquad \text{or more concisely} \qquad \mathcal{R}' = f(\mathbf{q}'),$$

in which the prime denotes vector or tensor under the rotated frame. Here we demonstrate that a function of the following form satisfies the equivalence condition above:

$$\mathcal{R} = \mathcal{X}^\top \mathcal{E} \mathcal{X} + \gamma \boldsymbol{I} \tag{A.1}$$

where $\boldsymbol{I}$ is a second order identify tensor, $\mathcal{X}$ are vectors defined in the same coordinate system as $\mathbf{q}$ and $\mathcal{R}$, and $\mathcal{E}$ is a function with tensor output that is *invariant* with the input vector $\mathbf{q}$, i.e., $\mathcal{E}(\mathbf{q}) = \mathcal{E}(\mathbf{q}')$. We first note the fact that the identity tensor is equivariant under rotation, i.e., $\mathsf{Q}\boldsymbol{I}\mathsf{Q}^\top = \boldsymbol{I}$. The equivariance can be shown straightforwardly as follows:

$$\mathsf{Q}\mathcal{R}\mathsf{Q}^\top = \mathsf{Q}\mathcal{X}^\top \mathcal{E} \mathcal{X}\mathsf{Q}^\top + \gamma \boldsymbol{I} = (\mathsf{Q}\mathcal{X})^\top \mathcal{E} (\mathsf{Q}\mathcal{X})^\top + \gamma \boldsymbol{I}. \tag{A.2}$$

Or more concisely:

$$\mathcal{R}' = \mathcal{X}' \mathcal{E} \mathcal{X}'^\top + \gamma \boldsymbol{I}.$$

That is, the function form proposed in Eq. (A.1) holds in the rotated frame for any rotation Q.

Appendix B. Interpretation and symmetries of proposed formulation

The considered formulation of Reynolds stress $\mathcal{R} = \mathcal{X}^\top \mathcal{E} \mathcal{X} + \gamma \boldsymbol{I}$ is apparently similar to the deviatoric–hydrostatic (anisotropic–isotropic) decomposition frequently used in turbulence modeling [41]:

$$\mathcal{R} \equiv \overline{u_i u_j} = 2k \left(\boldsymbol{b} + \frac{\boldsymbol{I}}{3} \right) = 2k \left(\boldsymbol{V} \Lambda \boldsymbol{V}^\top + \frac{\boldsymbol{I}}{3} \right) \tag{B.1}$$

where k is the turbulent kinetic energy (half the trace of $\mathcal{R}$), $\boldsymbol{b} = \mathrm{dev}(\mathcal{R})/2k$ is the normalized anisotropy, and Λ and $\boldsymbol{V}$ are eigenvalues and eigenvectors of tensor $\boldsymbol{b}$. However, it is worth noting that the two forms are fundamentally different. First, the considered decomposition is a mathematical construction to guarantee rotational equivariance. It is not motivated physically. Second, the mathematical structures of the two decompositions are also different. Most notably, the deviatoric tensor $\mathbf{b}$ has trace zero by construction, while $\mathcal{X}^\top \mathcal{E} \mathcal{X}$ has a nonzero trace. Also note that for flows aligned with the xy-plane (normal to z-axis), the xz-, yz-components of the part $\mathcal{X}^\top \mathcal{E} \mathcal{X}$ are zeros, while both components are nonzero in general for the normalized anisotropic tensor $\boldsymbol{b}$.

It can be seen from the derivation above that a formulation containing only the first part, i.e., $\mathcal{R} = \mathcal{X}^\top \mathcal{E} \mathcal{X}$, would be sufficient to guarantee equivariance of the formulation. Unfortunately, it fails to represent a correct function space for modeling Reynolds stresses in plane mean-strain conditions (i.e., statistically one- or two-dimensional mean flows). As such,

Table C.1
Detailed architectures of the embedding network and the fitting network. The architecture specifies the number of neurons in each layer in sequence, from the input layer to the hidden layers (if any) and the output layer. The numbers of neurons in the input and output layers are highlighted in bold. Note that the number of input neurons in the embedding network for the DNS scenario changes to $l' = 8$ with an additional scalar feature of mean pressure.

	Embedding network	Fitting network ($\mathcal{D} \mapsto \mathcal{E}, \gamma$)
No. of input neurons	$l' = 7$	$m \times m' = 256$ ($\mathcal{D} \in \mathbb{R}^{64\times 4}$)
No. of hidden layers	3	2
Architecture	(**7**, 32, 64, 64, **64**)	(**256**, 64, 64, **65**)
No. of output neurons	$m = 64$	65 ($\mathcal{E} \in \mathbb{R}^{64\times 64}$, $\gamma \in \mathbb{R}$)
Activation functions	ReLU, Linear (last layer)	ReLU, Linear (last layer)
No. of trainable parameters	10688	24833

the isotropic part $\gamma \boldsymbol{I}$ is added to the formulation to address the functional space mismatch problem, which is further detailed below.

Without loss of generality, consider a turbulent flow with the two-dimensional mean flow in the xy-plane (i.e., the z-direction is statistically homogeneous and thus has zero strain-rate). The z-component of the relative coordinates $\mathcal{X}$ is zero. In this case simple algebra shows that the tensor $\mathcal{X}^\top \mathcal{E} \mathcal{X}$ has the following form:

$$\begin{bmatrix} \mathcal{R}_{xx} & \mathcal{R}_{xy} & 0 \\ & \mathcal{R}_{yy} & 0 \\ \text{symm} & & 0 \end{bmatrix}$$

It is noteworthy that by construction the formulation correctly ensures that (1) the output is a symmetric tensor and (2) that the shear components $\mathcal{R}_{xz}$ and $\mathcal{R}_{yz}$ terms are zero in such two-dimensional mean flows. However, the normal stress component $\mathcal{R}_{zz}$ is also forced to zero, while in general this term is nonzero even in statistically two-dimensional flows (or even in the simplest isotropic homogeneous flows). For example, it has been shown by experiments that the ratios of normal stress components (i.e., turbulence intensities) in the streamwise (x-), wall-normal (y-), and spanwise (z-) directions are $\mathcal{R}_{xx} : \mathcal{R}_{yy} : \mathcal{R}_{zz} \approx 1 : 0.4 : 0.6$ in the log-law region of a zero-pressure gradient flat-late boundary layer [54].

One would be tempted to remedy the inadvertent singularity by generating the training data in a general plane (i.e., by rotating the coordinate frame so that the flow is not aligned with the xy-plane). However, this is not a valid workaround. Since the formulation is rotational equivariant by construction, no matter in which coordinate system the training is performed, the trained model will still predict $\mathcal{R}_{zz} = 0$ if the flow is aligned with the xy-plane, which violates the physics discussed above.

There is also another perspective to understand the inadequacy of the term $\mathcal{X}^\top \mathcal{E} \mathcal{X}$. When the z-component of the relative coordinate $\mathcal{X}$ is always zero, $\mathcal{X}$'s row space is at most two-dimensional. Note that the row space of $\mathcal{X}^\top \mathcal{E} \mathcal{X}$ is no larger than that of $\mathcal{X}$, i.e., it has a rank of no more than two. In contrast, the physical Reynolds stress has a full rank (i.e., of three) in general even for statistically one- or two-dimensional mean flows. Hence, the formulation $\mathcal{R} = \mathcal{X}^\top \mathcal{E} \mathcal{X}$ cannot adequately represent Reynolds stresses in one- or two-dimensional mean flows.

In view of such limitations of $\mathcal{X}^\top \mathcal{E} \mathcal{X}$, we propose to add a term that allows the $\mathcal{R}_{zz}$ component to be nonzero, while preserving the equivariance of the first part ($\mathcal{X}^\top \mathcal{E} \mathcal{X}$) as well as the existing symmetries of the output tensor (i.e., $\mathcal{R}_{yz} = \mathcal{R}_{yz} = 0$ and $\mathcal{R} = \mathcal{R}^\top$). The term $\gamma \boldsymbol{I}$ satisfies all these requirements. Note that here γ is a trainable scalar parameter as opposed to $2k/3$ in the anisotropic decomposition Eq. (B.1). Finally, because the formulation is equivariant, such a construction is a valid remedy regardless of the direction of the plane strain.

Appendix C. Network architectures and training parameters

Detailed architecture of VCNN-e, consisting of an embedding network and a fitting network, is provided in Table C.1. The embedding network operates identically on the scalar features $\boldsymbol{c} \in \mathbb{R}^{l'}$ associated with each point in the cloud and outputs a row of m elements in matrix $\mathcal{G}$. The fitting network maps the invariant feature matrix $\mathcal{D}$ to a diagonal matrix $\mathcal{E}$ and a single element γ.

The neural networks are trained using the Adam optimizer [55]. Both training processes take 5000 epochs, with batch sizes of 512 for synthetic data and 256 for DNS data, respectively. The learning rate is set to be 0.001 at the beginning of the training process.

Appendix D. Non-dimensionalization of the Reynolds stress transport equation

In the first scenario, the VCNN-e is utilized to emulate the Reynolds stress transport equation, which may disregard the kinematic viscosity for fully developed turbulent flows. This equation is expressed as

$$\mathbf{u} \cdot \nabla \mathcal{R} - \nabla \cdot (\nu_t \nabla \mathcal{R}) = \mathcal{P} + \Phi - E,$$

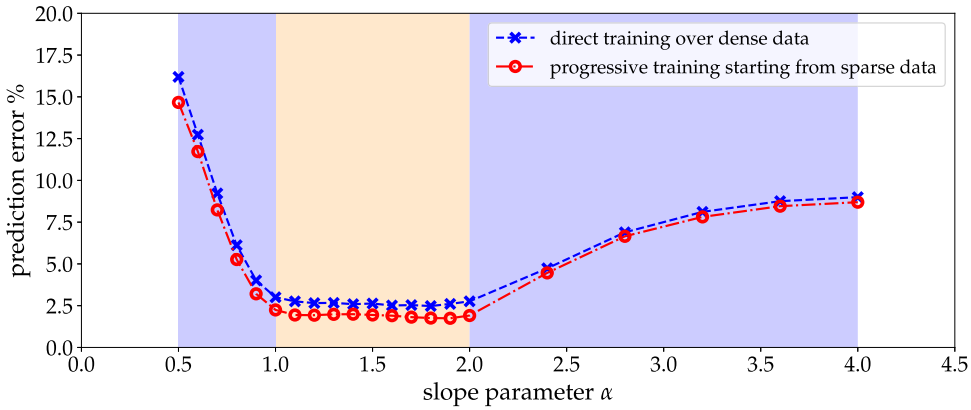


Fig. E.13. Comparison of direct and progressive training methods in terms of prediction performance for turbulence kinetic energy k at various slope parameters α between 0.5 and 4. The blue dashed and red dash-dotted lines represent the prediction errors of the trained models by direct and progressive training methods, respectively. The testings on 21 flows are based on all the data points within the cloud, and thus the comparison is fair.

where ν_t, $\mathcal{P}$, Φ, and E are modeled as functions of $\mathcal{R}$ and $\mathbf{u}$. Although the PDE is in a dimensional form, all inputs for the neural operator are dimensionless. As a result, the neural operator can be trained using flows from various length- and time-scales, as long as they are dynamically similar.

We use the hill height H and the bulk velocity u_b to normalize the variables as follows:

$$\widetilde{\mathbf{u}} = \frac{\mathbf{u}}{u_b}, \quad \widetilde{\mathcal{R}} = \frac{\mathcal{R}}{u_b^2}, \quad \widetilde{\ell_m} = \frac{\ell_m}{H}, \quad \widetilde{\nabla} = H\nabla,$$

where the dimensionless variables are indicated with the superscript. As such, we can rewrite the Reynolds stress transport equation in the following dimensionless form:

$$\widetilde{\mathbf{u}} \cdot \widetilde{\nabla}\widetilde{\mathcal{R}} - \widetilde{\nabla} \cdot \left(\widetilde{\nu_t}\widetilde{\nabla}\widetilde{\mathcal{R}}\right) = \widetilde{\mathcal{P}} + \widetilde{\Phi} - \widetilde{E}, \tag{D.1}$$

where

$$\widetilde{\mathcal{P}} = -\left[\widetilde{\mathcal{R}} \cdot \widetilde{\nabla}\widetilde{\mathbf{u}} + (\widetilde{\mathcal{R}} \cdot \widetilde{\nabla}\widetilde{\mathbf{u}})^\top\right], \tag{D.2}$$

$$\widetilde{\Phi} = \frac{C_1}{\widetilde{\tau}}\,\mathrm{dev}(\widetilde{\mathcal{R}}) + C_2\,\mathrm{dev}(\widetilde{\mathcal{P}}), \quad \text{with} \quad \widetilde{\tau} = \frac{\widetilde{k}}{\widetilde{\varepsilon}}, \tag{D.3}$$

$$\widetilde{E} = \frac{2}{3}\widetilde{\varepsilon}\boldsymbol{I} \quad \text{with} \quad \widetilde{\varepsilon} = C_D\frac{\widetilde{k}^{3/2}}{\widetilde{\ell_m}}, \tag{D.4}$$

$$\widetilde{\nu_t} = C_\mu\frac{\widetilde{k}^2}{\widetilde{\varepsilon}}. \tag{D.5}$$

From these equations, we can see the solutions of Reynolds stress transport equation at different scales are dynamically similar. Note that the characteristic length H and the characteristic velocity u_b must scale at the same ratio to achieve the same Reynolds number for dynamic similarity, because the production term depends on the velocity gradient.

Appendix E. Progressive training of neural operator

In this work, we have explored two training methods: (1) direct training that always uses dense data, with each cloud having $n = 300$ points, and (2) progressive training first with sparse data ($n = 25$ points in the cloud) and then dense data. In the first method, we directly train the model with 5000 epochs. In the second method, we first train the model using the sparse data for 1000 epochs and then using the dense data for 4000 epochs. Under the same number of epochs, the second strategy reduces training time by approximately 18% compared to the first method, with an average of 11.12 and 13.56 seconds per epoch, respectively. The prediction performances of the trained models under two different methods are compared over 21 different configurations, which is shown in Fig. E.13. We can see that, with less training time, the progressive method performs better in all testing cases. This is because the first stage of training using sparse data (i.e., a subset of full training data) accelerates the convergence of the neural operator. Note that for progressive training, the number of training epochs with sparse data is not fine-tuned. We select 1000 epochs as an example to demonstrate how progressive training improves the training efficiency. More settings can be investigated to optimize the progressive training.

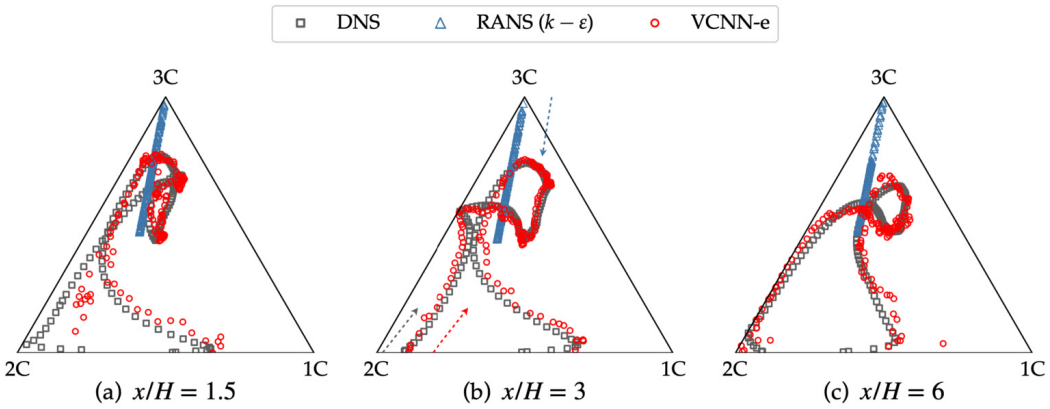


Fig. F.14. Predicted Reynolds stress anisotropy for testing configuration with slope parameter $\alpha = 1.0$. Prediction from the nonlocal model (VCNN-e) is compared with the ground truth (DNS) and $k-\varepsilon$ RANS result at three different cross-sections. The Reynolds stress anisotropy componentality is represented by its position in the Barycentric: one-component state (1C), two-component isotropic state (2C), and three-component isotropic state (3C), as well as a combination thereof (interior). The dashed arrows indicate the direction from the bottom wall to the top wall.

Appendix F. Predictions of anisotropy states

We examine the anisotropy states of the predicted Reynolds stress in Section 4, which exhibit a high degree of similarity with the original DNS data as well. The anisotropy state is represented by the position within the Barycentric triangle shown in Fig. F.14. The three vertices, 1C, 2C, and 3C, of the Barycentric triangle represent, in order, the one-component state, the two-component isotropic state, and the three-component isotropic state. The interior position represents a combination thereof. The coordinates of a location in the Barycentric triangle are determined by the eigenvalues of the normalized Reynolds stress anisotropy $\boldsymbol{b}$. More details on the calculation of coordinates in the Barycentric triangle can be found in the paper [39].

As illustrated in Fig. F.14, the positions of the predicted anisotropy states within the Barycentric triangle are comparable to those of the DNS data along three cross-sections ($x/H = 1.5, 3, 6$). Such consistency demonstrates that the VCNN-e is capable of providing a reasonable anisotropy prediction for an unseen configuration despite the fact that the anisotropy is not explicitly included in the loss function. At cross-section $x/H = 1.5$, the near-wall region close to the hill displays the most apparent discrepancy. This is because the region lies within the recirculation zone, where the different hill slope causes the mean flow field to differ from that in the training cases. Moreover, the positions of predicted turbulence states from VCNN-e show better componentality than the RANS result. The positions of near-wall turbulence states from the standard $k-\varepsilon$ RANS model are observed near the three-component isotropic state (vertex 3C), whereas both predicted and actual positions are indicated to be close to the two-component state (bottom edge).

References

[1] P.R. Spalart, S.R. Allmaras, A one-equation turbulence model for aerodynamic flows, AIAA J. 94 (1992).
[2] B. Launder, B. Sharma, Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc, Lett. Heat Mass Transf. 1 (2) (1974) 131–137.
[3] B.E. Launder, G.J. Reece, W. Rodi, Progress in the development of a Reynolds-stress turbulence closure, J. Fluid Mech. 68 (3) (1975) 537–566.
[4] B.D. Coleman, M.E. Gurtin, Thermodynamics with internal state variables, J. Chem. Phys. 47 (2) (1967) 597–613.
[5] J. Han, C. Ma, Z. Ma, W. E, Uniformly accurate machine learning-based hydrodynamic models for kinetic equations, Proc. Natl. Acad. Sci. 116 (44) (2019) 21983–21991.
[6] H. Lei, L. Wu, W. E, Machine-learning-based non-newtonian fluid model with molecular fidelity, Phys. Rev. E 102 (4) (2020) 043309.
[7] W. E, J. Han, L. Zhang, Machine-learning-assisted modeling, Phys. Today 74 (7) (2021) 36–41.
[8] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. 807 (2016) 155–166.
[9] J.-X. Wang, J.-L. Wu, H. Xiao, Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data, Phys. Rev. Fluids 2 (3) (2017) 034603.
[10] J.-L. Wu, H. Xiao, E.G. Paterson, Physics-informed machine learning approach for augmenting turbulence models: a comprehensive framework, Phys. Rev. Fluids 3 (2018) 074602.
[11] M. Schmelzer, R.P. Dwight, P. Cinnella, Discovery of algebraic Reynolds-stress models using sparse symbolic regression, Flow Turbul. Combust. 104 (2) (2020) 579–603.
[12] F.E. Bock, R.C. Aydin, C.J. Cyron, N. Huber, S.R. Kalidindi, B. Klusemann, A review of the application of machine learning and data mining approaches in continuum materials mechanics, Front. Mater. 6 (2019) 110.
[13] D.Z. Huang, K. Xu, C. Farhat, E. Darve, Learning constitutive relations from indirect observations using deep neural networks, J. Comput. Phys. (2020) 109491.
[14] K. Xu, D.Z. Huang, E. Darve, Learning constitutive relations using symmetric positive definite neural networks, J. Comput. Phys. 428 (2021) 110072.
[15] F. Masi, I. Stefanou, P. Vannucci, V. Maffi-Berthier, Thermodynamics-based artificial neural networks for constitutive modeling, J. Mech. Phys. Solids 147 (2021) 104277.
[16] J.B. Scoggins, J. Han, M. Massot, Machine learning moment closures for accurate and efficient simulation of polydisperse evaporating sprays, in: AIAA Scitech 2021 Forum, 2021, p. 1786.

[17] X.-L. Zhang, H. Xiao, X. Luo, G. He, Ensemble Kalman method for learning turbulence models from indirect observation data, J. Fluid Mech. 949 (2022) A26, https://doi.org/10.1017/jfm.2022.744.
[18] J.-L. Wu, R. Sun, S. Laizet, H. Xiao, Representation of stress tensor perturbations with application in machine-learning-assisted turbulence modeling, Comput. Methods Appl. Mech. Eng. 346 (2019) 707–726.
[19] M.I. Zafar, H. Xiao, M.M. Choudhari, F. Li, C.-L. Chang, P. Paredes, B. Venkatachari, Convolutional neural network for transition modeling based on linear stability theory, Phys. Rev. Fluids 5 (2020) 113903.
[20] S. Taghizadeh, F.D. Witherden, S.S. Girimaji, Turbulence closure modeling with data-driven techniques: physical compatibility and consistency considerations, New J. Phys. 22 (9) (2020) 093023.
[21] R. Wang, R. Walters, R. Yu, Incorporating symmetry into deep dynamics models for improved generalization, in: International Conference on Learning Representations, 2021.
[22] S.B. Pope, A more general effective-viscosity hypothesis, J. Fluid Mech. 72 (2) (1975) 331–340.
[23] T. Gatski, C. Speziale, On explicit algebraic stress models for complex turbulent flows, J. Fluid Mech. 254 (1993) 59–79.
[24] C.G. Speziale, S. Sarkar, T.B. Gatski, Modelling the pressure–strain correlation of turbulence: an invariant dynamical systems approach, J. Fluid Mech. 227 (1991) 245–272.
[25] T.B. Gatski, M.Y. Hussaini, J.L. Lumley, Simulation and Modeling of Turbulent Flows, Oxford University Press, 1996.
[26] L. Gao, Y. Du, H. Li, G. Lin, RotEqNet: rotation-equivariant network for fluid systems with symmetric high-order tensors, J. Comput. Phys. 461 (2022) 111205.
[27] Z. Long, Y. Lu, X. Ma, B. Dong, PDE-Net: learning PDEs from data, in: International Conference on Machine Learning, PMLR, 2018, pp. 3208–3216.
[28] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Eng. 361 (2020) 112732.
[29] B. Kim, V.C. Azevedo, N. Thuerey, T. Kim, M. Gross, B. Solenthaler, DeepFluids: a generative network for parameterized fluid simulations, Comput. Graph. Forum (Proc. Eurograph.) 38 (2) (2019).
[30] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (3) (2021) 218–229.
[31] C. Ma, B. Zhu, X.-Q. Xu, W. Wang, Machine learning surrogate models for Landau fluid closure, Phys. Plasmas 27 (4) (2020) 042502.
[32] M.D. Ribeiro, A. Rehman, S. Ahmed, A. Dengel, DeepCFD: efficient steady-state laminar flow approximation with deep convolutional neural networks, preprint, arXiv:2004.08826, 2020.
[33] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: graph kernel network for partial differential equations, preprint, arXiv:2003.03485, 2020.
[34] Z. Li, N.B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, in: International Conference on Learning Representations, 2021.
[35] J. Han, L. Zhang, R. Car, W. E, Deep Potential: a general representation of a many-body potential energy surface, Commun. Comput. Phys. 23 (3) (2018) 629–639.
[36] L. Zhang, J. Han, H. Wang, W. Saidi, R. Car, W. E, End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems, in: Advances in Neural Information Processing Systems, 2018, pp. 4436–4446.
[37] X.-H. Zhou, J. Han, H. Xiao, Frame-independent vector-cloud neural network for nonlocal constitutive modeling on arbitrary grids, Comput. Methods Appl. Mech. Eng. 388 (2022) 114211.
[38] G.M. Sommers, M.F.C. Andrade, L. Zhang, H. Wang, R. Car, Raman spectrum and polarizability of liquid water from deep neural networks, Phys. Chem. Chem. Phys. 22 (19) (2020) 10592–10602.
[39] H. Xiao, J.-L. Wu, S. Laizet, L. Duan, Flows over periodic hills of parameterized geometries: a dataset for data-driven turbulence modeling from direct simulations, Comput. Fluids 200 (2020) 104431.
[40] B. Launder, G.J. Reece, W. Rodi, Progress in the development of a Reynolds-stress turbulence closure, J. Fluid Mech. 68 (03) (1975) 537–566.
[41] S.B. Pope, Turbulent Flows, Cambridge University Press, Cambridge, 2000.
[42] The OpenFOAM Foundation, OpenFOAM User Guide, https://cfd.direct/openfoam/user-guide, 2020.
[43] R.I. Issa, Solution of the implicitly discretised fluid flow equations by operator-splitting, J. Comput. Phys. 62 (1986) 40–65.
[44] X.-H. Zhou, J. Han, H. Xiao, Learning nonlocal constitutive models with neural networks, Comput. Methods Appl. Mech. Eng. 384 (2021) 113927.
[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: an imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019, pp. 8026–8037.
[46] X.-H. Zhou, J. Han, H. Xiao, Vector-cloud neural network with equivariance (VCNN-e) for tensors, https://github.com/xuhuizhou-vt/VCNN-nonlocal-constitutive-model/tree/master/tensor-transport.
[47] J.L. Lumley, G.R. Newman, The return to isotropy of homogeneous turbulence, J. Fluid Mech. 82 (1) (1977) 161–178.
[48] H. Xiao, S. Laizet, Periodic hills of parameterized geometries DNS database, https://github.com/xiaoh/para-database-for-PIML/tree/master/pehill-29-cases-DNS.
[49] S. Laizet, N. Li, Incompact3d: a powerful tool to tackle turbulence problems with up to O(10^5) computational cores, Int. J. Numer. Methods Fluids 67 (11) (2011) 1735–1757.
[50] R. Xu, X.-H. Zhou, J. Han, R.P. Dwight, H. Xiao, A PDE-free, neural network-based eddy viscosity model coupled with RANS equations, Int. J. Heat Fluid Flow 98 (2022) 109051.
[51] B. Launder, B. Sharma, Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc, Lett. Heat Mass Transf. 2 (1974) 131–137.
[52] M.I. Zafar, J. Han, X.-H. Zhou, H. Xiao, Frame invariance and scalability of neural operators for partial differential equations, Commun. Comput. Phys. 32 (2) (2022) 336–363.
[53] M.Z. Yousif, L. Yu, S. Hoyas, R. Vinuesa, H. Lim, A deep-learning approach for reconstructing 3D turbulent flows from 2D observation data, Sci. Rep. 13 (1) (2023) 2529.
[54] P.S. Klebanoff, Characteristics of turbulence in a boundary layer with zero pressure gradient, Tech. Rep., National Bureau of Standards, Gaithersburg, MD, 1955.
[55] D. Kingma, J. Ba, Adam: a method for stochastic optimization, in: Proceedings of the International Conference on Learning Representations, 2015.

Part II

Field-to-Field Mapping: Neural Operator for Accelerating RANS Simulations with Enhanced Initial Conditions

Chapter 4

Neural operator-based super-fidelity: A warm-start approach for accelerating steady-state simulations

Neural operator-based super-fidelity: A warm-start approach for accelerating steady-state simulations

Xu-Hui Zhou[a], Jiequn Han[b], Muhammad I. Zafar[a], Eric M. Wolf[c], Christopher R. Schrock[d], Christopher J. Roy[a], Heng Xiao[e]

[a]*Kevin T. Crofton Department of Aerospace and Ocean Engineering, Virginia Tech, Blacksburg, 24060, VA, USA*
[b]*Center for Computational Mathematics, Flatiron Institute, New York, 10010, NY, USA*
[c]*Ohio Aerospace Institute, Wright-Patterson Air Force Base, Dayton, 45433, OH, USA*
[d]*Air Force Research Laboratory, Wright-Patterson Air Force Base, Dayton, 45433, OH, USA*
[e]*Stuttgart Center for Simulation Science, University of Stuttgart, Stuttgart, 70569, BW, Germany*

Abstract

In recent years, using neural networks for the accelerated solving of partial differential equations (PDEs) has become a research hotspot in both academic and industrial settings. However, relying solely on trained neural networks as surrogate models can lead to a trust crisis, as the accuracy of predicted solutions is significantly influenced by data volume and quality, as well as training algorithms. This concern is particularly pronounced in scientific tasks that prioritize computational precision and deterministic outcomes. In this study, we introduce a novel approach that utilizes neural networks as a warm-start (initialization) tool, which we refer to as "super-fidelity", to accelerate the solution of steady-state PDEs while ensuring result accuracy. The concept of super-fidelity, inspired by super-resolution in computer vision, involves mapping solutions from low-fidelity computational models to targeted high-fidelity ones using a vector-cloud neural network with equivariance (VCNN-e). This network preserves all desired invariance and equivariance properties for scalar and vector solutions, respectively, and adapts seamlessly to varying spatial resolutions. We evaluate this approach in two typical scientific computing scenarios: one with weak nonlinearity, exemplified by flows around elliptical cylinders at low Reynolds numbers with a monotonic convergence pattern; and the other with strong nonlinearity, exemplified by flows over airfoils at high Reynolds numbers with an oscillatory convergence pattern. In both scenarios, the neural operator-based initialization method achieves at least a two-fold acceleration in convergence while maintaining the same level of accuracy, as compared to commonly used initialization methods based on uniform fields or potential flows. Its robustness is further demonstrated across various iterative algorithms with different linear equation solvers. Additionally, we investigate time savings across multiple simulations, even accounting for model development time, and propose a practical strategy for efficient training data generation. In summary, our approach promises an effective solution to accelerate steady-state PDE solutions with neural networks while upholding result accuracy, particularly in scientific applications where precision is paramount.

*Corresponding author.

Keywords: neural network/operator, partial differential equations, acceleration, super-fidelity, warm-start

1. Introduction

Partial differential equations (PDEs) serve as a fundamental and ubiquitous tool to describe how various properties change continuously in space and time. Their significance extends to numerous fields, including physics, engineering, finance, and beyond. While the development of numerical methods has enabled efficient solutions for certain types of PDEs, there remains a subset of PDEs for which traditional techniques are computationally intensive, thus limiting their applicability in tackling complex problems. Recent years have witnessed a new promising paradigm in scientific computing, as neural networks have emerged as promising tools for PDE solutions. Neural networks, with the capacity to approximate complex functions, can deliver near-ground-truth predictions quickly once trained well. The latest development of neural operators has further extended this capability to infinite-dimensional function spaces, enabling trained models to adapt to inputs with arbitrary resolutions/discretizations; see, e.g., [1–6]. This paradigm shift in computational science has brought us more insights and new possibilities for solving PDEs.

However, relying on neural network predictions as the ultimate solution to PDEs often triggers a trust crisis, especially for tasks that prioritize computational precision and outcome determinism. This trust issue exists in both two prevalent neural network-based approaches for PDE solving, including data-driven methods such as supervised learning of solution operators, and physics-driven methods such as physics-informed neural networks (PINNs). In the case of data-driven methods, training an accurate and reliable network model is highly challenging, which significantly depends on data volume and quality, as well as training algorithms. While studies have shown that well-trained models can produce results with desired accuracy in certain cases [7, 8], the inherent black-box nature of neural networks and the absence of physical constraints (e.g., conservation laws) during training render these predictions impractical for researchers who rely on first principles in their work. On the other hand, PINNs aim to solve PDEs by training neural networks to satisfy physical constraints [9]. However, comparative studies between PINNs and traditional solvers indicate that PINNs do not exhibit a clear advantage either in terms of accuracy or computational speed for various PDEs in low-dimensional cases [10]. Hence, for scientific computing tasks that require solution precision, traditional solvers should continue to occupy a central role. It is essential to mitigate the risk of misusing neural network predictions and to ensure the accuracy of the outcomes, at least at the current stage.

In this paper, we propose an approach that employs neural network predictions as initial conditions to accelerate the solution of steady-state PDEs. We have chosen steady-state fluid flow problems to demonstrate the efficacy of our approach, considering their industrial relevance and the inherent complexity associated with computational models based on the Navier–Stokes (N–S) equations. For practitioners in computational

fluid dynamics (CFD) who typically employ uniform fields or solutions from lower-fidelity models as initial conditions (see "traditional workflow" in Fig. 1, including the (a), (c), and (d) panels), trained neural networks can deliver better initial conditions, thereby speeding up the solving process. This is particularly beneficial to problems involving multiple simulations under varying conditions, a common need in addressing optimal design or control problems in diverse engineering applications [11]. For researchers devoted to accelerating PDE solving through machine learning (see "surrogate model" in Fig. 1, including the (a), (b), and (d) panels), integrating traditional solvers as the final touch ensures the accuracy of results. Specifically, our proposed solution strategy involves the development of an approximate mapping from the solution of a low-fidelity model (i.e., Laplace's equation for potential flow) to the solution of a high-fidelity model (i.e., the target PDEs). This mapping, which is straightforward and fast to evaluate, provides a solution that is much closer to the final solution than the low-fidelity model. By utilizing the output of this mapping as an improved initial guess, we can effectively warm-start the iterative solving process, further reducing the total computational time. We have referred to this task as "super-fidelity", drawing inspiration from the super-resolution concept in image processing, which enhances a low-resolution image to achieve higher resolution [12]. The objective of super-fidelity is to bridge the gap between low and high-fidelity models, thereby enhancing the efficiency and stability of steady-state simulations in complex flow investigations.

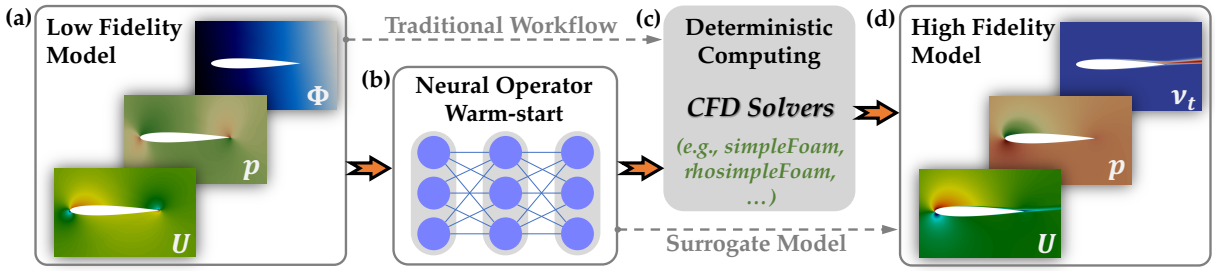


Figure 1: Schematic workflow of accelerating steady-state fluid flow simulations with the neural operator warm-start approach. The neural operator is trained to map solutions from a low-fidelity model to a high-fidelity model and then provides improved initial conditions for CFD simulations. This approach transcends traditional workflows by delivering notably enhanced initial conditions to reduce the computational time. It also bolsters the reliability of employing neural networks for scientific problems by performing the deterministic computing afterwards, in contrast to using neural networks purely as surrogate models.

In the context of warm-starting existing iterative solvers for high-fidelity models, our focus lies in constructing a mapping from low-fidelity solutions to high-fidelity ones – the core component of our super-fidelity approach. This mapping, mathematically, is also referred to as an operator between two function spaces. The field of operator learning, which has received significant research interest in recent years, offers such a way forward. This approach aims to develop a data-driven approximation of mappings/operators between function spaces to provide efficient evaluations of the target output after an initial training phase on available

data. In particular, neural networks with specialized architectures, also known as neural operators, have shown considerable promise in approximating such mappings. In this work, we utilize our recently developed neural operator, the vector-cloud neural network with equivariance (VCNN-e) [6, 13], to establish the desired mapping in super-fidelity. The VCNN-e guarantees the physical symmetry in flow simulations, allowing the operator to be applied in different coordinate systems and discretizations seamlessly. Furthermore, it considers the nonlocal property of flow simulations, enabling learning to be done with just a few solutions by decomposing the whole solution field into many patches for efficient learning. The entire workflow of our super-fidelity approach includes all four panels shown in Fig. 1. The second panel, referred to as the warm-start, involves the training of a super-fidelity model based on neural operators. Our numerical experiments on two distinct types of steady-state simulations, one with weak nonlinearity and the other with strong nonlinearity, illustrate the effectiveness of the super-fidelity approach. In both cases, the warm-start models achieve acceleration ratios of no less than two folds without any compromise on accuracy.

The remainder of this paper is organized as follows. Section 2 introduces the computational models adopted in this study, including the governing equations for both high-fidelity and low-fidelity models, and describes the neural operator architecture. In Section 3, we present the neural operator predictions and investigate the resulting acceleration effects for numerical solvers in two testing cases. In Section 4, we introduce an efficient strategy for data generation and explore potential application scenarios to achieve a "real" acceleration by reducing overall computation time, which includes both data generation and training phases. We also discuss the difficulties and limitations of developing reliable neural network surrogates for PDEs. Section 5 provides a summary of this work and some concluding remarks.

2. Problem Statement and Methodology

2.1. Physical model

In this study, we assume the high-fidelity computational models for steady-state flow fields adhere to the following conservation law, presented in a general and compact form.

$$\nabla \cdot \mathcal{F}(\boldsymbol{U}) - \mathcal{S} = 0, \tag{1}$$

where $\boldsymbol{U}$ is the conserved variable vector, $\mathcal{F}$ and $\mathcal{S}$ denote the flux vector and the source vector, respectively. Taking N–S equations as an example, if we neglect the influence of body forces and radiative heating (i.e., $\mathcal{S} = 0$), the notations are defined as follows,

$$\boldsymbol{U} = \begin{bmatrix} \rho \\ \rho \boldsymbol{u} \\ \rho e_t \end{bmatrix}, \qquad \mathcal{F}(\boldsymbol{U}) = \begin{bmatrix} \rho \boldsymbol{u}^\top \\ \rho \boldsymbol{u} \otimes \boldsymbol{u} + p\boldsymbol{I} - \boldsymbol{\tau} \\ (\rho e_t + p)\boldsymbol{u}^\top - \boldsymbol{u}^\top \boldsymbol{\tau} + \boldsymbol{q}^\top \end{bmatrix}, \tag{2}$$

where ρ, $\boldsymbol{u}$, p, $\boldsymbol{q}$ and e_t denote the fluid's density, velocity, pressure, heat flux, and total energy per unit mass, respectively; $\boldsymbol{\tau}$ is the viscous stress tensor and $\boldsymbol{I}$ is the identity tensor. The system of equations has to be completed by a thermodynamic equation of state (EOS) in a general form of $p = p(\rho, e)$, where the internal energy per unit mass $e = e_t - \frac{1}{2}|\boldsymbol{u}|^2$.

Note that Eq. (2) varies according to specific fluid flow simulation scenarios. For instance, in the case of an incompressible laminar flow, where density is assumed to be constant, the energy equation involving the third components in $\boldsymbol{U}$ and $\mathcal{F}$ can often be decoupled from the mass and momentum equations, and can be further neglected if the flow is isothermal. Consequently, the problem simplifies, and only velocity and pressure need to be solved.

The low-fidelity model considered in this work is based on the potential flow theory, which represents idealized fluid motion. Specifically, the fluid is assumed to be inviscid and incompressible, while the flow itself is considered irrotational. The steady-state potential flow is governed by Laplace's equation [14]

$$\nabla^2 \Phi = 0, \quad \text{with } \boldsymbol{u} = \nabla \Phi, \tag{3}$$

where Φ is the velocity potential. Despite its simplifications and limitations in modeling real fluid behavior, potential flow provides rapid simulations for studying various fluid flow phenomena, typically within seconds. This makes it an attractive initialization option in steady-state fluid flow simulations, with the aim of improving stability and convergence speed, as seen in simulation software such as Ansys [15] and OpenFOAM [16]. In view of the physical information contained in, and the cost-effectiveness to obtain, the potential flow field, it is natural to use potential flow as input for the super-fidelity model. Therefore, the super-fidelity task is to train a neural operator to map the solution of Eq. (3) to that of Eq. (1).

2.2. Equivarient neural operator

The neural operator employed in the super-fidelity task is developed based on the VCNN-e [6] to satisfy all the symmetries and function space requirements for predicting both scalar and vector fields. The neural operator forms a region-to-point mapping, $\mathcal{Q} \mapsto S(\boldsymbol{x}_0)$, where $S(\boldsymbol{x}_0)$ is the solution of a high-fidelity model at a point of interest $\boldsymbol{x}_0$, and $\mathcal{Q}$ indicates the collection of potential flow (low-fidelity model) features $\{\mathbf{q}_i\}_{i=1}^n$ on n points sampled from the computational domain. The feature vector $\mathbf{q}$ attached to each point is chosen to include the relative coordinate $\boldsymbol{x}' = \boldsymbol{x} - \boldsymbol{x}_0$, potential flow velocity $\boldsymbol{u}^L$, and additional six scalar quantities $\boldsymbol{c}^L = [\mathsf{u}^L, p^L, \Phi, \theta, \eta, r]$, with the superscript L denoting the low-fidelity model. In scalar quantities, u and p represent velocity magnitude and pressure; θ and η denote cell volume and wall distance; and r is proximity (inverse of relative distance) to the point of interest. As such, the feature vector $\mathbf{q}$ is $2 + 2 + 6 = 10$ dimensional for the two-dimensional flows considered in this work, and we define the input matrix as $\mathcal{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_n]^\top \in \mathbb{R}^{n \times 10}$. The output of the neural operator includes the solutions of velocity $\boldsymbol{u}^H$ and other scalar quantities $\boldsymbol{c}^H$ (e.g., pressure) of a high-fidelity model, i.e., $S(\boldsymbol{x}_0) = [\boldsymbol{u}^H(\boldsymbol{x}_0), \boldsymbol{c}^H(\boldsymbol{x}_0)]$.

Note that all input and output of the neural operator are non-dimensionalized using characteristic scales and normalized.

The neural operator VCNN-e consists of three modules at a high level: (a) an embedding module, (b) a fitting module, and (c) a rotating module, as illustrated in Fig. 2. The first module (Fig. 2(a)) intends to find an invariant representation, denoted as $\mathcal{D}$, for the input cloud of potential flow, which has translational, rotational, and permutational invariance. Specifically, the translational invariance is ensured by the use of relative coordinates rather than absolute ones; the rotational invariance is ensured by the inner product between feature vectors via the operation $\mathcal{Q}\mathcal{Q}^\top$; and the permutational invariance and resolution adaptivity is guaranteed by the average over all the points via the operations $\frac{1}{n}\mathcal{G}^\top\mathcal{Q}$ and $\frac{1}{n}\mathcal{Q}^\top\mathcal{G}'$. The second module utilizes $\mathcal{D}$, referred to as the invariant feature matrix, as the input to fit both an invariant vector $\mathcal{E}$ and predictions of scalar quantities $\mathbf{c}^H$ for the high-fidelity model through a fitting neural network. Finally, the third module (Fig. 2(c)) incorporates the embedded coordinates $\tilde{\mathcal{X}} = \frac{1}{n}\mathcal{G}^\top\mathcal{X} \in \mathbb{R}^{m\times 2}$ from the first module where $\mathcal{X} = [\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_n]^\top \in \mathbb{R}^{n\times 2}$ and rotates properly to get the velocity prediction through $\boldsymbol{u}^H = \tilde{\mathcal{X}}^\top\mathcal{E}$. Since $\mathcal{X}$ rotates in the same way as the original coordinates under rotation, the predicted velocity vector preserves the desired equivariance. Note that the output of the neural operator consists of $\boldsymbol{u}^H$ from the rotating module and $\boldsymbol{c}^H$ from the fitting module, both of which serve as the initial conditions for simulating the high-fidelity model. The neural operator architecture is detailed in Appendix A. More details regarding the design of VCNN-e and the proof of symmetries can be found in the literature [6, 13].

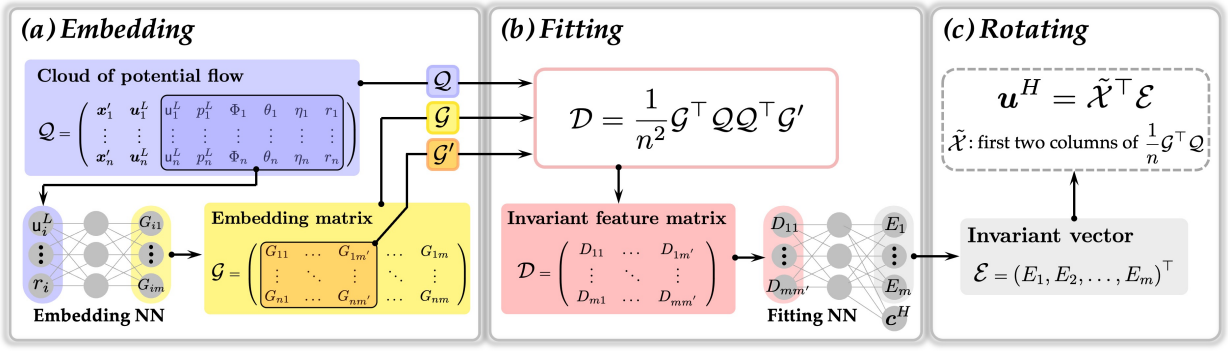
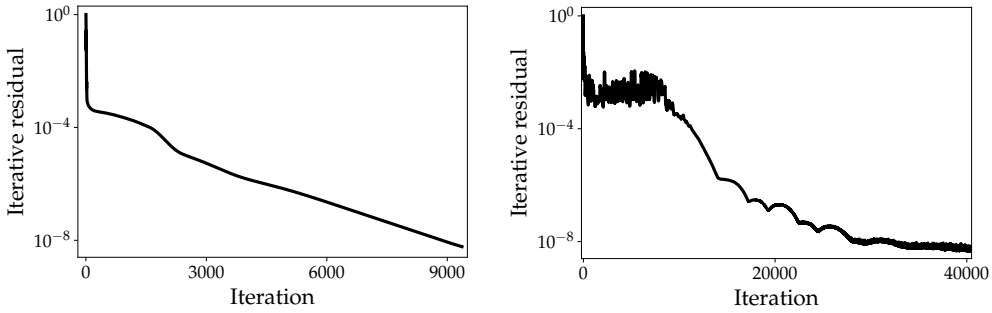


Figure 2: Detailed architecture of the neural operator VCNN-e for mapping solution of a low-fidelity model (potential flow) to that of a high-fidelity model: (a) embed the frame-independent features $\{\boldsymbol{c}_i\}_{i=1}^n$ to form the embedding matrix $\mathcal{G} \in \mathbb{R}^{n\times m}$; (b) project the pairwise inner-product matrix $\mathcal{Q}\mathcal{Q}^\top$ to the learned embedding matrix $\mathcal{G}$ and its submatrix $\mathcal{G}'$ to yield an invariant feature matrix $\mathcal{D} \in \mathbb{R}^{m\times m'}$; flatten and feed the feature matrix $\mathcal{D}$ into the fitting network to predict the invariant vector $\mathcal{E}$ and scalar quantities $\boldsymbol{c}^H$; (c) rotate $\mathcal{E}$ through the embedded coordinates $\tilde{\mathcal{X}}$ and get the prediction of velocity $\boldsymbol{u}^H$. The final output consists of $\boldsymbol{c}^H$ in (b), which is invariant to the frame rotation, and $\boldsymbol{u}^H$ in (c), which is equivariant to the frame rotation, while both of them are invariant to the frame translation and the ordering of points in the cloud.

3. Result

In this section, we present a comprehensive evaluation of our warm-start approach by applying it to two different flow scenarios. The first scenario focuses on incompressible laminar flows over parameterized elliptical cylinders, while the second scenario involves compressible turbulent flows ($Re = 6 \times 10^6$) over various airfoils at different angles of attack. The convergence behaviors in these two flow scenarios align with the two typical patterns commonly observed in scientific computing [17, 18], as illustrated in Fig. 3. The first scenario demonstrates weaker nonlinearity with low Reynolds numbers, leading to a monotonic decrease in the residual. In contrast, the second scenario displays much stronger nonlinearity due to the high Reynolds number, causing the residual to oscillate while still trending downward. The neural operator is trained with just a few flow cases in both scenarios and the predictions on testing flows are used for initializing the steady-state simulations. The results demonstrate the efficacy of our approach, achieving acceleration ratios of at least two-fold while preserving the same level of accuracy for most testing flow simulations. On the other hand, if we take the prediction of the neural operator as the solution to PDEs directly, although the pointwise error is relatively small on average, the integral quantities (often of much more interest in engineering, such as drag and lift coefficients) may deviate more significantly from the values obtained from an accurate solution.



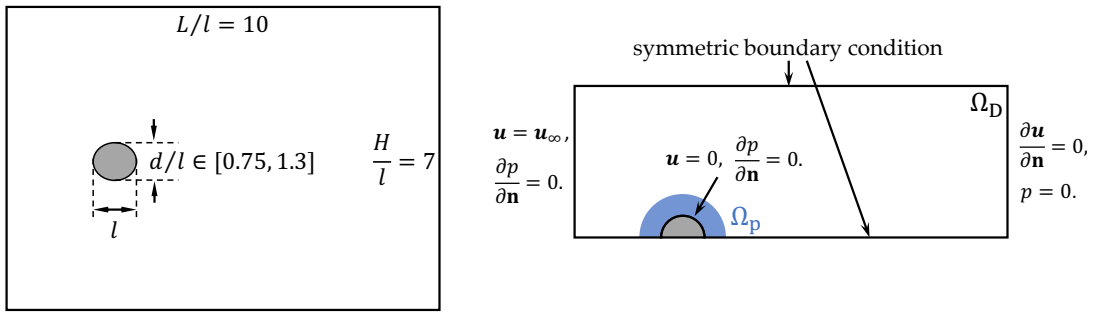
(a) Monotonic convergence (b) Oscillatory convergence

Figure 3: Two typical convergence processes in scientific computing: (a) monotonic convergence and (b) oscillatory convergence.

3.1. Incompressible laminar flow

Our first scenario considers incompressible laminar flows over elliptical cylinders at different Reynolds numbers, which is illustrated in Fig. 4(a). The Reynolds number is defined as $Re = \mathsf{u}_\infty d/\nu$, where u_∞ and ν denote the freestream velocity magnitude and kinematic viscosity, respectively. We adjust the non-dimensionalized cylinder thickness, denoted as d/l, within the range of 0.75 to 1.3 to control the Reynolds number, maintaining it between 30 and 52. Specifically, we generate 12 flows with uniformly distributed

Reynolds numbers, and divide them into two groups: six training flows with $Re = [30, 34, \ldots, 50]$, five interpolated testing flows with $Re = [32, 36, \ldots, 48]$, and one extrapolated testing flow with $Re = 52$. Here,



(a) Flow setup (b) Computation setup

Figure 4: Setup for the incompressible laminar flows over parameterized elliptical cylinders: (a) flow setup, where the ratio of the thickness d to length l of the elliptical cylinder is adjusted between 0.75 and 1.3 to generate a family of laminar flows at different Reynolds numbers, and (b) computation setup showing the computational domain Ω_{D} with the boundary conditions. Note that Ω_{p} refers to a region near the cylinder, where the distance from any point within this region to the cylinder wall is less than half of the cylinder's length l.

by assuming ρ to be constant and the Stokes's stress constitutive equation $\boldsymbol{\tau} = \nu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^{\top})$, the governing equations (1) and (2) are reduced to the following steady-state N–S equations:

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0, \\ \mathbf{u} \cdot \nabla \mathbf{u} - \nu \nabla^2 \mathbf{u} + \frac{1}{\rho} \nabla p &= 0, \end{aligned} \tag{4}$$

where velocity $\boldsymbol{u}$ and pressure p are to be solved. Therefore, the objective of super-fidelity is to learn the mapping from potential flows to steady-state incompressible laminar flows, i.e., from the solution of Eq. (3) to Eq. (4). The numerical simulations of potential and laminar flows are performed with the open-source CFD platform OpenFOAM [16], employing the solvers `potentialFoam` and `simpleFoam`, respectively. We conduct the simulations on half of the full domain by implementing symmetric boundary condition on the bottom boundary in Fig. 4(b). This setting eliminates any potential unsteadiness that may arise in full-domain simulations of laminar flows. In the simulation of laminar flows, Eq. (4) is iteratively solved until the residuals satisfy the convergence criterion, i.e., $\mathcal{R} < 10^{-8}$. In this study, residuals are defined in an absolute manner, as detailed in Appendix B.

During the training phase, we do not utilize the potential flow field across the entire computational domain Ω_{D} as input for training efficiency. Instead, the input consists of two parts: (1) the potential flow within the near-cylinder region Ω_{p}, which is represented by the feature vectors attached to about 400 points sampled in this region, and (2) the potential flow at a target point being predicted. The desired output is the velocity

and pressure of the laminar flow at the target point within Ω_D. Since Ω_D contains approximately 7000 cell points, we have about 7000 input-output data pairs for each training or testing flow. The neural operator is trained to minimize the misfit, using mean squared error (squared L_2 norm), between its predictions and the iteratively converged laminar flow fields. The training process is performed on an open-source machine learning framework PyTorch [19], which takes about half an hour on a single V100 graphics processing unit (GPU). More information regarding the training settings can be found in Appendix A.

The trained model exhibits good prediction performance on the testing flows, as shown in Fig. 5. The left column presents the predictions for the testing flow at $Re = 48$, which closely resembles the iteratively converged solutions displayed in the middle column. This prediction capability is further demonstrated by the small prediction errors in both solution and drag coefficient across all six testing flows, as provided in Table 1. In this work, the ground truths used to calculate prediction errors for solution and integral quantities refer to the solution achieved through iterative convergence at $\mathcal{R} = 10^{-8}$ and its corresponding integral quantities. However, it is important to note that while the model shows promise in prediction, the predicted solutions remain unreliable for direct application. Noticeable non-smoothness in the predictions is evident in the pointwise relative error plots in the right column of Fig. 5. The relative errors, particularly those near the cylinder surface and other boundaries, may lead to violations of the boundary conditions. Moreover, the non-smoothness in predictions and non-compliance with boundary conditions becomes more apparent during the convergence process of drag coefficients when integrating the predictions into the solver for iterative computation, as shown in Fig. 6. Here, the iterative errors of drag coefficients can reach up to 200% within the initial few iterative steps when using predicted solutions as initial conditions. Consequently, traditional solvers are still necessary for achieving more accurate and reliable solutions.

Table 1: Prediction errors for six testing flows around elliptical cylinders, including five interpolated flows and one extrapolated flow. The relative L_2 norm errors for solution (velocity and pressure in Ω_D) and the relative errors for drag coefficient are presented.

	Interpolation					**Extrapolation**
Prediction error (%)	$Re = 32$	$Re = 36$	$Re = 40$	$Re = 44$	$Re = 48$	$Re = 52$
Solution	0.82	0.81	0.89	0.96	1.0	1.2
Drag coefficient, C_D	3.6	4.8	3.6	1.0	2.7	6.9

Using neural networks for initial conditions significantly accelerates iterative convergence, which is illustrated in Fig. 7. Specifically, we compare the iterative convergence processes using three different initial conditions: uniform field, potential flow, and neural network prediction. We conduct this comparison for both interpolated and extrapolated testing flows. It is evident from the first two panels of Fig. 7 that using

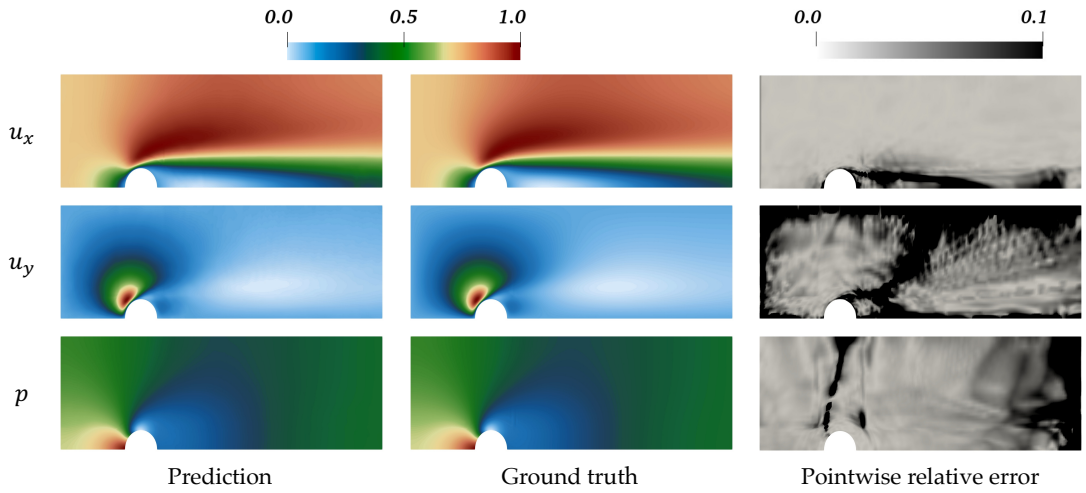


Figure 5: Comparison of the prediction (left column) of the solution and the corresponding ground truth (middle column), along with the pointwise relative error plots (right column) for the testing flow over an elliptical cylinder at $Re = 48$. The ground truth refers to the iteratively converged solution of velocity, u_x and u_y, and pressure, p, at a residual of 10^{-8}. The ground truth and prediction are non-dimensionalizaed and scaled between 0 and 1, while the pointwise relative errors are scaled between 0 and 0.1 for clarity.

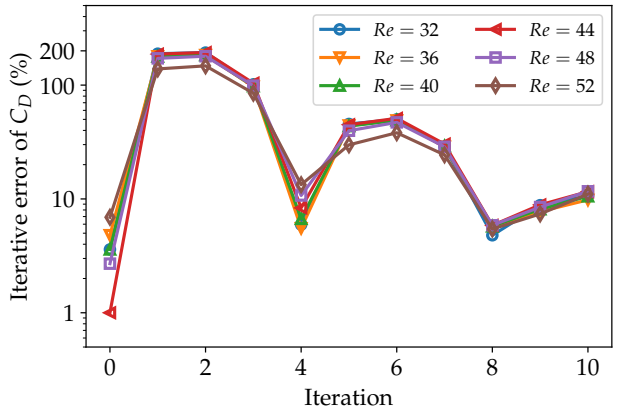


Figure 6: Iterative errors of drag coefficients in six testing flows over elliptical cylinders after using neural network predictions as initial conditions.

neural network predictions for initialization establishes an early lead in the convergence process in both testing cases and maintains this advantage consistently. As a result, it significantly reduces the number of iterations required to achieve convergence, thereby accelerating the iterative solution without loss of accuracy. For better showcasing this acceleration capability, we employ another refined initial condition for reference. Specifically, this refined initial condition is achieved by early stopping the simulation at a residual of 10^{-5} using the uniform field for initialization. We can observe that the convergence processes based on the neural network prediction and the refined field are quite close, with the former being slightly slower.

In other words, the neural network prediction in this scenario accelerate the solution process by effectively skipping the initial convergence phase, enabling the residual to approach approximately 10^{-5}, with only marginal adjustments in the subsequent convergence rate.

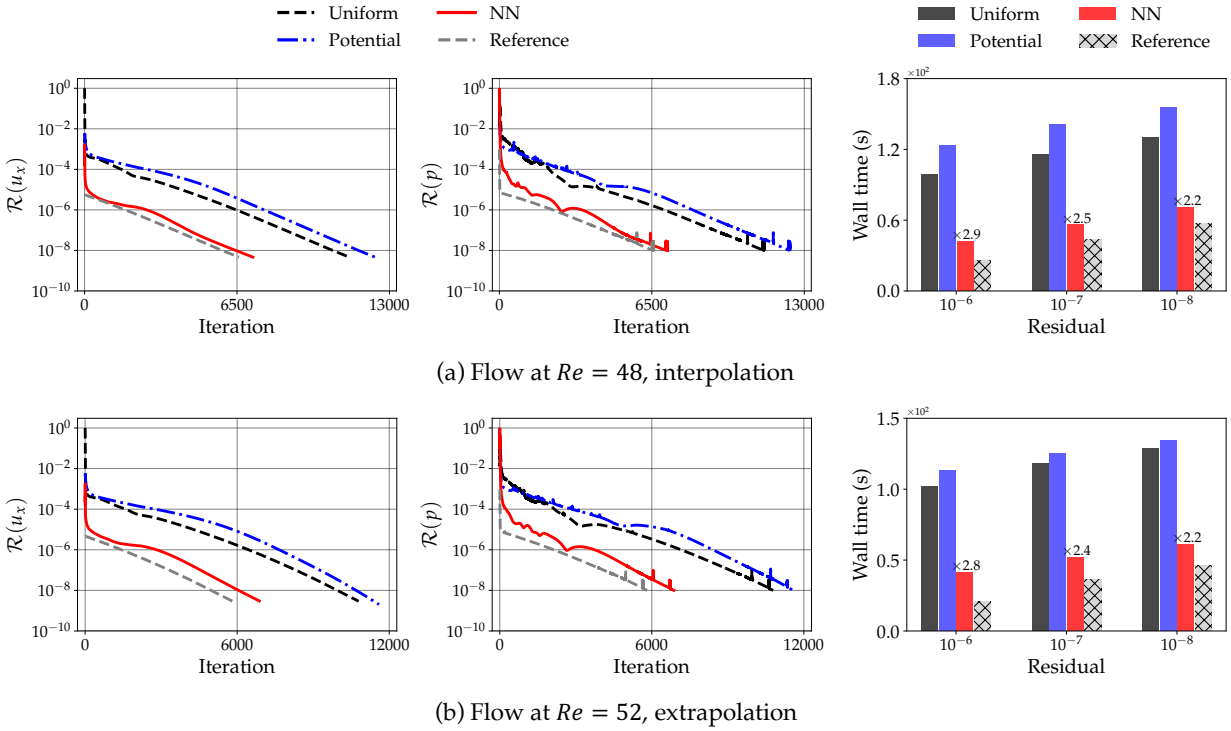


Figure 7: Comparison of three different initialization methods: uniforms fields, potential flow, and neural network prediction for both (a) interpolated and (b) extrapolated testing flows. Left two columns: the convergence processes of velocity and pressure; right column: the wall time required to satisfy three different convergence criteria (10^{-6}, 10^{-7}, and 10^{-8}). The reference here serves as an ideal initial condition at a residual of 10^{-5}, offering a more comprehensive assessment of the acceleration performance of neural network predictions.

This acceleration is more evidently reflected in the computational time, as is illustrated in the third panel of Fig. 7. Specifically, we establish three different convergence criteria for residuals at 10^{-6}, 10^{-7}, and 10^{-8}, and then compare the wall time associated with four different initial conditions. Note that the wall time covers the entire computational duration, including both pre-processing phase and iterative computation from initial state to final convergence. For potential flow initialization, the pre-processing time involves the time to solve Eq. (3); for neural network initialization, it involves solving Eq. (3) and the additional time needed for neural network prediction; for the reference case, we assume there is no pre-processing time. In each testing flow, neural network initialization significantly reduces the wall time needed to achieve any of the convergence criteria. To quantitatively evaluate this acceleration effect, we have introduced an acceleration ratio in this study, which is calculated as the wall time required when using potential flow

initialization divided by that when using neural network initialization. For the least demanding convergence criterion (10^{-6}), neural network initialization achieves an almost threefold acceleration, and for the strictest convergence criterion (10^{-8}), the acceleration is also more than twofold. While the reference shows further acceleration, it represents a more ideal scenario with a better trained network, which requires significantly larger training data volume and more training time.

3.2. Turbulent flow

Our second scenario considers turbulent flows over different airfoils at different angles of attack. The incoming Mach number remains constant at 0.15, and the Reynolds number based on chord length is $Re_c = 6 \times 10^6$. We select four airfoils (NACA0008, NACA0010, NACA0012, and NACA0015) and four angles of attack (9°, 10°, 12°, and 13°), denoted as α, to generate 16 flows for training and testing. The testing flows are determined using Latin hypercube sampling [20] to ensure the representativity of testing flows, leading to 4 testing flows and 12 training flows. We also add the flow over NACA0011 at $\alpha = 11°$ for testing, where neither the airfoil nor the angle of attack are included in the training data. Overall, the five testing flows can be categorized into three types based on their relationship with the training flows: interpolation, weak interpolation, and extrapolation, as detailed in Table. 2. Here, by introducing a density-weighted time average decomposition of $\boldsymbol{u}$ and e_t, and a standard time average decomposition of ρ and p, the governing equations (1) and (2) give the following equations, which is also known as the steady-state Farve-averaged Navier–Stokes equations [14].

$$\begin{aligned}
\frac{\partial}{\partial x_j}(\bar{\rho}\hat{u}_j) &= 0, \\
\frac{\partial}{\partial x_j}(\hat{u}_j\bar{\rho}\hat{u}_i) &= -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial \overline{\sigma}_{ij}}{\partial x_j} + \frac{\partial \tau_{ij}}{\partial x_j}, \\
\frac{\partial}{\partial x_j}[\hat{u}_j(\bar{\rho}\hat{e}_t + \bar{p})] &= \frac{\partial}{\partial x_j}(\overline{\sigma}_{ij}\hat{u}_i + \overline{\sigma_{ij}u_i''}) - \frac{\partial}{\partial x_j}\left(\bar{q}_j + c_p\overline{\rho u_j'' T''} - \hat{u}_i\tau_{ij} + \frac{1}{2}\overline{\rho u_i'' u_i'' u_j''}\right).
\end{aligned} \tag{5}$$

The standard time (Reynolds) averages are denoted by an overbar, $\bar{\phi}$, while the density-weighted time (Favre) averages are denoted by a hat, $\hat{\phi} = \overline{\rho\phi}/\bar{\phi}$. Fluctuations around Reynolds and Favre averages are denoted by single and double primes, $\phi' = \phi - \bar{\phi}$ and $\phi'' = \phi - \hat{\phi}$, respectively. In Eq. (5), the viscous stress $\bar{\sigma}_{ij}$, which corresponds to $\boldsymbol{\tau}$ in Eq. (2), is modeled as $\bar{\sigma}_{ij} = 2\hat{\mu}(\hat{S}_{ij} - \frac{1}{3}\frac{\partial \hat{u}_k}{\partial x_k}\delta_{ij})$, while the Reynolds stress τ_{ij} ($\equiv -\overline{\rho u_i'' u_j''}$), stemming from velocity fluctuations, is modeled as $\tau_{ij} = 2\hat{\mu}_t(\hat{S}_{ij} - \frac{1}{3}\frac{\partial \hat{u}_k}{\partial x_k}\delta_{ij}) - \frac{2}{3}\bar{\rho}k\delta_{ij}$, where $\hat{S}_{ij}$ is the mean velocity strain, k ($\equiv \frac{\widehat{u_i'' u_i''}}{2}$) is the turbulent energy, $\hat{\mu}$ and $\hat{\mu}_t$ denote molecular and turbulent viscosity, respectively. Here, $\hat{\mu}_t$ is modeled through the Spalart-Allmaras turbulence model [21]. The heat flux $\bar{q}_j$ and the turbulent heat flux $c_p\overline{\rho u_j'' T''}$ are modeled as $-\frac{c_p\hat{\mu}}{Pr}\frac{\partial \hat{T}}{\partial x_j}$ and $-\frac{c_p\hat{\mu}_t}{Pr_t}\frac{\partial \hat{T}}{\partial x_j}$, respectively, where c_p is the heat capacity at constant pressure, $\hat{T}$ is the temperature, Pr and Pr_t are laminar and turbulent Prandtl numbers, respective, with constant values of 0.72 and 0.85. The molecular diffusion term $\overline{\sigma_{ij}u_i''}$ and the turbulent transport term $-\frac{1}{2}\overline{\rho u_i'' u_i'' u_j''}$ are neglected considering their relatively small influences. To close

Eq. (5), the EOS is specified as

$$\bar{p}(\bar{\rho}, \hat{e}) = (\gamma - 1)\bar{\rho}\hat{e}, \quad \text{with } \hat{e} = \hat{e}_t - \frac{1}{2}\hat{u}_i\hat{u}_i - k, \tag{6}$$

where the heat capacity ratio γ is taken as constant at 1.4, and $\hat{e}$ is related to $\hat{T}$ by $\hat{e} = \frac{1}{\gamma}c_p\hat{T}$. By incorporating Eq. (6), we solve Eq. (5) to obtain the velocity, pressure and internal energy per unit mass for the flows over airfoils.

Table 2: Training and testing cases in flows over airfoils. The check mark (✓) denotes a training flow while the cross mark (✗) represents a testing flow. The subscripts "i", "wi", and "e" refer to three testing types: interpolation, weak interpolation, and extrapolation.

	NACA0008	NACA0010	NACA0011	NACA0012	NACA0015
$\alpha = 9°$	✓	✓		$\times_{\text{wi}}$	✓
$\alpha = 10°$	✓	✓		✓	$\times_{\text{wi}}$
$\alpha = 11°$			$\times_{\text{i}}$		
$\alpha = 12°$	✓	$\times_{\text{i}}$		✓	✓
$\alpha = 13°$	$\times_{\text{e}}$	✓		✓	✓

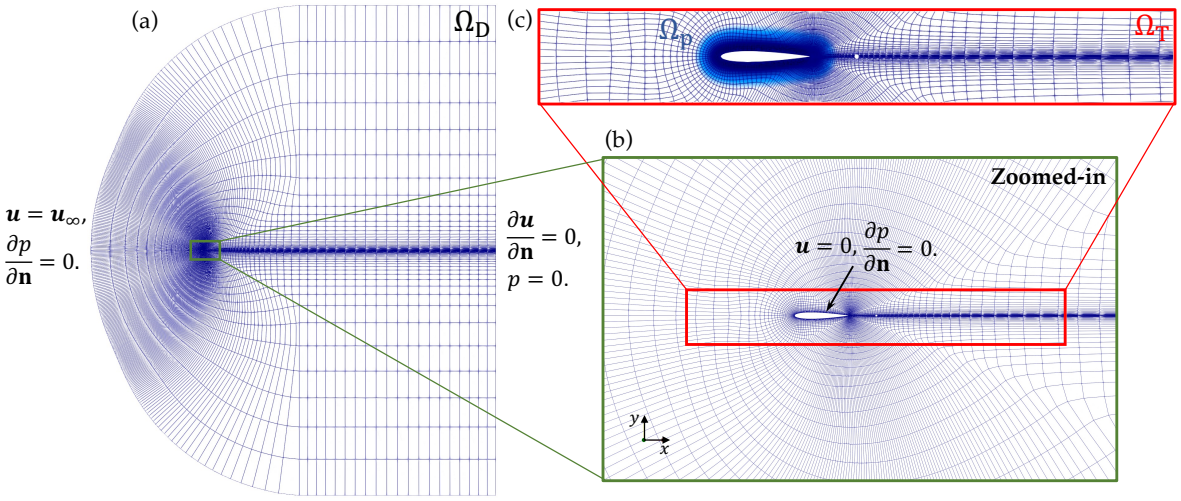


Figure 8: Computation setup for turbulent flows over airfoils: (a) the computational domain Ω_D with boundary conditions for the inlet and outlet, (b) a zoomed-in region around airfoil with boundary conditions for the airfoil surface, and (c) the training region Ω_T, where steady-state turbulent flow fields are predicted, and a near-airfoil region Ω_p used for representing potential flows. Note that Ω_p refers to a region where the distance from any point within this region to the airfoil surface is less than 0.3 times the chord length of the airfoil.

The goal of super-fidelity here is to learn the mapping from potential flows to steady-state turbulent flows, i.e., from the solution of Eq. (3) to Eq. (5). The numerical simulations of turbulent flows are performed using the steady-state solver `rhoSimpleFoam` in OpenFOAM, with the simulation domain Ω_D and the boundary

conditions illustrated in Fig. 8. Note that despite the incoming Mach number being within the range for subsonic flows, we use the compressible solver in consideration of a potentially high local Mach number, as suggested by information from the NASA website [22]. For data generation, Eq. (5) is iteratively solved until the residuals satisfy the convergence criterion, i.e., $\mathcal{R} < 10^{-8}$.

During the training phase, we concentrate on predicting the turbulent flows in a specific region around the airfoils (see Ω_{T} in Fig. 8), while approximating the flows in the remaining region with incoming flows since the effect of airfoils out of Ω_{T} is relatively small. Similar to the representation approach in the first scenario, the potential flow used as input does not encompass the entire domain. Specifically, the input includes two parts: (1) the potential flow within the near-airfoil region Ω_{p}, which is represented by the feature vectors attached to about 4000 points sampled in this region, and (2) the potential flow at a target point being predicted. The desired output is the velocity, pressure, and internal energy per unit mass at the target point within the training region Ω_{T}. Since Ω_{T} consists of approximately 14000 cell points, we have 14000 input-output data pairs for each training or testing flow. The training process on 12 flows is performed using PyTorch, which takes about 6.5 hours on a single V100 GPU. More information regarding the training settings can be found in Appendix A.

The trained model demonstrates good predictive performance even in this much more complex scenario. In Fig. 9, we present a comparison between the neural operator predictions and the iteratively converged solutions for the testing flow over the NACA0011 airfoil at $\alpha = 11^{\circ}$. The predicted velocity, pressure, and temperature fields, which have been scaled between 0 and 1, exhibit a remarkable degree of visual similarity with their corresponding ground truth counterparts, as illustrated in the first two columns. This agreement is further demonstrated by the testing errors provided in Table 3. Notably, the prediction errors of solutions in Ω_{T} for all five testing flows remain consistently below 6%. However, these predicted results have limited immediate applicability. Firstly, the presence of relatively high relative errors in specific regions, as shown in the third column of Fig. 9, can lead to non-smoothness and non-conservation for subsequent analyses. Secondly, while the prediction errors for the solutions are reasonably small, the integral quantities such as drag and lift coefficients may deviate more significantly from the iteratively converged values, as demonstrated in Table 3. In weak interpolation flows, despite the solution errors of approximately 3%, the relative errors for drag coefficient exceed 10%, which may not meet the precision standards in application scenarios.

To further enhance the solution precision, we employ the neural network predictions as initial conditions for solving Eq. (5) numerically. This approach exhibits a more robust convergence process and a notably faster convergence rate when compared to conventional initialization methods. In Fig. 10, we present a comparative analysis of convergence performance using three distinct initial conditions: uniform field, potential flow, and neural network prediction, across three different testing flow scenarios. In both interpolation and

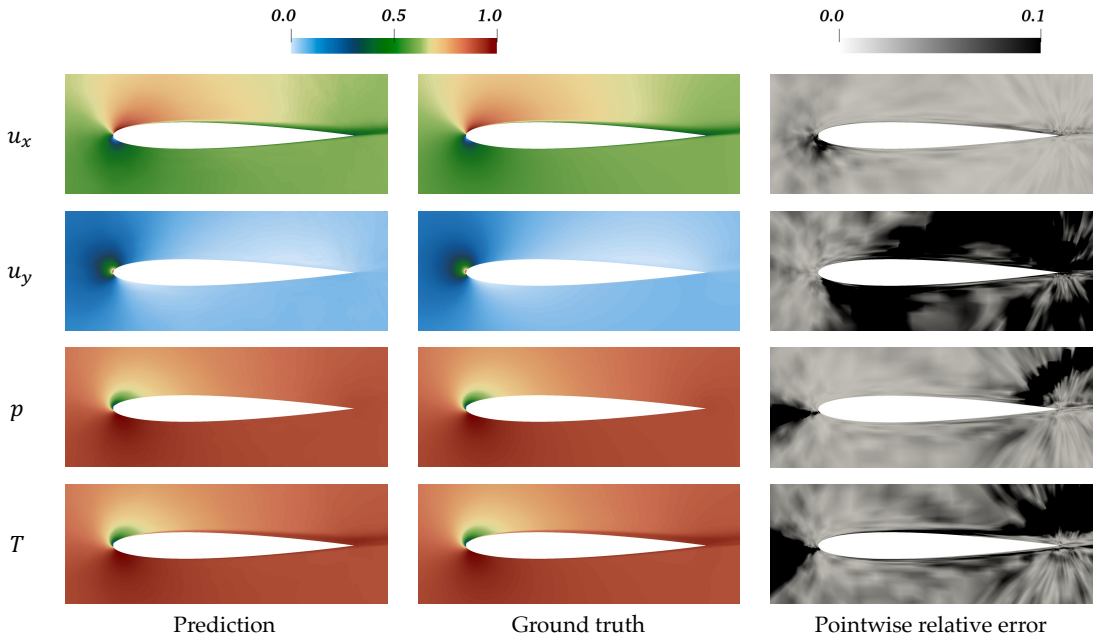


Figure 9: Comparison of the prediction (left column) of the solution and the corresponding ground truth (middle column), along with the pointwise relative error plots (right column) for the testing flow over NACA0011 at $\alpha = 11°$. The ground truth refers to the iteratively converged solution at a residual of 10^{-8}. The ground truth and prediction are non-dimensionalizaed and scaled between 0 and 1, while the pointwise relative errors are scaled between 0 and 0.1 for clarity.

Table 3: Prediction errors for five testing flows over airfoils, including two interpolated flows, two weakly interpolated flows, and one extrapolated flow. The relative L_2 norm errors for solution (velocity, pressure, and internal energy per unit mass in training region Ω_T) and the relative errors for drag and lift coefficients are presented.

	Interpolation		**Weak interpolation**		**Extrapolation**
Prediction error (%)	NACA0010	NACA0011	NACA0012	NACA0015	NACA0008
Solution in Ω_T	3.0	5.9	3.3	3.3	5.5
Drag coefficient, C_D	3.6	8.6	15.9	11.4	6.6
Lift coefficient, C_L	3.0	3.9	5.4	4.5	2.7

weak interpolation scenarios, it is evident that simulations initialized with neural network predictions require significantly fewer iterations to reach a specific residual level when compared to those initialized with uniform fields or potential flows.

The accelerated convergence can be attributed to two key factors. First, neural network predictions enhance convergence stability, as clearly seen in the iterative convergence processes shown in the left three columns of Fig. 10. Simulations initialized with neural network predictions effectively bypass the initial oscillatory phases that are present in simulations initialized with uniform fields or potential flows, transforming

oscillatory convergence into a more stable and monotonic pattern. Second, neural network prediction improve convergence rate. To illustrate this, we introduce a refined initial condition, referred to as "reference", which is obtained by early stopping the numerical simulation at a residual of 10^{-5} using the uniform field for initialization. It is evident that the convergence processes based on the reference also exhibit monotonic behavior without oscillations. However, neural network initialization outperforms the reference when aiming to achieve a relatively small convergence criterion, as demonstrated by the slopes of the convergence curves and their intersections.

The superiority of neural network initialization is more pronounced in the fourth column of Fig. 10, where the wall time for numerical simulations using different initialization methods is compared. For all three selected convergence criteria, neural network initialization consistently achieves more than twofold acceleration ratios. Notably, neural network initialization outperforms the reference significantly for the convergence criterion $\mathcal{R} < 10^{-8}$, and slightly for $\mathcal{R} < 10^{-7}$. This advantage clearly illustrates that in highly nonlinear cases, neural network initialization for acceleration doesn't depend solely on providing initial conditions with small residuals but also faster convergence during iterations, which is distinct from scenarios with weak nonlinearity where acceleration primarily depends on achieving a smaller initial residual.

The neural network initialization approach does not present clear advantages over alternative initialization methods for the extrapolated testing flow, as illustrated in Figure 10(c). Despite an initial, brief lead due to the neural network prediction with a smaller residual, the subsequent sharp increase in residual significantly hinders the convergence. Consequently, the wall time required for the simulation using neural network initialization is nearly identical to that of simulations employing uniform field or potential flow for initialization. Nevertheless, we continue to ensure the accuracy of solutions in extrapolated cases thanks to the engagement of CFD solvers. In contrast, achieving high accuracy in the extrapolation regime is a formidable task for neural network-based surrogate models, given their inherent nature of interpolation.

Neural network initialization also significantly accelerates the convergence of integral quantities, as illustrated in Fig. 11. In the left and middle panels, we present the iterative errors of drag and lift coefficients for the interpolated testing flow over NACA0011 using three different initialization methods. Here, we consider the drag and lift coefficients at the convergence point, where the iterative residuals satisfy $\mathcal{R} < 10^{-8}$, as their ground truth values for calculating the iterative errors. We can observe that, despite initial oscillations, the iterative errors of drag and lift coefficients achieved with neural network initialization rapidly converge to the 5% and 1% levels, exhibiting clear advantages over those resulting from the use of uniform fields or potential flows for initialization. Consequently, neural network initialization attains a 16-fold acceleration ratio for the 5% error level and an 11-fold acceleration ratio for the 1% error level for both drag and lift coefficients, as shown in the right panel. It is important to note that monitoring iterative errors of integral quantities alone, rather than iterative residuals, can be misleading and should not be employed in the numerical solution of

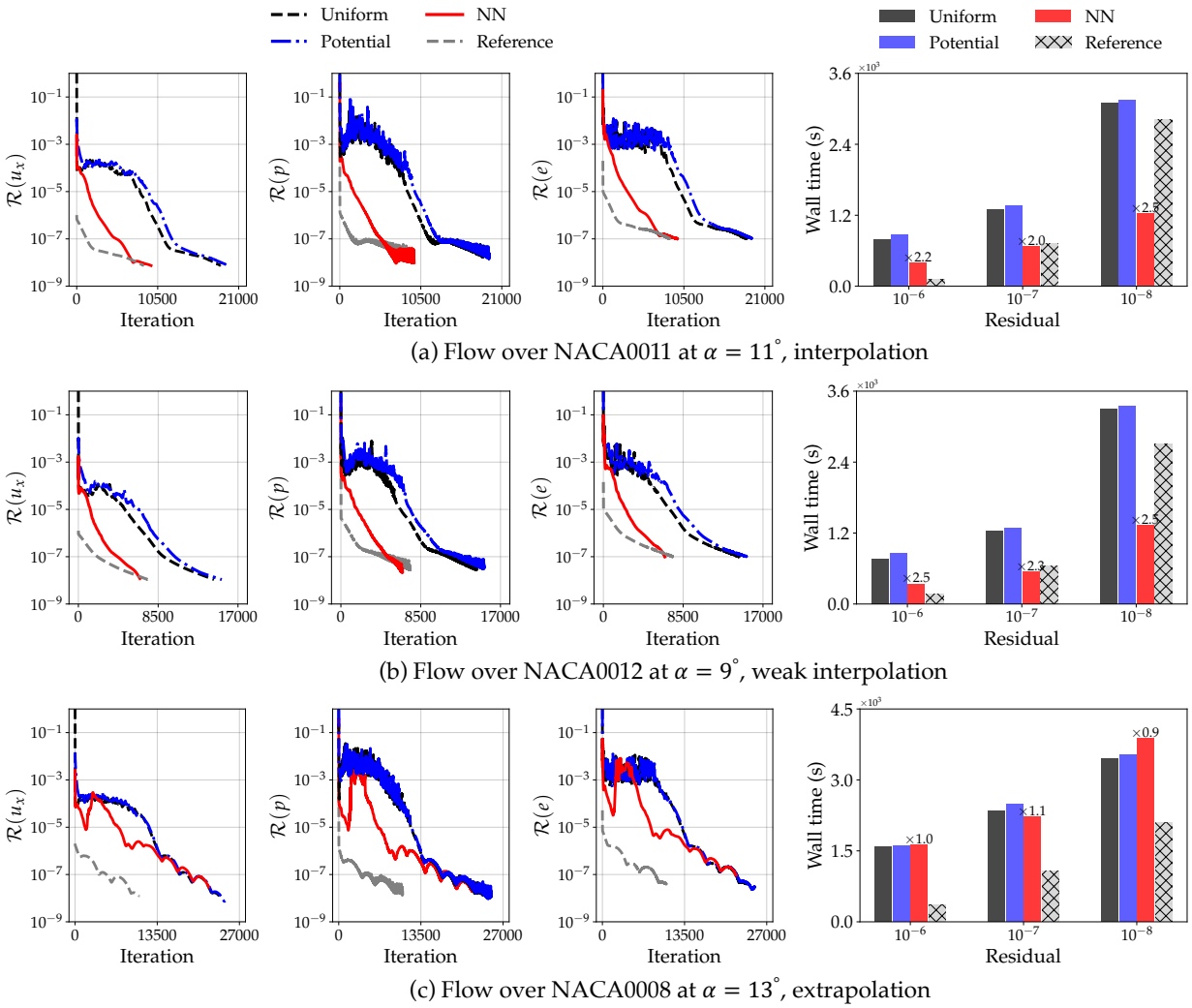


Figure 10: Comparison of three different initialization methods: uniform fields, potential flow, and neural network prediction for (a) interpolated, (b) weakly interpolated, and (c) extrapolated testing flows. Left three columns: the convergence processes of velocity, pressure, and internal energy per unit mass; right column: the wall time required to satisfy three different convergence criteria (10^{-6}, 10^{-7}, and 10^{-8}). The reference here serves as an ideal initial condition at a residual of 10^{-5}, offering a more comprehensive assessment of the acceleration performance of neural network predictions.

PDEs. In this study, we investigate the acceleration performance in the convergence of integral quantities under the assumption of a more experienced scenario, where researchers/engineers have a high degree of confidence in the favorable behavior of iterative residuals.

Finally, the warm-start approach consistently exhibits acceleration performance across various iterative algorithms with different linear equation solvers. Specifically, we have chosen three distinct combinations of linear equation solvers for the numerical solution of Eq. 5 after initialization. Each combination employs two distinct linear equation solvers: one used for solving pressure, and the other used for solving velocity

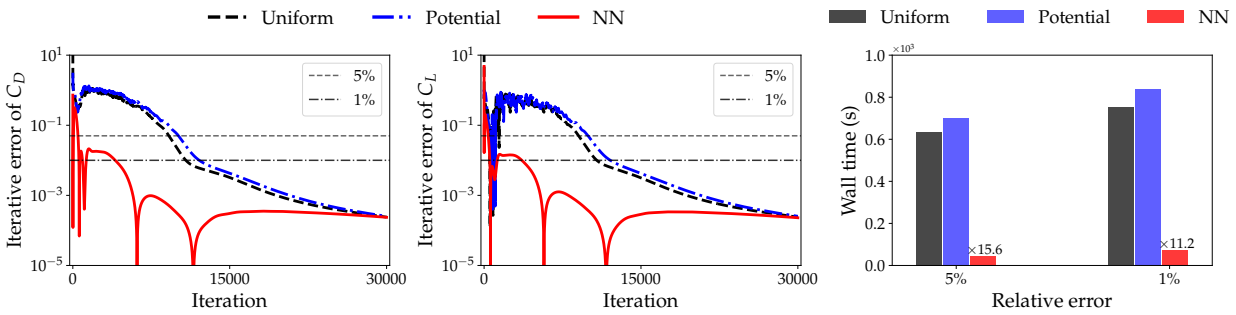
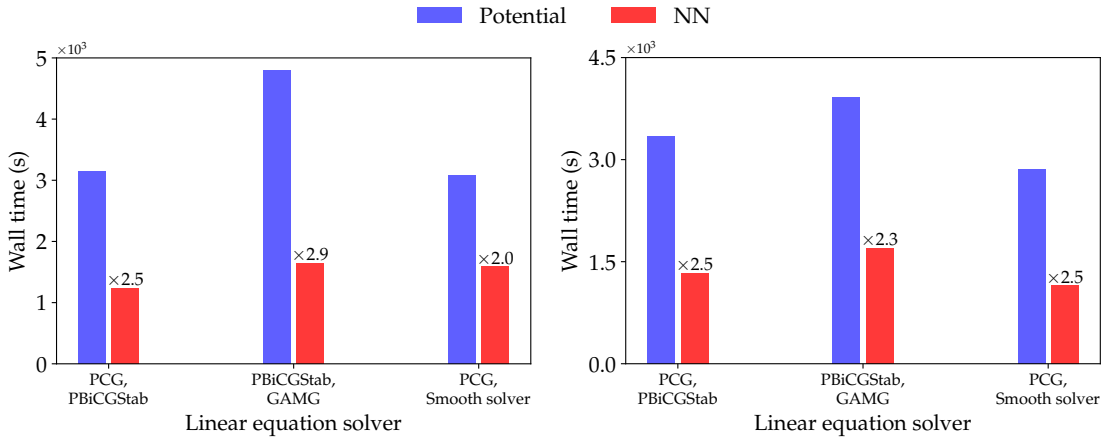


Figure 11: Comparison of three different initialization methods: uniform field, potential flow, and neural network prediction in the convergence of integral quantities. Left two panels: iterative errors for drag and lift coefficients; right panel: the wall time required to satisfy two relative errors (5% and 1%).

and internal energy per unit mass. Details regarding the linear equation solvers and associated preconditioners/smoothers in each combination are provided in Table D.8 in Appendix D. In Fig. 12, we compare the wall time required to achieve the convergence criterion ($\mathcal{R} < 10^{-8}$) when using neural network prediction and potential flow for initialization, employing the three different combinations of linear equation solvers. It is clear that neural network initialization consistently achieves a stable acceleration ratio, ranging between two and three, for each combination of linear equation solvers in both interpolated and weakly interpolated testing flows. This remarkable consistency highlights the robustness of our proposed warm-start approach in accelerating the convergence processes.



(a) Flow over NACA0011 at $\alpha = 11^{\circ}$, interpolation

(b) Flow over NACA0012 at $\alpha = 9^{\circ}$, weak interpolation

Figure 12: Acceleration performance across different linear equation solvers in both interpolated and weakly interpolated testing flows over airfoils, demonstrating the robustness of the warm-start approach. Three different configurations are compared: (1) PCG and PBiCGStab, (2) PBiCGStab and GAMG, and (3) PCG and smooth solver. In each configuration, the former is utilized for solving pressure and the latter is employed for solving velocity and internal energy per unit mass.

4. Discussions

4.1. Acceleration for multiple simulations

Despite the acceleration effect demonstrated in the above results, it is crucial to clarify that the warm-start model does not result in savings of total computation time for a single numerical simulation, as the time invested in data generation and model training far surpasses the time saved through improved initialization. Hence, we suggest employing this approach in scenarios involving multiple simulations to realize a substantial reduction in overall time. The reduction in total time depends primarily on two key factors: first, the time required to develop a warm-start model, and second, the number of numerical simulations needed after a neural operator is well-trained for warm-start. As such, we explore strategies for reducing model preparation time and investigate the impact of the number of simulations on the ultimate acceleration performance.

When preparing the warm-start model, we found that training the model with lower-quality data can lead to acceleration results comparable to those presented above. The significant advantage here lies in the considerably reduced time required for generating such lower-quality data. To investigate the influence of training data quality, we employed three different lower-quality datasets to train the neural operator and assessed their impact on acceleration performance in the aforementioned two scenarios. In Fig. 13, we present the comparison for the testing flow around an elliptical cylinder. The warm-start models trained with data at residuals of 10^{-3}, 10^{-4}, and 10^{-5} exhibit increasing acceleration effects, with the last achieving results close to those from training with data at a residual of 10^{-8}, as shown in the left two panels. In the third panel, we illustrate the performance of the corresponding four models, highlighting minimal prediction errors ($< 2\%$) and the similar, large acceleration ratios for the two models trained with data at residuals of 10^{-5} and 10^{-8}, respectively.

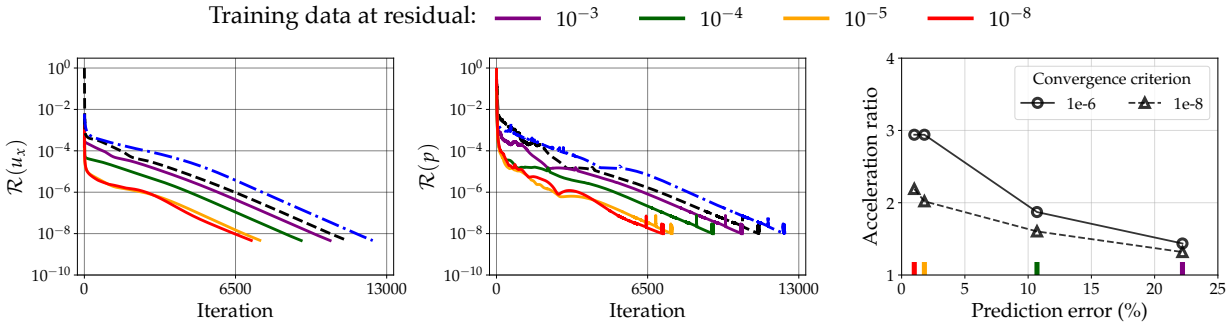
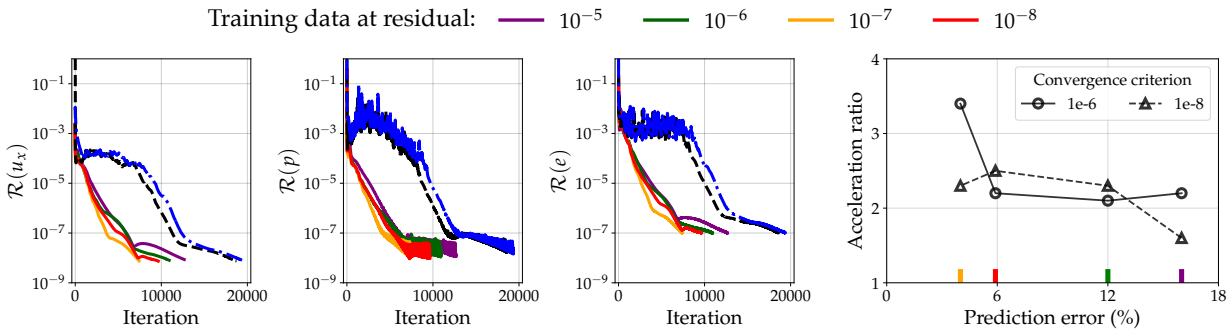


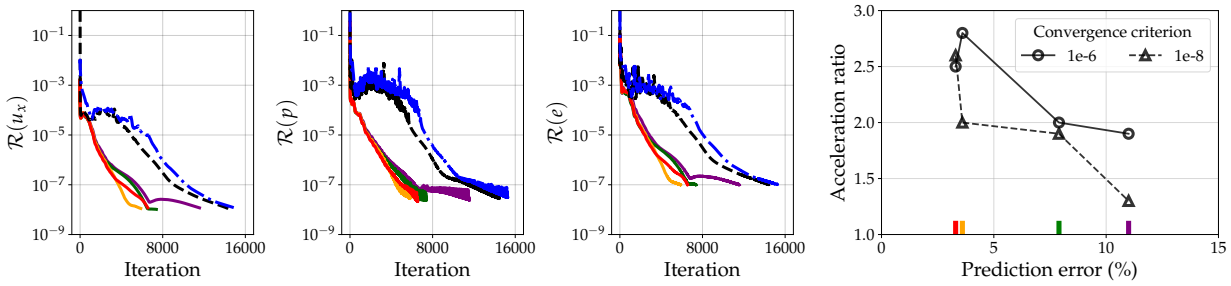
Figure 13: Comparison of the acceleration performance of the warm-start models trained with data of different quality in the testing flow over an elliptical cylinder. The left two panels show the corresponding convergence processes of velocity and pressure, respectively, while the right panel illustrates their prediction errors and acceleration ratios based on two distinct convergence criteria.

Similar effects of training data quality are also found in the second scenario for turbulent flows over

airfoils, as shown in Fig. 14. From the convergence processes illustrated in the first three columns, we can observe that all warm-start models trained with lower-quality data exhibit noticeable acceleration compared to those initialized with uniform or potential flows. Notably, the models trained with data at residuals of 10^{-6} and 10^{-7} display convergence processes that are similar to those based on data at a residual of 10^{-8}. This similarity is more clearly shown in the fourth column. The prediction error provided by the model trained with data at a residual of 10^{-7} closely approximates that associated with the residual of 10^{-8}, with both falling below 6% in the two testing flows. Consequently, this similarity in prediction errors results in comparable acceleration ratios, both exceeding twofold improvement. On the other hand, while the prediction errors based on the model trained with data at a residual of 10^{-6} nearly double, the corresponding acceleration ratios remain relatively stable, with the values about two. The results presented above demonstrate the practicality of training warm-start models using lower-quality data. This strategy reduces the time required for model development while preserving acceleration ratios of at least twofold.



(a) Flow over NACA0011 at $\alpha = 11^{\circ}$, interpolation



(b) Flow over NACA0012 at $\alpha = 9^{\circ}$, weak interpolation

Figure 14: Comparison of the acceleration performance of the warm-start models trained with data of different quality in the testing flows over airfoils. The left three panels show the corresponding convergence processes of velocity, pressure, and internal energy per mass, respectively, while the right panel illustrates their prediction errors and acceleration ratios based on two distinct convergence criteria.

We now present a potential application scenario for the warm-start approach. Based on the findings mentioned above, we assume that the warm-start model, developed to accelerate flow simulations over airfoils,

is trained using a lower-quality dataset with a residual tolerance of 10^{-6} and is capable of achieving a twofold acceleration ratio. The total preparation time for the model is approximately 10 hours, comprising roughly 3.5 hours for data generation across 12 training flows and approximately 6.5 hours for model training. We have conducted a comparative analysis of the total time required for simulating different quantities of flows within the pipeline, using uniform fields for initialization versus neural network predictions for initialization. The results of this comparison are presented in Table 4. It is evident that for a single numerical simulation ($N = 1$), the time required for neural network initialization is nearly ten times that based on uniform initialization due to the model preparation phase. However, as the number of simulations increases, the time requirements for both approaches gradually converge. Notably, when $N = 15$, the time for both approaches becomes almost identical. When $N = 100$ and $N = 1000$, the warm-start method achieves a substantial reduction in total time, offering acceleration ratios of approximately twofold, with the latter exhibiting even more pronounced time savings. Consequently, developing a neural operator-based warm-start model is well-justified in these scenarios.

Table 4: Comparison of total wall time for varying simulation quantities: uniform initialization vs. neural network initialization. The time for model preparation, including 3.5 hours for data generation and 6.5 hours for model training, is considered in the neural network approach.

	Number of simulations				
Wall time comparison	$N = 1$	$N = 10$	$N = 15$	$N = 100$	$N = 1000$
Uniform initialization (h)	1.2	12	18	120	1200
Neural operator initialization (h)	10.5	15.3	18	63	540
Acceleration ratio in total time	0.1	0.8	1.0	1.9	2.2

4.2. Limitation of neural networks/operators as surrogate models

Directly using neural networks or neural operators as surrogate models has attracted increasing interest in various scientific simulations in recent years. Despite the impressive predictive performance and fast evaluation speed, certain limitations of neural network-based surrogate models must be acknowledged. Specifically, the efficacy of neural network-based surrogate models thrives in scenarios characterized by a relatively lower dimensionality of the parameter space. When trained on extensive and diverse datasets/flows, these models demonstrate exceptional performance, enabling the discernment of intricate patterns and engendering commendable predictions. Nevertheless, the acquisition of such large-scale datasets demands substantial computational and data collection resources, which renders them impractical in applications with high-dimensional parameter spaces and limited computational capabilities. On the other hand, while these models perform well in fields covered by the training data, their efficacy diminishes considerably when extrapolating beyond

the bounds of the training data. It is crucial to note that certain fields pose challenges in distinguishing between interpolation and extrapolation cases. Flows through porous media and urban canopies stand as prime examples [23, 24], where the heterogeneous composition of pores, voids, solid phases, and city-specific multi-scale land surface render the parameterization of geometries in these flows a difficult task.

In contrast, our super-fidelity approach, which integrates classical solvers with neural operators, addresses these limitations. The classical iterative algorithm ensures accuracy and robustness in challenging scenarios, while the neural operator improves efficiency for cases well-represented in the training data.

5. Conclusion

In this study, our primary objective is to accelerate the solution process of a high-fidelity computational model described by steady-state PDEs without sacrificing solution accuracy. To achieve this, we introduce the idea of "super-fidelity", which refers to the training of a neural operator to efficiently map solutions from a lower-fidelity model to the high-fidelity model. The lower-fidelity model allows for considerably faster computation, yet retains crucial physical insights. The solution predicted by the trained neural operator is not considered the final result; instead, it serves as an initial condition for solving the high-fidelity model, enhancing solution efficiency and ensuring result accuracy.

We have demonstrated the efficacy of this approach in two distinct scenarios: incompressible laminar flows around elliptical cylinders at low Reynolds numbers and turbulent flows over airfoils at high Reynolds numbers. These scenarios represent typical scientific computing situations. In the first scenario characterized by weak nonlinearity, the iterative convergence exhibits a monotonic pattern. In contrast, the second scenario marked by strong nonlinearity displays an oscillatory convergence pattern. For each scenario, we first highlight the limitations of relying solely on neural operator predictions as final solutions, and then assess the acceleration achieved by using neural operator predictions for initialization, comparing it with conventional initialization methods based on uniform fields or potential flows.

The neural operator initialization delivers no less than a twofold acceleration while maintaining the same level of accuracy. It is noted that the acceleration behavior differs between the two scenarios. In the first scenario with weak nonlinearity, the acceleration is attributed to the provision of initial condition with a smaller residual while the convergence rate remains nearly unchanged. In the second scenario marked by strong nonlinearity, the refined initial condition effectively transforms the oscillatory convergence into a more stable, monotonic pattern. Additionally, the convergence rate also improves. Although the acceleration performance in extrapolated testing flows may not be as pronounced due to changes in flow physics, the accuracy of the results is still guaranteed, thanks to the iterations from traditional solvers.

We suggest the application of this method in scenarios involving multiple simulations to achieve a substantial reduction in overall time, even when taking the time investment in model development into account.

To further optimize the acceleration, we have introduced a practical strategy to reduce the time for model preparation. Our findings show that training the neural operator using lower-quality data still yields similar acceleration performance, while significantly reducing the time required for generating training data.

Acknowledgments

X.-H. Zhou is supported by the U.S. Air Force under agreement number FA865019-2-2204. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The computational resources used for this project were provided by the Advanced Research Computing (ARC) of Virginia Tech, which is gratefully acknowledged.

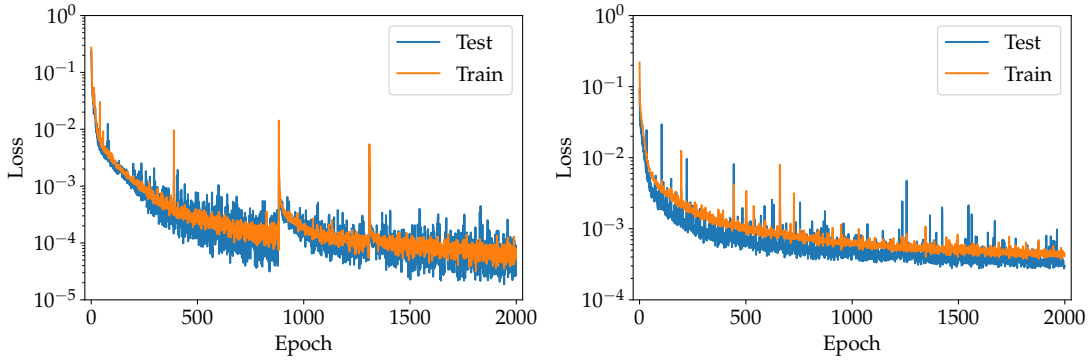
Appendix A. Neural operator architecture and training details

Detailed neural operator (VCNN-e) architecture, consisting of an embedding network and a fitting network, is provided in Table A.5. The embedding network operates identically on the scalar features $\boldsymbol{c}^L$ attached to each point in the cloud $\mathcal{Q}$ and outputs a row of m elements in matrix $\mathcal{G}$. The fitting network maps the invariant feature matrix $\mathcal{D}$ to an invariant vector $\mathcal{E}$ and predictions of scalar quantities $\boldsymbol{c}^H$.

The parameters of VCNN-e are optimized using the Adam optimizer [25], with the standard mean squared error (MSE) as the loss function. In both flow scenarios, the training processes take 2000 epochs, with batch sizes of 256 and 128 for the first and second scenarios, respectively. The learning rate is set to be 0.001 at the beginning of the training process. The training and testing losses during the training processes are shown in Fig. A.15.

Table A.5: Detailed architectures of the embedding network and the fitting network in VCNN-e. The architecture specifies the number of neurons in each layer in sequence, from the input layer to the hidden layers (if any) and the output layer. The numbers of neurons in the input and output layers are highlighted in bold. Note that the number of output neurons in the fitting network for the second scenario (i.e., flows over airfoils) changes to 67 with two additional scalar features, internal energy per unit mass e and turbulent eddy viscosity ν_t, i.e., $\boldsymbol{c}^H \in \mathbb{R}^3$.

	Embedding network ($\boldsymbol{c}^L \mapsto \mathcal{G}$)	Fitting network ($\mathcal{D} \mapsto \mathcal{E}, \boldsymbol{c}^H$)
No. of input neurons	6	$m \times m' = 256\,(\mathcal{D} \in \mathbb{R}^{64\times 4})$
No. of hidden layers	3	2
Architecture	(**6**, 16, 32, 64, **64**)	(**256**, 64, 64, **65**)
No. of output neurons	$m = 64$	65 ($\mathcal{E} \in \mathbb{R}^{64}$, $\boldsymbol{c}^H \in \mathbb{R}$)
Activation functions	ReLU, Linear (last layer)	ReLU, Linear (last layer)
No. of trainable parameters	6928	24833



(a) Flow over elliptical cylinder (b) Flow over airfoil

Figure A.15: Training and testing losses for two different scenarios: (a) flows over elliptical cylinders at low Reynolds numbers, and (b) flows over airfoils at high Reynolds numbers. Both the training and testing losses at epoch 2000 are within the range of $[10^{-5}, 10^{-4}]$ in (a), and $[10^{-4}, 10^{-3}]$ in (b).

Appendix B. Residual definition

In this study, we compare the iterative convergence of residual using different initialization methods. To ensure a fair and unbiased comparison, we calculate the residuals in an absolute manner, independent of the initial conditions. The residual calculation is as follows. For a matrix system

$$[\mathbf{A}][\mathbf{\Psi}] = [\mathbf{b}], \tag{B.1}$$

the residual vector is defined as

$$\mathbf{R} = \mathbf{b} - \mathbf{A} \cdot \mathbf{\Psi}. \tag{B.2}$$

We apply residual scaling with the following normalisation procedure:

$$n_s = \|\mathbf{A} \cdot \mathbf{\Psi} - \mathbf{A} \cdot \overline{\mathbf{\Psi}}\| + \|\mathbf{b} - \mathbf{A} \cdot \overline{\mathbf{\Psi}}\|, \tag{B.3}$$

where $\|\cdot\|$ is the matrix norm, calculated as the sum of the magnitude of each element therein, and $\overline{\mathbf{\Psi}}$ is the average of the solution vector. The scaled residual, i.e., the residual used throughout this paper, is finally given by

$$\begin{aligned} \mathcal{R} &= \frac{1}{n_s}\|\mathbf{R}\| \\ &= \frac{\|\mathbf{b} - \mathbf{A} \cdot \mathbf{\Psi}\|}{\|\mathbf{A} \cdot \mathbf{\Psi} - \mathbf{A} \cdot \overline{\mathbf{\Psi}}\| + \|\mathbf{b} - \mathbf{A} \cdot \overline{\mathbf{\Psi}}\|}. \end{aligned} \tag{B.4}$$

With this definition, the residual $\mathcal{R}$ for a uniform field is always one, as $\mathbf{\Psi} = \overline{\mathbf{\Psi}}$ always holds true.

Appendix C. Acceleration ratios across all testing flows

The comprehensive acceleration performance, achieved by neural operator initialization, is presented in Tables C.6 and C.7 for two distinct flow scenarios.

For testing flows around elliptical cylinders (see Table C.6), the acceleration ratios consistently surpass twofold. They peak at nearly threefold when the residual tolerance is set at 10^{-6} and gradually decrease to twofold with a tolerance of 10^{-8}. The decrease is in line with expectations, as the neural network prediction barely affects the convergence rate, resulting in a gradual erosion of the initial advantage when applying a stricter convergence criterion. For testing flows over airfoils (see Table C.7), despite the increased complexity, the majority of acceleration ratios remain greater than twofold. However, due to the substantial nonlinearity therein, there is no discernible correlation between the acceleration ratios and the convergence criteria.

Table C.6: Acceleration ratios for all six testing flows around elliptical cylinders using three different convergence criteria.

	Interpolation					**Extrapolation**
Convergence criterion	$Re = 32$	$Re = 36$	$Re = 40$	$Re = 44$	$Re = 48$	$Re = 52$
$\mathcal{R}(u_x), \mathcal{R}(u_y), \mathcal{R}(p) < 10^{-6}$	2.5	2.8	2.8	2.8	2.9	2.8
$\mathcal{R}(u_x), \mathcal{R}(u_y), \mathcal{R}(p) < 10^{-7}$	2.2	2.6	2.5	2.4	2.5	2.4
$\mathcal{R}(u_x), \mathcal{R}(u_y), \mathcal{R}(p) < 10^{-8}$	2.0	2.2	2.1	2.0	2.2	2.2

Table C.7: Acceleration ratios for all five testing flows over airfoils using three different convergence criteria.

	Interpolation		**Weak interpolation**		**Extrapolation**
Convergence criterion	NACA0010	NACA0011	NACA0012	NACA0015	NACA0008
$\mathcal{R}(u_x), \mathcal{R}(u_y), \mathcal{R}(p), \mathcal{R}(e) < 10^{-6}$	2.3	2.2	2.5	2.0	1.0
$\mathcal{R}(u_x), \mathcal{R}(u_y), \mathcal{R}(p), \mathcal{R}(e) < 10^{-7}$	1.9	2.0	2.3	2.1	1.1
$\mathcal{R}(u_x), \mathcal{R}(u_y), \mathcal{R}(p), \mathcal{R}(e) < 10^{-8}$	1.8	2.5	2.5	1.4	0.9

Appendix D. Linear equation solvers

To showcase the robustness of the warm-start approach, we have chosen three different combinations of linear equation solvers for the iterative solution following initialization. The linear equation solvers and associated preconditioner/smoothers in each combination are presented in Table D.8.

In the first configuration, we use PCG solver with FDIC preconditioner for pressure and PBiCGStab solver with DILU preconditioner for velocity and internal energy per unit mass. In the second configuration, we employ PBiCGStab solver with DIC preconditioner for pressure and GAMG solver with symGaussSeidel smoother for velocity and internal energy per unit mass. In the third configuration, PCG solver with FDIC preconditioner is applied for pressure, while a smooth solver with symGaussSeidel smoother is used for velocity and internal energy per unit mass. More details regarding the linear equation solvers, preconditioners,

and smoothers can be found in OpenFOAM [16].

Table D.8: Three configurations of linear equation solvers used for demonstrating the robustness of the warm-start approach.

	Configuration 1		**Configuration 2**		**Configuration 3**	
	p	$\boldsymbol{u} \mid e$	p	$\boldsymbol{u} \mid e$	p	$\boldsymbol{u} \mid e$
Linear equation solver	PCG	PBiCGStab	PBiCGStab	GAMG	PCG	smoothSolver
Preconditioner	FDIC	DILU	DIC	-	FDIC	-
Smoother	-	-	-	symGaussSeidel	-	symGaussSeidel

References

[1] J. Han, C. Ma, Z. Ma, W. E, Uniformly accurate machine learning-based hydrodynamic models for kinetic equations, Proceedings of the National Academy of Sciences 116 (44) (2019) 21983–21991.

[2] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nature machine intelligence 3 (3) (2021) 218–229.

[3] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, in: International Conference on Learning Representations, 2021.

[4] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, A. Anandkumar, Neural operator: Learning maps between function spaces with applications to pdes., J. Mach. Learn. Res. 24 (89) (2023) 1–97.

[5] W. Xiong, X. Huang, Z. Zhang, R. Deng, P. Sun, Y. Tian, Koopman neural operator as a mesh-free solver of non-linear partial differential equations, arXiv preprint arXiv:2301.10022 (2023).

[6] J. Han, X.-H. Zhou, H. Xiao, An equivariant neural operator for developing nonlocal tensorial constitutive models, Journal of Computational Physics 488 (2023) 112243.

[7] K. Shukla, V. Oommen, A. Peyvan, M. Penwarden, L. Bravo, A. Ghoshal, R. M. Kirby, G. E. Karniadakis, Deep neural operators can serve as accurate surrogates for shape optimization: a case study for airfoils, arXiv preprint arXiv:2302.00807 (2023).

[8] L.-W. Chen, N. Thuerey, Towards high-accuracy deep learning inference of compressible flows over aerofoils, Computers & Fluids 250 (2023) 105707.

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

[10] T. G. Grossmann, U. J. Komorowska, J. Latz, C.-B. Schönlieb, Can physics-informed neural networks beat the finite element method?, arXiv preprint arXiv:2302.04107 (2023).

[11] J. Li, M. A. Bouhlel, J. R. Martins, Data-based approach for fast airfoil analysis and optimization, AIAA Journal 57 (2) (2019) 581–596.

[12] C. Dong, C. C. Loy, K. He, X. Tang, Image super-resolution using deep convolutional networks, IEEE transactions on pattern analysis and machine intelligence 38 (2) (2015) 295–307.

[13] X.-H. Zhou, J. Han, H. Xiao, Frame-independent vector-cloud neural network for nonlocal constitutive modeling on arbitrary grids, Computer Methods in Applied Mechanics and Engineering 388 (2022) 114211.

[14] C. Hirsch, Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics, Elsevier, 2007.

[15] I. Ansys, Ansys fluent theory guide15. 0, ANSYS Inc (2013).

[16] The OpenFOAM Foundation, OpenFOAM User Guide (2020).
URL `https://cfd.direct/openfoam/user-guide`

[17] W. L. Oberkampf, C. J. Roy, Verification and validation in scientific computing, Cambridge university press, 2010.

[18] T. S. Phillips, C. J. Roy, Richardson extrapolation-based discretization uncertainty estimation for computational fluid dynamics, Journal of Fluids Engineering 136 (12) (2014) 121401.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems 32 (2019).

[20] M. MCKAY, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 21 (1979) 239–245.

[21] P. Spalart, S. Allmaras, A one-equation turbulence model for aerodynamic flows, in: 30th aerospace sciences meeting and exhibit, 1992, p. 439.

[22] NASA, 2D NACA 0012 Airfoil Validation Case, https://turbmodels.larc.nasa.gov/naca0012_val.html.

[23] X.-H. Zhou, J. E. McClure, C. Chen, H. Xiao, Neural network–based pore flow field prediction in porous media using super resolution, Physical Review Fluids 7 (7) (2022) 074302.

[24] Y. Lu, X.-H. Zhou, H. Xiao, Q. Li, Using machine learning to predict urban canopy flows for land surface modeling, Geophysical Research Letters 50 (1) (2023) e2022GL102313.

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

Part III

Ensemble-to-Ensemble Mapping: Neural Operator for Generalized Ensemble Filtering with Improved Data Assimilation Performance

Chapter 5

BI-EqNO: Generalized approximate Bayesian inference with an equivariant neural operator framework

BI-EqNO: Generalized Approximate **B**ayesian **I**nference with an **Eq**uivariant **N**eural **O**perator Framework

Xu-Hui Zhou[a], Zhuo-Ran Liu[b], Heng Xiao[b,*]

[a]*Kevin T. Crofton Department of Aerospace and Ocean Engineering, Virginia Tech, Blacksburg, 24060, VA, USA*
[b]*Stuttgart Center for Simulation Science, University of Stuttgart, Stuttgart, 70569, BW, Germany*

Abstract

Bayesian inference provides a robust statistical framework for updating prior beliefs based on new data via Bayes' theorem. However, performing exact Bayesian inference is often computationally infeasible in practical applications, requiring the use of approximate methods. Although these methods have achieved broad success, they continue to face significant challenges in accurately estimating marginal likelihoods. These challenges arise from the rigid, predefined functional structures of deterministic models, such as Gaussian process, and the limitations of small sample sizes in stochastic models like ensemble Kalman method, among other factors. In this work, we introduce BI-EqNO, an equivariant neural operator framework for generalized approximate Bayesian inference, developed to improve both deterministic and stochastic approaches. BI-EqNO transforms prior distributions into corresponding posteriors conditioned on observational data, achieving higher efficiency and accuracy through data-driven training. The framework is flexible, accommodating diverse prior and posterior representations with arbitrary discretizations and varying numbers of observations. Crucially, BI-EqNO's architecture is specifically designed to preserve symmetry in these functional transformations, ensuring (1) permutation equivariance between prior and posterior representations and (2) permutation invariance with respect to the observation data. We demonstrate the utility of this framework with two examples: (1) BI-EqNO as a generalized Gaussian process (gGP) for regression, and (2) BI-EqNO as an ensemble neural filter (EnNF) for sequential data assimilation. Our results indicate that gGP outperforms the traditional Gaussian process by offering a more flexible representation of mean and covariance functions. Furthermore, EnNF not only emulates the widely used ensemble Kalman filter but also has the potential to function as a "super" filter, representing a series of ensemble filters, and exhibits superior assimilation performance, particularly in small-ensemble settings. This study highlights the versatility and effectiveness of the BI-EqNO framework, which enhances approximate Bayesian inference methods through data-driven training, enabling more accurate inferences while reducing computational costs across a wide range of applications.

*Corresponding author.
Email address: heng.xiao@simtech.uni-stuttgart.de (Heng Xiao)

Keywords: approximate Bayesian inference, equivariant neural operator, Gaussian process, generalized Gaussian process, ensemble Kalman filter, ensemble neural filter

1. Introduction

Bayesian inference provides a robust statistical framework for integrating prior knowledge with observed data. It has found extensive applications across diverse fields, enhancing predictive capabilities and decision-making processes. For example, in turbulence modeling, Bayesian inference is used to calibrate turbulence models in Reynolds-averaged Navier–Stokes equations with high-fidelity data from experiment measurements and direct numerical simulations [1–4], thereby improving simulation accuracy for engineering design and optimization. In weather forecasting, it provides the foundation for a wide range of data assimilation methods such as the ensemble Kalman filter, which assimilate data from satellites, radar, and ground observations to refine predictions from numerical weather models [5–7]. Similarly, in medical diagnosis, Bayesian inference integrates clinical knowledge with patient data to enhance diagnostic precision and enable personalized care [8, 9]. The mathematical process of Bayesian inference involves updating the prior probability distribution $p(\boldsymbol{\Theta})$, representing initial beliefs about model parameters $\boldsymbol{\Theta}$, using new data $\mathcal{D}$ through the likelihood $p(\mathcal{D}|\boldsymbol{\Theta})$ to obtain the posterior distribution $p(\boldsymbol{\Theta}|\mathcal{D})$:

$$p(\boldsymbol{\Theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\Theta})\, p(\boldsymbol{\Theta})}{p(\mathcal{D})} = \frac{p(\mathcal{D}|\boldsymbol{\Theta})\, p(\boldsymbol{\Theta})}{\int p(\mathcal{D}|\boldsymbol{\Theta})\, p(\boldsymbol{\Theta})\, d\boldsymbol{\Theta}},$$

where $p(\mathcal{D})$ is the marginal likelihood serving as a normalization factor. Exact computation of posterior distributions is often computationally challenging, especially for complex models with high-dimensional parameter spaces [10]. This necessitates the use of approximate Bayesian inference methods [11].

1.1. Approximate Bayesian inference

Approximate Bayesian inference methods are generally categorized into deterministic and stochastic approaches [12]. Deterministic methods, such as maximum a posteriori (MAP) estimation, Laplace approximation, and variational inference, aim to approximate the true posterior with a simpler, computationally tractable distribution. MAP estimation identifies the mode of the posterior distribution, representing the most likely parameter values, without accounting for the uncertainty. Laplace approximation refines this by modeling the posterior as a Gaussian distribution centered at the MAP estimate, with variance derived from the curvature of the log-posterior at the mode. Variational inference uses a broader range of tractable families, such as Gaussians, exponentials, or mixtures, and optimizes the chosen approximation by minimizing the Kullback–Leibler (KL) divergence from the true posterior, allowing for more flexible representations.

Deterministic methods encounter challenges in both selecting suitable distribution families and accurately modeling the selected distributions. Gaussian distributions are frequently chosen for their mathematical

tractability in representing diverse phenomena. Once a distribution is selected, it is crucial to model it with appropriate functional structures. For instance, the mean and covariance functions defining a multivariate Gaussian distribution are essential for accurate posterior approximation, especially when data is limited. A common misstep is using an inappropriate covariance function; for example, employing a distance-decaying covariance function to model data with periodic characteristics leads to inaccurate posterior approximations. This underscores the importance of correctly modeling the chosen distribution to achieve reliable Bayesian inference.

Stochastic methods, such as Markov chain Monte Carlo (MCMC) and sequential Monte Carlo (SMC) [13], offer greater flexibility since they do not require predefined distribution families. This adaptability makes them particularly effective in managing complex and high-dimensional posterior landscapes. MCMC methods generate a Markov chain to sample from the target distribution at equilibrium, theoretically providing a comprehensive representation of the posterior distribution. Conversely, SMC methods, such as particle filters and ensemble-based filters [13], are designed for real-time data assimilation, where prior estimates are iteratively updated with new sequential observations. Particle filters represent the posterior distribution using a set of samples that are propagated and reweighted according to observed data, facilitating a flexible, non-parametric approach to handling nonlinear and non-Gaussian process [14]. The ensemble Kalman filter employs an ensemble of state samples to efficiently approximate the mean and covariance of the posterior, making it particularly suitable for large-scale systems with linear or mildly nonlinear dynamics [15–17].

Despite their flexibility and effectiveness, stochastic methods have a notable limitation in that they often require a sufficiently large sample size to accurately approximate the posterior distribution. For example, in SMC methods, a large sample size is necessary to capture the full range of system uncertainties, particularly for large-scale systems like atmosphere and ocean. This results in substantial computational intensity, as propagating numerous samples through numerical simulations demands significant processing power and memory. For instance, the European Centre for Medium-Range Weather Forecasts uses an ensemble of 51 samples for weather forecasting, with each run taking approximately 10 petaflop/s of processing power and producing several terabytes of data daily [18]. Similarly, ocean modeling using the HYbrid Coordinate Ocean Model with a $1/12^\circ$ resolution can require tens of thousands of core hours per simulation day [19]. When computational resources are limited, it is crucial to explore more efficient simulation methods and advance filtering techniques. Approaches such as reduced order modeling and surrogate modeling provide viable alternatives to traditional numerical simulations, enabling large-ensemble data assimilation with substantially reduced computational costs [20, 21]. On the other hand, to improve filtering performance, techniques like localization and inflation are commonly employed to alleviate the challenges posed by low-rank approximations with limited samples in ensemble-based methods [22]. Despite their effectiveness, these methods often require significant problem-specific tuning and may still have limitations in certain scenarios. As a result,

developing a more flexible and universally applicable filtering approach that maintains accuracy across a wide range of conditions remains a critical area of ongoing research.

1.2. Neural networks for approximate Bayesian inference

Recent advancements in machine learning offer new approaches to addressing the challenges faced by both deterministic and stochastic methods. In deterministic methods, neural networks offer a powerful and flexible tool for modeling probability distributions. Enhancements to Gaussian processes for regression problems demonstrate this utility [23–25]. Among these, neural processes have emerged as an effective and scalable approach [25]. They employ an encoder-decoder architecture where the encoder maps observation data into a latent space that captures uncertainty and variability, and the decoder maps this latent representation back to prediction values, conditioned on the prediction points. Trained on observation data, the neural processes demonstrate superior predictive capability and reduced computational complexity compared to Gaussian processes. However, they treat each target point individually without considering their covariance, thus neglecting the critical aspect of uncertainty inherent in Gaussian processes.

In stochastic methods, neural networks can provide a more accurate and flexible scheme for updating prior samples based on observations. For example, a U-Net-based neural network has been developed to enhance the ensemble Kalman filter by considering a nonlinear mapping between analysis increments and innovations [26]. This surrogate filter shows advantages in systems with non-Gaussian state distributions and under small ensemble conditions. However, as it updates each sample individually without considering ensemble statistics, it may not fully leverage the benefits of ensemble-based methods. Some studies retain the ensemble Kalman filter as an essential component of the filtering process and augment it with a trained neural-network corrector [27, 28], resulting in a two-step assimilation process.

Despite the improvements from leveraging neural networks, aforementioned studies often focus on specific application scenarios from a discrete perspective, limiting their broader applicability to real-world problems with diverse prior and posterior representations. Mathematically, we can view the prior-to-posterior transformation as a general functional mapping, which shall be applicable across various representations of both distributions. Recent developments in operator learning provide novel methodologies for handling such functional transformations. Neural operators, initially proposed for the supervised learning of solution operators in partial differential equations (PDEs) [29–31], distinguish themselves from standard neural networks by learning mappings between infinite-dimensional function spaces and adapting to different input discretizations. This adaptability allows them to handle various input forms and scales, making them highly versatile and particularly advantageous in applications requiring high-dimensional functional mappings, such as fluid dynamics and climate modeling. Neural operators have demonstrated success across a variety of fields including constitutive modeling [32, 33], aerodynamic shape optimization [34, 35], and advancing diverse scientific simulations [36–39].

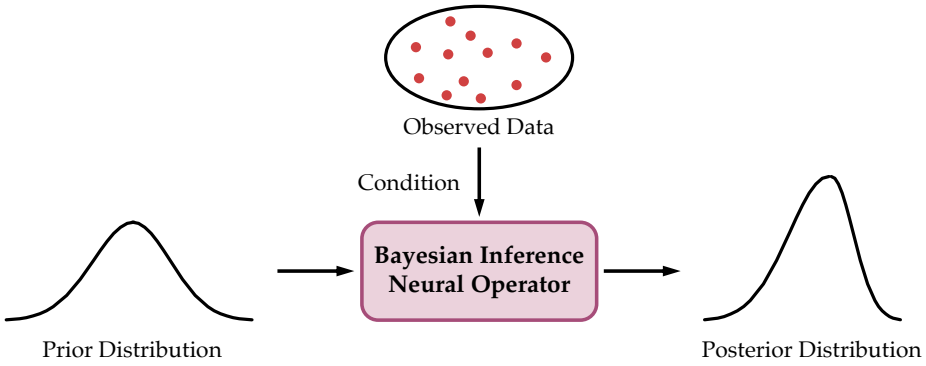


Figure 1: Schematic illustration of the neural operator-based framework for generalized approximate Bayesian inference, showing the transformation from the prior distribution to the posterior distribution through data-driven training, conditioned on the observed data.

In this work, we leverage the advantages of neural operators and develop an equivariant neural operator framework for generalized approximate Bayesian inference, referred to as BI-EqNO, to address the limitations in both deterministic and stochastic methods. Through data-driven training, BI-EqNO transforms the prior distribution into the posterior distribution, conditioned on observed data, which is illustrated in Fig. 1. This transformation allows for flexible representations of both the prior and posterior distributions, as well as the observed data. Additionally, it preserves symmetry in the functional mapping by ensuring permutation equivariance between the prior and posterior representations and maintaining invariance to the ordering of observed data. Here, permutation equivariance means that if the input data is reordered, the model adjusts its predictions to maintain the correct correspondence between input and output values. Permutation invariance, on the other hand, ensures that reordering the observed data does not affect the model's predictions. These properties are fundamental to Bayesian inference-based approaches and are essential for ensuring BI-EqNO's robustness across diverse applications. For example, in the structural health monitoring of a bridge, Gaussian processes can predict displacements throughout the structure using sensor data from various locations. If the prediction locations are reordered, the model updates the displacements accordingly, reflecting permutation equivariance. On the other hand, if the sequence of sensor measurements is rearranged, the predictions remain stable, demonstrating permutation invariance.

We demonstrate the flexibility and potential of BI-EqNO through the development of (1) the generalized Gaussian process (gGP) for regression and (2) the ensemble neural filter (EnNF) for sequential data assimilation. The generalized Gaussian process defines its mean and covariance functions using neural operators, effectively overcoming the limitations associated with predefined kernel structures in Gaussian process. Additionally, the ensemble neural filter functions as a generalized ensemble-based approach that can be trained and calibrated using data to enhance filtering performance, especially under conditions with limited ensemble sizes.

The remainder of this paper is organized as follows. Section 2 introduces BI-EqNO framework and details its further development to the generalized Gaussian process for regression and the ensemble neural filter for sequential data assimilation. Section 3 presents an evaluation of the generalized Gaussian process and the ensemble neural filter through representative case studies. Finally, Section 4 offers a summary and concluding remarks.

2. Methodology

2.1. Description of BI-EqNO framework

We propose the BI-EqNO framework to tackle challenges in approximate Bayesian inference. The schematic architecture of the BI-EqNO is illustrated in Fig. 2, showing the mapping from a collection of input elements $\{\mathbf{x}_i\}_{i=1}^N$ to the output $\{\mathbf{x}_i^{\mathrm{P}}\}_{i=1}^N$, conditioned on the observation data $\{\mathbf{d}_i\}_{i=1}^M$, through trainable neural operators. Before providing a detailed architectural description, we first present a high-level overview of its use across different scenarios.

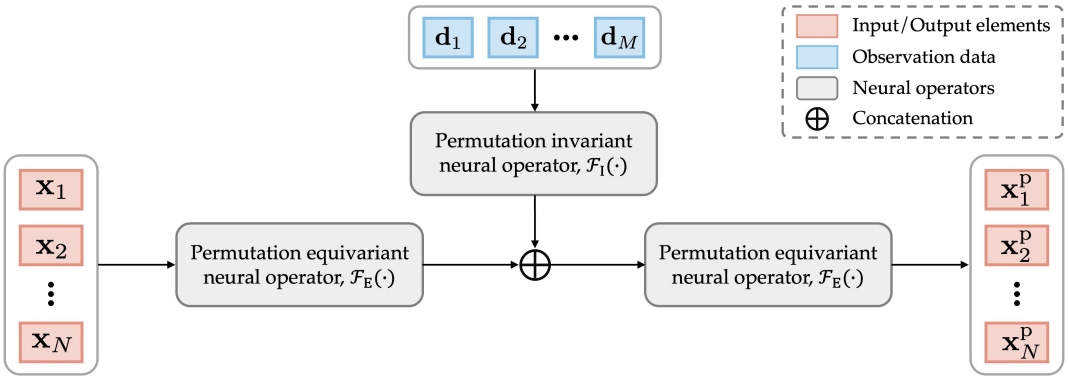


Figure 2: Schematic architecture of the equivariant neural operator framework for generalized approximate Bayesian inference. The input elements $\{\mathbf{x}_i\}_{i=1}^N$ are embedded through a permutation equivariant neural operator, while the observation data $\{\mathbf{d}_i\}_{i=1}^M$ are embedded through a permutation invariant neural operator. These embeddings are concatenated and passed through another permutation equivariant neural operator to generate the output $\{\mathbf{x}_i^{\mathrm{P}}\}_{i=1}^N$.

Scenarios of BI-EqNO usage The BI-EqNO framework is designed to transform a prior distribution to a posterior distribution conditioned on observations. However, it does not serve solely as a direct mapping between these two distributions; rather, its inputs and outputs are defined to suit different application scenarios. In the following, we present two representative use cases that illustrate its versatility.

(1) **Prior-to-posterior mapping operator.** In this scenario, BI-EqNO functions as a mapping operator that directly transforms a prior distribution into a posterior distribution over the state space. Thus, the input and output of BI-EqNO are the corresponding samples or other representations of the state from the prior and posterior distributions.

(2) **Neural implicit representation of distributions.** Neural implicit representation refers to the use of neural networks to estimate a function continuously from its discrete representations. In this context, BI-EqNO maps coordinates (such as spatial locations, time points, or positions in feature space) into desired distribution representations (such as the mean and covariance of a Gaussian distribution). Consequently, the input to BI-EqNO consists of the coordinates, while the output corresponds to the representation of the prior or posterior distribution at these coordinates.

BI-EqNO architecture In either usage scenario, the BI-EqNO should maintain permutation equivariance between the input and output elements, while ensuring the numbering of the observations does not affect the output. As shown in Fig. 2, to achieve this equivariance, the input elements $\{\mathbf{x}_i\}_{i=1}^N$ are embedded through a permutation equivariant neural operator $\mathcal{F}_{\mathrm{E}}$ to preserve the ordering of the input elements. Simultaneously, the observations $\{\mathbf{d}_i\}_{i=1}^M$ are embedded through a permutation invariant neural operator $\mathcal{F}_{\mathrm{I}}$, ensuring invariance to the ordering of the observations. The embeddings from both neural operators are then concatenated and passed through another permutation equivariant neural operator, serving as a fitting network, to produce the output $\{\mathbf{x}_i^{\mathrm{P}}\}_{i=1}^N$.

Mathematically, the permutation invariant neural operator, $\mathcal{F}_{\mathrm{I}} : (\mathbb{R}^p)^N \to \mathbb{R}^q$, satisfies the property that for any reordering σ of the indices $\{1, 2, \ldots, N\}$ and elements $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^p$, the following holds:

$$\mathcal{F}_{\mathrm{I}}(\mathbf{x}_{\sigma(1)}, \mathbf{x}_{\sigma(2)}, \ldots, \mathbf{x}_{\sigma(N)}) = \mathcal{F}_{\mathrm{I}}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N). \tag{1}$$

The architecture of the neural operator designed to achieve permutation invariance is illustrated in Fig. 3(a). This architecture employs an *embedding-averaging-fitting* structure, which acts as a universal approximation of any permutation invariant function [40]. Specifically, the input elements $\{\mathbf{x}_i\}_{i=1}^N$ are firstly processed through a shared embedding network, which applies the same neural network to each element independently. These embeddings are then averaged to ensure the result is invariant to the ordering of the input elements. The averaged embedding is subsequently passed through a fitting network to produce the final output.

On the other hand, the permutation equivariant neural operator, $\mathcal{F}_{\mathrm{E}} : (\mathbb{R}^p)^N \to (\mathbb{R}^q)^N$, satisfies the condition:

$$\mathcal{F}_{\mathrm{E}}(\mathbf{x}_{\sigma(1)}, \mathbf{x}_{\sigma(2)}, \ldots, \mathbf{x}_{\sigma(N)}) = \sigma\left(\mathcal{F}_{\mathrm{E}}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)\right). \tag{2}$$

The designed neural operator that achieves this equivariance extends the architecture of the invariant operator by concatenating the element-wise embeddings through another shared embedding network, followed by a shared fitting network, as is illustrated in Fig. 3(b). The composite architecture, which combines a permutation invariant neural operator with a neural network acting on each input element, represents one type of approximation for permutation equivariant functions. However, it is not clear if this representation serves as a universal approximation.

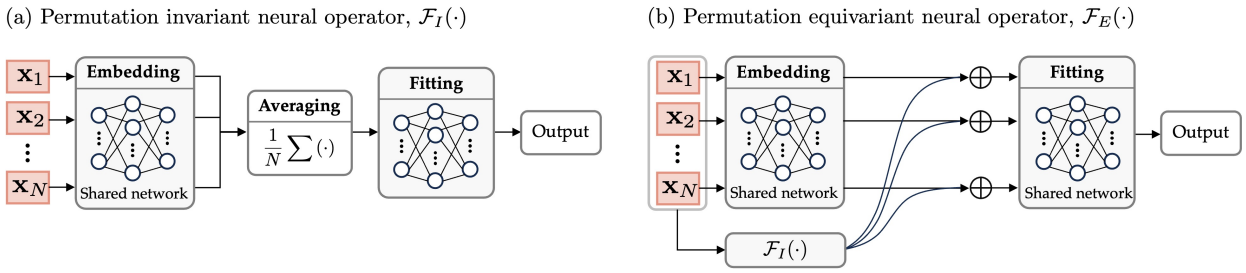


Figure 3: Architectures of (a) the permutation invariant neural operator $\mathcal{F}_{\mathrm{I}}$ and (b) the permutation equivariant neural operator $\mathcal{F}_{\mathrm{E}}$. In (a), each input element is individually processed through a shared embedding network. The resulting N embeddings are then averaged to remove the influence of ordering, and the aggregated embedding is passed through a fitting network to produce the output. In (b), $\mathcal{F}_{\mathrm{E}}$ extends $\mathcal{F}_{\mathrm{I}}$ by concatenating element-wise embeddings to preserve the ordering of the input elements.

2.2. BI-EqNO as generalized Gaussian process

Gaussian processes are extensively used in various regression tasks to predict values at unobserved locations based on a limited set of observed data. However, their effectiveness is often constrained by the need for predefined covariance structures, which can limit their flexibility and adaptability to complex data patterns. To address this limitation, we employ the BI-EqNO as a neural implicit representation of prior and posterior distributions, providing a more flexible and trainable approach to covariance modeling. This innovation leads to the development of a generalized Gaussian process.

Regression The goal of regression is to predict the values of one or more continuous target variables y given a vector $\boldsymbol{x}$ of input variables. Given a dataset $\mathcal{D} = \{X, Y\}$ of M observations with input values $X = \left[\boldsymbol{x}_1^{\mathrm{dat}}, \boldsymbol{x}_2^{\mathrm{dat}}, \ldots, \boldsymbol{x}_M^{\mathrm{dat}}\right]$ and corresponding target values $Y = \left[\tilde{y}_1^{\mathrm{dat}}, \tilde{y}_2^{\mathrm{dat}}, \ldots, \tilde{y}_M^{\mathrm{dat}}\right]^\top$, the task is to predict the target values Y^* for new input values $X^* = [\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \ldots, \boldsymbol{x}_N^*]$, where N is the number of test points. Here, the target variable y is assumed to be a scalar.

Regression with Gaussian processes Gaussian processes provide a probabilistic framework for regression by defining a distribution over the underlying functions f. The key assumption is that the function values at any finite set of inputs are jointly Gaussian. Specifically, the steps in Gaussian process regression are as follows:

(1) Specify a prior distribution as a Gaussian process:

$$f(\boldsymbol{x}) \sim \mathcal{GP}\left(\mathsf{m}(\boldsymbol{x}), \mathsf{K}(\boldsymbol{x}, \boldsymbol{x}')\right), \tag{3}$$

where $\mathsf{m}(\boldsymbol{x})$ denotes the mean function and $\mathsf{K}(\boldsymbol{x}, \boldsymbol{x}')$ denotes the kernel (or covariance) function, which together characterize the Gaussian distribution.

(2) Formulate the joint distribution between the observed target values Y and the function values $Y^* = f(X^*)$ at test points under the defined prior:

$$\begin{bmatrix} Y \\ Y^* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathsf{m}(X) \\ \mathsf{m}(X^*) \end{bmatrix}, \begin{bmatrix} \mathsf{K}(X,X) + \sigma_\varepsilon^2 \mathbf{I} & \mathsf{K}(X,X^*) \\ \mathsf{K}(X^*,X) & \mathsf{K}(X^*,X^*) \end{bmatrix} \right), \tag{4}$$

where the observed target values are assumed to be corrupted by Gaussian noise ϵ with variance σ_ϵ^2, i.e., $\tilde{y}_i^{\mathrm{dat}} = f(\boldsymbol{x}_i^{\mathrm{dat}}) + \epsilon_i$, with $\epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2)$.

(3) Compute the posterior distribution of Y^*, conditioned on the training dataset $\mathcal{D}$, using Bayes's theorem:

$$p\left(Y^* | X^*, \mathcal{D}\right) = \frac{p\left(Y^*, Y | X, X^*\right)}{p\left(Y|X\right)}. \tag{5}$$

This results in the conditional posterior Gaussian distribution $p\left(Y^*|X^*,\mathcal{D}\right) = \mathcal{N}\left(\mathsf{m}_{\mathcal{D}}(X^*), \mathsf{K}_{\mathcal{D}}(X^*,X^*)\right)$, with mean $\mathsf{m}_{\mathcal{D}}(X^*)$ and covariance $\mathsf{K}_{\mathcal{D}}(X^*,X^*)$ as follows:

$$\begin{aligned} \mathsf{m}_{\mathcal{D}}(X^*) &= \mathsf{m}(X^*) + \mathsf{K}(X^*,X)\left[\mathsf{K}(X,X) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1}\left[Y - \mathsf{m}(X)\right], \\ \mathsf{K}_{\mathcal{D}}(X^*,X^*) &= \mathsf{K}(X^*,X^*) - \mathsf{K}(X^*,X)\left[\mathsf{K}(X,X) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1}\mathsf{K}(X,X^*). \end{aligned} \tag{6}$$

From Eq. (6), it is evident that the posterior depends on the kernel function K: Both the kernel functional form and the associated hyperparameters determine the model's efficacy. The kernel functional form, such as the squared exponential kernel, is usually predetermined and remains fixed throughout the inference process, while the hyperparameters $\boldsymbol{\varphi}$ that control the kernel's length scale and variance can be optimized by maximizing the log marginal likelihood, i.e.,

$$\boldsymbol{\varphi}_{\mathrm{opt}} = \underset{\boldsymbol{\varphi}}{\mathrm{argmax}} \log p(Y|X,\boldsymbol{\varphi}). \tag{7}$$

However, selecting an appropriate kernel can be particularly challenging in real-world scenarios that involve discontinuities, such as sudden changes in signal, or multiple spatial scales, like varying levels of detail in geospatial data.

Generalized Gaussian process: Requirements To address the difficulty in kernel selection, we develop the generalized Gaussian process based on the BI-EqNO framework. Unlike the Gaussian process that relies on predefined kernels, the generalized Gaussian process uses a neural operator-based mean function $\mathsf{m}^{\mathrm{NN}}(\boldsymbol{x})$ and covariance function $\mathsf{K}^{\mathrm{NN}}(\boldsymbol{x},\boldsymbol{x}')$ to define the function distribution, i.e., $f(\boldsymbol{x}) \sim \mathcal{GP}\left(\mathsf{m}^{\mathrm{NN}}(\boldsymbol{x}), \mathsf{K}^{\mathrm{NN}}(\boldsymbol{x},\boldsymbol{x}')\right)$, allowing for greater flexibility and adaptability. Note that directly applying BI-EqNO to represent the generalized Gaussian process poses challenges due to the strict requirements for the outputs—the mean vector $\mathsf{m}^{\mathrm{NN}}(X^*)$ and the covariance matrix $\mathsf{K}^{\mathrm{NN}}(X^*,X^*)$. These include maintaining proper dimensionality of matrices, ensuring permutation invariance and equivariance with respect to observation and test points, respectively, and guaranteeing symmetry and positive semi-definiteness of the covariance matrix. Specifically, they must satisfy the following mathematical constraints:

(1) Their dimensions should adapt to the number of test points in X^*. Specifically, $\mathsf{m}^{\mathrm{NN}}(X^*) \in \mathbb{R}^N$ and $\mathsf{K}^{\mathrm{NN}}(X^*, X^*) \in \mathbb{R}^{N\times N}$.

(2) Both $\mathsf{m}^{\mathrm{NN}}(X^*)$ and $\mathsf{K}^{\mathrm{NN}}(X^*, X^*)$ must be permutation invariant to the observation points X and permutation equivariant to the test points X^*, as implied by Eq. (6) and detailed in Appendix A.

(3) The covariance matrix $\mathsf{K}^{\mathrm{NN}}(X^*, X^*)$ should be symmetric and positive semi-definite.

To satisfy these constraints, we represent the output of covariance as:

$$\mathsf{K}^{\mathrm{NN}}(X^*, X^*) = LL^\top + \exp(D), \tag{8}$$

where $L = [\boldsymbol{l}_1, \ldots, \boldsymbol{l}_N]^\top$ is a low-rank matrix and $D = \mathrm{diag}\,(d_1, \ldots, d_N)$ is a diagonal matrix. The latent vectors $\{\boldsymbol{l}_i\}_{i=1}^N$ and the diagonal elements $\{d_i\}_{i=1}^N$ must adhere to the second constraint as well.

Generalized Gaussian process: Network architecture In view of the requirements, the architecture is designed to balance structural simplicity with functional flexibility while imposing the additional constraints. The architecture incorporates both a permutation equivariant neural operator for the test points and a permutation invariant neural operator for the observation data. Specifically, as shown in Fig. 4, the test points $\{\boldsymbol{x}_i^*\}_{i=1}^N$ are embedded through a permutation equivariant neural operator, which is built upon two shared neural networks: ϕ_{int} for interaction embedding and ϕ_{self} for self-embedding. The observation data $\{(\boldsymbol{x}_i^{\mathrm{dat}}, \tilde{y}_i^{\mathrm{dat}})\}_{i=1}^M$ are embedded using a permutation invariant neural operator, based on a shared network ϕ_{d}. To simplify the architecture, the design omits the fitting components from the individual operators, as introduced earlier in Fig. 3. Instead, the embeddings from both operators are concatenated and passed through a shared fitting neural network ϕ_{fit}. This fitting network generates the final outputs: the mean values $\{\mathsf{m}^{\mathrm{NN}}(\boldsymbol{x}_i^*)\}_{i=1}^N$, the latent vectors $\{\boldsymbol{l}_i\}_{i=1}^N$ and the diagonal elements $\{d_i\}_{i=1}^N$. Mathematically, the output is given by:

$$\{\mathsf{m}^{\mathrm{NN}}(\boldsymbol{x}_i^*), \boldsymbol{l}_i, d_i\} = \phi_{\mathrm{fit}}\left(\phi_{\mathrm{self}}\left(\boldsymbol{x}_i^*\right) \oplus \frac{1}{N}\sum_{j=1}^{N} \phi_{\mathrm{int}}\left(\boldsymbol{x}_j^*\right) \oplus \frac{1}{M}\sum_{k=1}^{M} \phi_{\mathrm{d}}\left(\boldsymbol{x}_k^{\mathrm{dat}}, \tilde{y}_k^{\mathrm{dat}}\right)\right) \quad \text{for} \quad i \in \{1, \ldots, N\}. \tag{9}$$

Generalized Gaussian process: Training The generalized Gaussian process is trained by requiring it to predict a randomly selected subset of $\mathcal{D}$, denoted as $\mathcal{D}' = \{X', Y'\}$, conditioned on the remaining set $\mathcal{D}\backslash\mathcal{D}'$ as observations. The subset size can be flexible but fixed in this work for training efficiency. The parameters associated with the generalized Gaussian process, specifically the weights $\boldsymbol{w}$ in ϕ_{self}, ϕ_{int}, ϕ_{d}, and ϕ_{fit}, are optimized by minimizing the negative conditional log probability:

$$\begin{aligned}\boldsymbol{w}_{\mathrm{opt}} = \underset{\boldsymbol{w}}{\mathrm{argmin}} \Bigg[&\frac{1}{2}\left(Y' - \mathsf{m}^{\mathrm{NN}}(X'; \boldsymbol{w})\right)^\top \mathsf{K}^{\mathrm{NN}}(X', X'; \boldsymbol{w})^{-1}\left(Y' - \mathsf{m}^{\mathrm{NN}}(X'; \boldsymbol{w})\right) \\ &+ \frac{1}{2}\log|\mathsf{K}^{\mathrm{NN}}(X', X'; \boldsymbol{w})| + \frac{M'}{2}\log(2\pi)\Bigg],\end{aligned} \tag{10}$$

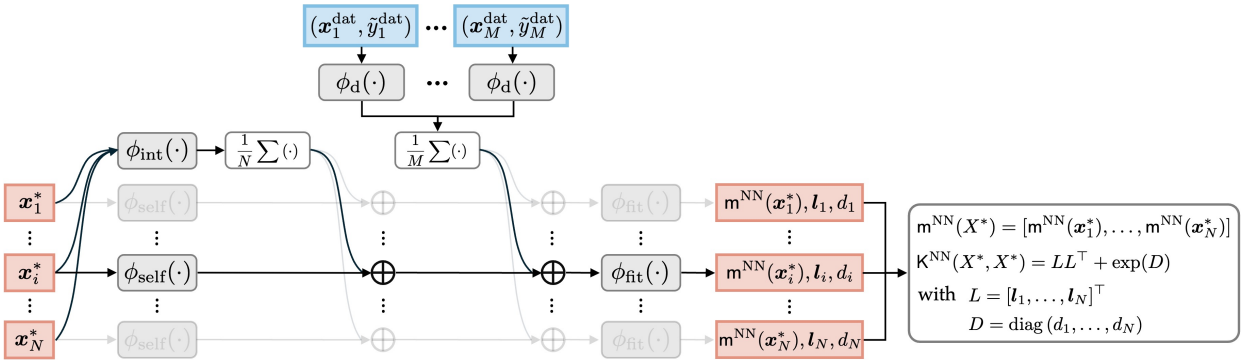


Figure 4: Architecture of the generalized Gaussian process (gGP) for regression. It defines a permutation equivariant mapping from the test points $X^* = \{\boldsymbol{x}_i^*\}_{i=1}^N$ to the mean $\mathsf{m}^{\text{NN}}(X^*)$ and covariance $\mathsf{K}^{\text{NN}}(X^*, X^*)$ of their function values, conditioned on the observation data $\mathcal{D} = \{(\boldsymbol{x}_i^{\text{dat}}, \tilde{y}_i^{\text{dat}})\}_{i=1}^M$ in a permutation invariant manner. From left to right: The test points X^* are processed through two shared neural networks, ϕ_{int} for interaction embedding and ϕ_{self} for self-embedding. Meanwhile, the observation data $\mathcal{D}$ is embedded through a share network ϕ_{d}. The embeddings from both test points and observation data are concatenated and passed through a shared fitting network ϕ_{fit} to generate the predicted function values $\{\mathsf{m}^{\text{NN}}(\boldsymbol{x}_i^*)\}_{i=1}^N$ and the covariance matrix $\mathsf{K}^{\text{NN}}(X^*, X^*)$, based on the latent vectors $\{\boldsymbol{l}_i\}_{i=1}^N$ and the diagonal elements $\{d_i\}_{i=1}^N$.

where $|\cdot|$ denotes the determinant of a matrix and M' is the number of points in X'. After training, the posterior distribution of the function values at test points X^* becomes

$$p\left(Y^* | X^*, \mathcal{D}\right) = \mathcal{N}\left(\mathsf{m}^{\text{NN}}(X^*), \mathsf{K}^{\text{NN}}(X^*, X^*); \boldsymbol{w}_{\text{opt}}\right), \tag{11}$$

where the mean vector and covariance matrix are fully determined by the optimized weights $\boldsymbol{w}_{\text{opt}}$, given the entire set of the observations $\mathcal{D}$.

Comparison to Gaussian process We compare the Gaussian process (GP) with the generalized Gaussian process (gGP) for regression, as summarized in Table 1. In the standard GP framework, the prior distribution of $f(X^*)$ is characterized by the mean vector $\mathsf{m}(X^*)$ and the covariance matrix $\mathsf{K}(X^*, X^*; \boldsymbol{\varphi}_{\text{init}})$, where $\boldsymbol{\varphi}_{\text{init}}$ denotes the initial hyperparameters of a predefined kernel function. The posterior distribution is analytically derived via Bayes' theorem, updating the mean and covariance to $\mathsf{m}_{\mathcal{D}}(X^*)$ and $\mathsf{K}_{\mathcal{D}}(X^*, X^*)$, respectively, which still depend on the initial hyperparameters $\boldsymbol{\varphi}_{\text{init}}$. Further optimization adjusts these hyperparameters to $\boldsymbol{\varphi}_{\text{opt}}$.

In contrast, gGP employs neural operators to model the mean and covariance, denoted as $\mathsf{m}^{\text{NN}}(X^*)$ and $\mathsf{K}^{\text{NN}}(X^*, X^*)$, both parameterized by the neural operator weights $\boldsymbol{w}$. The posterior distribution is obtained via data-driven training, with the weights being optimized from initial values $\boldsymbol{w}_{\text{init}}$ to $\boldsymbol{w}_{\text{opt}}$. This comparison underscores key methodological differences and optimization approaches: GP relies on hyperparameter tuning with a predefined kernel structure, while gGP leverages neural networks for enhanced flexibility in kernel representation.

		Gaussian process (GP)	Generalized Gaussian process (gGP)
Bayesian inference	**Prior** $p(f(X^*)\|X^*)$	$\mathcal{N}\left(\mathsf{m}(X^*), \mathsf{K}(X^*, X^*; \boldsymbol{\varphi}_{\text{init}})\right)$	$\mathcal{N}\left(\mathsf{m}^{\text{NN}}(X^*), \mathsf{K}^{\text{NN}}(X^*, X^*); \boldsymbol{w}_{\text{init}}\right)$
	Posterior $p(f(X^*)\|X^*, \mathcal{D})$	$\mathcal{N}\left(\mathsf{m}_{\mathcal{D}}(X^*), \mathsf{K}_{\mathcal{D}}(X^*, X^*); \boldsymbol{\varphi}_{\text{init}}\right)$ obtained by Bayes' theorem, see Eq. (6)	$\mathcal{N}\left(\mathsf{m}^{\text{NN}}(X^*), \mathsf{K}^{\text{NN}}(X^*, X^*); \boldsymbol{w}_{\text{opt}}\right)$ obtained by training neural networks
	Posterior with optimization	$\mathcal{N}\left(\mathsf{m}_{\mathcal{D}}(X^*), \mathsf{K}_{\mathcal{D}}(X^*, X^*); \boldsymbol{\varphi}_{\text{opt}}\right)$	— (same as the above cell)
Optimization		Hyperparameters $\boldsymbol{\varphi}$ with predefined kernel functional form	Neural network weights $\boldsymbol{w}$ with flexible kernel functional form

Table 1: Comparison of Gaussian process (GP) and generalized Gaussian process (gGP) for regression. The comparison highlights the methodological differences, focusing on the flexibility of neural networks in gGP, which allows for a more adaptable and flexible kernel representation compared to the predefined kernel structure in GP.

2.3. BI-EqNO as ensemble neural filter

We employ the BI-EqNO as a surrogate operator to update the state of a dynamic system with new observation data, leading to the ensemble neural filter (EnNF). This method provides a flexible, nonlinear update scheme that can potentially represent various ensemble-based filters, such as the ensemble Kalman filter (EnKF), and demonstrates certain advantages in small-ensemble scenarios.

Sequential data assimilation Sequential data assimilation involves continuously updating the state of a dynamic system as new observations become available. The goal is to combine model predictions and observational data to improve the state estimate over time. The state evolution is typically represented by a dynamic model, $\mathbf{z}(t_k) = \mathcal{M}\left(\mathbf{z}(t_{k-1})\right) + \mathbf{w}(t_k)$, where $\mathbf{z}(t_k)$ is the system's state at time t_k, $\mathcal{M}$ is the model operator, and $\mathbf{w}(t_k)$ is the model error. Observations are related to the state through an observation model, $\mathbf{d}(t_k) = \mathcal{H}\left(\mathbf{z}(t_k)\right) + \mathbf{v}(t_k)$, where $\mathbf{d}(t_k)$ is the observation vector, $\mathcal{H}$ is the observation operator mapping the state vector into observation space, and $\mathbf{v}(t_k)$ is the observation error. At each time step, the model's forecast of the state is updated using the latest observations, incorporating uncertainties in both the model and the observations. This iterative process refines the state estimate to ensure it remains as accurate as possible with new data.

Data assimilation with ensemble Kalman filter The ensemble Kalman filter (EnKF) is a sequential Monte Carlo method designed for sequential data assimilation in complex, high-dimensional systems, such as those found in geosciences and hydrology. It leverages a finite ensemble of states, comprising significantly fewer realizations than the state dimension, to approximate state uncertainty. Specifically, the EnKF maintains an ensemble of state realizations, $\mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N) \in \mathbb{R}^{n\times N}$, where each realization represents a possible state of the system. During an assimilation window, each realization is propagated through a numerical model to generate the forecast ensemble, $\mathbf{Z}^{\mathrm{f}} = \left(\mathbf{z}_1^{\mathrm{f}}, \mathbf{z}_2^{\mathrm{f}}, \ldots, \mathbf{z}_N^{\mathrm{f}}\right)$. These forecasts, also known as the prior ensemble, are then updated based on observations, $\mathbf{D} = (\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_N) \in \mathbb{R}^{m\times N}$, to obtain the posterior ensemble, $\mathbf{Z}^{\mathrm{a}} = (\mathbf{z}_1^{\mathrm{a}}, \mathbf{z}_2^{\mathrm{a}}, \ldots, \mathbf{z}_N^{\mathrm{a}})$. The update equation for each realization follows:

$$\mathbf{z}_i^{\mathrm{a}} = \mathbf{z}_i^{\mathrm{f}} + \mathbf{K}\left(\mathbf{d}_i - \mathcal{H}(\mathbf{z}_i^{\mathrm{f}})\right) \quad \text{with} \quad \mathbf{K} = \overline{\mathbf{C}}_{zy}\left(\overline{\mathbf{C}}_{yy} + \overline{\mathbf{C}}_{dd}\right)^{-1}, \tag{12}$$

where $\mathbf{K}$ is the Kalman gain matrix, $\overline{\mathbf{C}}_{zy}$ is the ensemble covariance matrix between the state vector $\mathbf{z}$ and the predicted observation $\mathbf{y} = \mathcal{H}(\mathbf{z})$, $\overline{\mathbf{C}}_{yy}$ is the ensemble covariance matrix of predicted observation, and $\overline{\mathbf{C}}_{dd}$ is the ensemble covariance matrix of observation errors.

As suggested by Eq. (12), the assimilation performance of EnKF relies on accurate covariance estimation, which necessitates a sufficiently large ensemble size N. In practical applications, however, computational limitations often restrict the ensemble size, leading to issues such as sampling errors, underestimation of state variability, and spurious correlations. To address these challenges, techniques such as localization are employed, which limit the influence of distant observations on the state updates. While effective, these techniques can be complex to implement and must be tailored to specific problems.

Ensemble neural filter: Requirements Our goal is to develop a "super" filter capable of representing a wide range of ensemble-based filters, while being easily calibrated through data-driven training. A straightforward design approach involves using the prior and posterior state ensembles as the input and output of the BI-EqNO. However, this approach, which entails a many-to-many full-field mapping, can result in high memory demands in high-dimensional systems and pose challenges during the training process. To overcome these issues, we redesign the filter, drawing inspiration from the core principles of the ensemble Kalman filter. The redesigned filter, referred to as the ensemble neural filter, adheres to the following requirements:

(1) The ensemble neural filter is flexible with arbitrary ensemble sizes.

(2) The permutations of prior and posterior realizations are equivariant.

(3) The update of each state variable shares the same ensemble neural filter.

(4) The ensemble neural filter depends only on the information within the observation space, consisting of observations, $\{\mathbf{d}_i\}_{i=1}^N$, and predicted observations, $\{\mathcal{H}(\mathbf{z}_i^{\mathrm{f}})\}_{i=1}^N$.

The first two requirements are straightforward to understand and implement, while the latter two are more complex and are further detailed in Appendix B.

Ensemble neural filter: Network architecture In light of the requirements, the architecture of the ensemble neural filter (EnNF) is designed to map between the prior and posterior ensembles of individual state variables for memory efficiency, while ensuring the fulfillment of other constraints. Specifically, the EnNF maps between $\{z^{\mathrm{f}}_{i,j}\}_{i=1}^{N}$ and $\{z^{\mathrm{a}}_{i,j}\}_{i=1}^{N}$, representing the prior and posterior ensembles of the j-th state variable, with this mapping shared across all $j \in \{1, \ldots, n\}$. Moreover, this mapping is uniquely conditioned on the information within the observation space. To achieve this, both the observations $\{\mathbf{d}_i\}_{i=1}^{N}$ and the predicted observations $\{\mathcal{H}(\mathbf{z}^{\mathrm{f}}_i)\}_{i=1}^{N}$ are integrated into the input of EnNF. The integration process preserves permutation equivariance, as the update of each realization depends on both the statistical properties of the integrated data and the specific data corresponding to each realization. As a result, the EnNF defines a permutation equivariant mapping from $\{(z^{\mathrm{f}}_{i,j}, \mathcal{H}(\mathbf{z}^{\mathrm{f}}_i), \mathbf{d}_i)\}_{i=1}^{N}$ to $\{z^{\mathrm{a}}_{i,j}\}_{i=1}^{N}$ for all $j \in \{1, \ldots, n\}$, which can be achieved using a single permutation equivariant neural operator, as shown in Fig. 5. The structure directly follows the design in Fig. 3(b), and therefore does not require further detailed description. Mathematically, the EnNF update equation is written as:

$$z^{\mathrm{a}}_{i,j} = \phi_{\mathrm{fit}}\left(\phi_{\mathrm{self}}\left(z^{\mathrm{f}}_{i,j}, \mathcal{H}(\mathbf{z}^{\mathrm{f}}_i), \mathbf{d}_i\right) \oplus \frac{1}{N}\sum_{i=1}^{N} \phi_{\mathrm{int}}\left(z^{\mathrm{f}}_{i,j}, \mathcal{H}(\mathbf{z}^{\mathrm{f}}_i), \mathbf{d}_i\right)\right) \quad \text{for} \quad i \in \{1, \ldots, N\} \quad \text{and} \quad j \in \{1, \ldots, n\}. \tag{13}$$

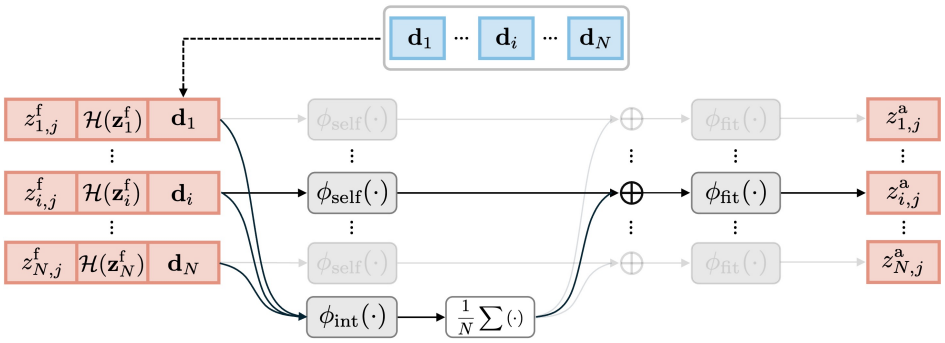


Figure 5: Architecture of the ensemble neural filter (EnNF) for sequential data assimilation. The EnNF defines a permutation equivariant mapping from prior to posterior ensembles of individual state variables, i.e., $\{z^{\mathrm{f}}_{i,j}\}_{i=1}^{N} \mapsto \{z^{\mathrm{a}}_{i,j}\}_{i=1}^{N}$, uniquely conditioned on the information within the observation space, $\{(\mathcal{H}(\mathbf{z}^{\mathrm{f}}_i), \mathbf{d}_i)\}_{i=1}^{N}$. The ensemble statistical properties and the realization-specific properties are embedded through a shared network ϕ_{int} and another shared network ϕ_{self}, respectively. These embeddings are concatenated and passed through a shared fitting network ϕ_{fit} to yield the updated state variables.

Comparison to EnKF We compare the ensemble Kalman filter (EnKF) and the ensemble neural filter (EnNF) for sequential data assimilation, as summarized in Table 2. Here, both filtering methods assume Gaussian distributions for the prior and posterior distributions of the state vector $\mathbf{z}$, though EnNF does not

inherently require this assumption and can be adapted for non-Gaussian scenarios. The mean is obtained by averaging the ensemble members, while the covariance is computed from the spread of the ensemble members around the mean. The key difference lies in the update formula. The EnKF update equation modifies each realization $\mathbf{z}_i^{\mathrm{f}}$ using the Kalman gain $\mathbf{K}$ and the innovation (the difference between the observation $\mathbf{d}_i$ and predicted observation $\mathcal{H}(\mathbf{z}_i^{\mathrm{f}})$). This leads to a linear update scheme (see Appendix B), where the updated realizations are confined to the subspace spanned by the prior ones. In contrast, the EnNF employs an equivariant neural operator to update the state, offering a more general filtering scheme that can flexibly adapt to data. Due to its nonlinearity, the EnNF allows for updates that can step outside the subspace spanned by the prior ones, potentially providing more accurate state updates in complex scenarios.

		Ensemble Kalman filter (EnKF)	**Ensemble neural filter (EnNF)**
Bayesian inference	**Prior** $p(\mathbf{z})$	$\mathcal{N}\left(\overline{\mathbf{z}}^{\mathrm{f}}, \mathbf{P}^{\mathrm{f}}\right)$, with $\overline{\mathbf{z}}^{\mathrm{f}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{z}_i^{\mathrm{f}}$ and $\mathbf{P}^{\mathrm{f}} = \frac{1}{N-1}\sum_{i=1}^{N}\left(\mathbf{z}_i^{\mathrm{f}} - \overline{\mathbf{z}}^{\mathrm{f}}\right)\left(\mathbf{z}_i^{\mathrm{f}} - \overline{\mathbf{z}}^{\mathrm{f}}\right)^{\top}$	
	Posterior $p(\mathbf{z}\|\mathcal{D})$	$\mathcal{N}\left(\overline{\mathbf{z}}^{\mathrm{a}}, \mathbf{P}^{\mathrm{a}}\right)$, with $\overline{\mathbf{z}}^{\mathrm{a}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{z}_i^{\mathrm{a}}$ and $\mathbf{P}^{\mathrm{f}} = \frac{1}{N-1}\sum_{i=1}^{N}\left(\mathbf{z}_i^{\mathrm{a}} - \overline{\mathbf{z}}^{\mathrm{a}}\right)\left(\mathbf{z}_i^{\mathrm{a}} - \overline{\mathbf{z}}^{\mathrm{a}}\right)^{\top}$	
Analysis		$\mathbf{z}_i^{\mathrm{a}} = \mathbf{z}_i^{\mathrm{f}} + \mathbf{K}\left(\mathbf{d}_i - \mathcal{H}(\mathbf{z}_i^{\mathrm{f}})\right)$	$\left\{z_{i,j}^{\mathrm{a}}\right\}_{i=1}^{N} = \mathcal{F}_{\mathrm{E}}\left(\left\{\left(z_{i,j}^{\mathrm{f}}, \mathcal{H}(\mathbf{z}_i^{\mathrm{f}}), \mathbf{d}_i\right)\right\}_{i=1}^{N}\right)$

Table 2: Comparison of ensemble Kalman filter (EnKF) and ensemble neural filter (EnNF) for sequential data assimilation. Here, both methods assume Gaussian distributions for the prior and posterior of the state vector, with the mean and covariance computed from the ensemble members. EnNF highlights its flexibility and adaptability to data by using a trainable neural operator for analysis.

3. Results

3.1. Generalized Gaussian process (gGP) for regression

We numerically investigate the regression performance of the generalized Gaussian process for learning two representative functions: a one-dimensional discontinuous function and a two-dimensional linear combination of truncated trigonometric functions. The discontinuous function is characterized by its non-smoothness with abrupt changes, while the latter function exhibits a multi-scale property. These features pose significant challenges in selecting appropriate kernel functions when using Gaussian processes. In contrast, the kernel functions in generalized Gaussian process can adapt flexibly to observational data through data-driven training, leading to superior regression performance compared to Gaussian processes.

3.1.1. Regression on a one-dimensional discontinuous function

Target function Discontinuous functions present a challenging test case for regression models due to their abrupt changes and discontinuities. Gaussian processes often fail to accurately capture these features because of the smoothness assumptions inherent in commonly used kernels, such as the squared exponential kernel. As a specific example, the one-dimensional discontinuous function (i.e., one-dimensional indicator function) employed in this study is defined as follows:

$$f(x) = \begin{cases} 0, & \text{if } 0 \leq x < 0.3, \\ 1, & \text{if } 0.3 \leq x < 0.7, \\ 0, & \text{if } 0.7 \leq x \leq 1. \end{cases} \tag{14}$$

Experiment setup The objective is to predict the function values at 1,000 uniformly distributed test points within the interval $[0, 1]$ using the provided observation data. The observation data $\mathcal{D}$ comprises input values $X = \left[x_1^{\mathrm{dat}}, \ldots, x_M^{\mathrm{dat}}\right]$ uniformly sampled from this interval, with corresponding target values $Y = \left[\tilde{y}_1^{\mathrm{dat}}, \ldots, \tilde{y}_M^{\mathrm{dat}}\right]^\top$ assumed to be corrupted by Gaussian noise:

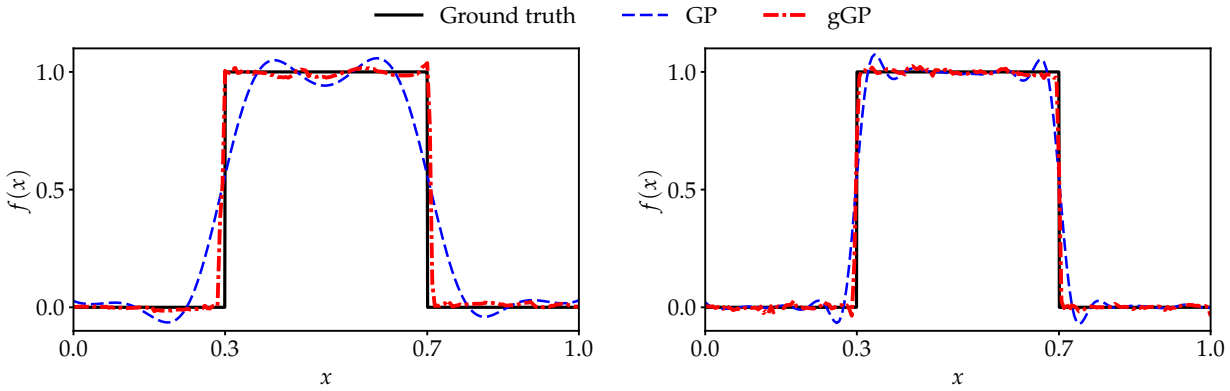
$$\tilde{y}_i^{\mathrm{dat}} = f(x_i^{\mathrm{dat}}) + \epsilon_i, \tag{15}$$

where $\epsilon_i \sim \mathcal{N}(0, 0.01^2)$. We consider two observation conditions: a sparse scenario with 30 observation points ($M = 30$) and a dense scenario with 100 observation points ($M = 100$).

For each observation condition, both Gaussian process and generalized Gaussian process are employed to perform the regression task. In the Gaussian process, a squared exponential kernel is utilized. Specifically, it is defined as $\mathsf{K}(x, x') = \sigma_f^2 \exp\left[-\frac{1}{2}(x - x')^2/\ell^2\right]$, where the hyperparameters, including the length scales ℓ and the variance σ_f, are optimized by maximizing the log marginal likelihood. In the generalized Gaussian process, the neural operator is trained on the observation data, which involves optimizing the same likelihood-based criterion to ensure accurate function approximation.

Data generation To train the generalized Gaussian process, we define the task as predicting a randomly selected subset $\mathcal{D}' = \{X', Y'\}$ from the entire dataset $\mathcal{D}$, conditioned on the remaining set $\mathcal{D}\backslash\mathcal{D}'$. Specifically, the training data consists of input-output pairs, where the input includes the test points X' and the observations from $\mathcal{D}\backslash\mathcal{D}'$, and the output corresponds to the observed target values Y'. The set X' contains $M'(< M)$ test points. Under the sparse observation condition, M' is set to 20, with the remaining 10 points used as observed points. Under the dense observation condition, M' is set to 10, with the remaining 90 points used as observed points. For both observation conditions, we randomly sample 2×10^4 data pairs without replacement, ensuring a comprehensive dataset for model training. With the training dataset, we train the generalized Gaussian process by minimizing the negative conditional log probability, as shown in Eq. (10).

Regression result The generalized Gaussian process demonstrates superior regression performance compared to the Gaussian process, particularly under the sparse observation condition, as shown in Fig. 6. In the sparse observation scenario (left plot), the Gaussian process model produces smooth predictions that fail to capture the abrupt changes in the true function. This limitation arises from the reliance on the smooth nature of the squared exponential kernel function [41], which is not well-suited to modeling the sharp transitions. In contrast, the generalized Gaussian process model accurately captures these discontinuities by learning the mean and kernel functions directly from the data, allowing it to handle intricate data features more effectively. With a more extensive dataset (right plot), both models exhibit improved accuracy. From a Bayesian perspective, as the data volume increases, the posterior distribution becomes more influenced by the data and less by the prior assumptions. For the Gaussian process model, this translates into a better ability to capture the true underlying function, including complex features such as abrupt changes. However, despite the Gaussian process model's enhanced performance with dense observations, the generalized Gaussian process model retains a certain advantage, owing to its training on the same dataset, which enhances its capacity to adapt to intricate data features.



(a) Regression with sparse observations

(b) Regression with dense observations

Figure 6: Comparison of regression performance for the one-dimensional discontinuous function using the Gaussian process (GP) and generalized Gaussian process (gGP) with sparse and dense observations. Panel (a) shows predictions with 30 observations, while panel (b) shows predictions with 100 observations. The GP produces smooth transitions in the discontinuous regions, whereas the gGP accurately captures the abrupt changes.

3.1.2. Regression on a two-dimensional multi-scale trigonometric function

Target function Multi-scale functions present another challenging case due to their inherent complexity, characterized by varying length scales and non-stationarity with distinct properties across different regions. Traditional Gaussian processes often struggle to accurately capture these features because the employed kernel functions typically assume a fixed length scale and stationarity over the entire input space. As a specific example, the two-dimensional multi-scale trigonometric function used in this study is defined as

follows:

$$
\begin{aligned}
f(x_1, x_2) = &\cos(2\pi x_1)\cos(2\pi x_2) + \sin(4\pi x_1)\sin(4\pi x_2)\mathbb{1}_{[1/4,3/4]^2}(x_1, x_2) \\
&+ \sin(8\pi x_1)\sin(8\pi x_2)\mathbb{1}_{[1/2,3/4]^2}(x_1, x_2) + \sin(16\pi x_1)\sin(16\pi x_2)\mathbb{1}_{[1/4,1/2]^2}(x_1, x_2).
\end{aligned} \tag{16}
$$

The input space for $\boldsymbol{x} = (x_1, x_2)$ is defined as $[0, 1]^2$. The two-dimensional indicator functions, such as $\mathbb{1}_{[1/4,3/4]^2}(x_1, x_2)$, activate specific components of the function only within defined regions of the input space. For instance, the term $\sin(4\pi x_1)\sin(4\pi x_2)$ is only present when (x_1, x_2) falls within the square region where both coordinates are between 1/4 and 3/4. This function incorporates four distinct length scales—1, 1/2, 1/4, and 1/8—and displays non-stationarity through its varying behaviors across different regions. These complexities require regression models to adapt to both changes in scale and local data structure, making this function an ideal benchmark for assessing a model's ability to handle intricate and heterogeneous data patterns.

Experiment setup The objective is to predict the function values at 16,384 uniformly distributed test points (2^{14} total, with 2^7 along each dimension) within the input space using the provided observation data. The observation data $\mathcal{D}$ comprises input values $X = \left[\boldsymbol{x}_1^{\mathrm{dat}}, \ldots, \boldsymbol{x}_M^{\mathrm{dat}}\right]$ uniformly sampled from the input space, with corresponding target values $Y = \left[\tilde{y}_1^{\mathrm{dat}}, \ldots, \tilde{y}_M^{\mathrm{dat}}\right]^\top$ assumed to be corrupted by Gaussian noise:

$$
\tilde{y}_i^{\mathrm{dat}} = f(\boldsymbol{x}_i^{\mathrm{dat}}) + \epsilon_i, \tag{17}
$$

where $\epsilon_i \sim \mathcal{N}(0, 0.01^2)$. We consider two observation conditions: a sparse scenario with 256 observation points ($M = 256$, with 16 along each dimension) and a dense scenario with 1,024 observation points ($M = 1024$, with 32 along each dimension). The number of observation points in the sparse scenario is determined based on the Nyquist–Shannon sampling theorem, which states that the sampling rate must be at least twice the highest frequency component present in the function to accurately reconstruct the signal. For the given multi-scale function, the highest frequency component corresponds to the finest scale of 1/8, necessitating a minimum sampling rate of 16 samples per unit length to avoid aliasing. Consequently, with 16 samples per dimension, the sparse observation grid is designed to meet this criterion, ensuring that the observed data sufficiently captures the essential features of the underlying function.

For each observation condition, both Gaussian process and generalized Gaussian process are employed to perform the regression task. In the Gaussian process approach, a squared exponential kernel, $\mathsf{K}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left[-\frac{1}{2}\left((x_1 - x_1')^2/\ell_1^2 + (x_2 - x_2')^2/\ell_2^2\right)\right]$, is utilized. The hyperparameters, including the length scales ℓ_1 and ℓ_2, and the variance σ_f, are optimized by maximizing the log marginal likelihood. In the generalized Gaussian process approach, the neural operator is trained on the observation data, which involves optimizing the same likelihood-based criterion to ensure accurate function approximation.

Data generation To train the generalized Gaussian process, we define the task as predicting a randomly selected subset $\mathcal{D}' = \{X', Y'\}$ from the entire dataset $\mathcal{D}$, conditioned on the remaining set $\mathcal{D}\backslash\mathcal{D}'$. Specifically, the training data consists of input-output pairs, where the input includes the test points X' and the observations from $\mathcal{D}\backslash\mathcal{D}'$, and the output corresponds to the observed target values Y'. The set X' contains $M'(< M)$ test points. In both the sparse and dense observation scenarios, M' is set to 32, with the remaining 224 and 992 points used as observed points, respectively. As such, we randomly sample 5×10^4 data pairs without replacement for training in each scenario. With the training dataset, we train the generalized Gaussian process by minimizing the negative conditional log probability, as shown in Eq. (10).

Regression result The generalized Gaussian process model also demonstrates superior regression performance on the multi-scale function compared to the Gaussian process model, as shown in Fig. 7. With sparse observations (top row), the Gaussian process model (left plot) produces overly smooth predictions that fail to capture the intricate multi-scale features present in the ground truth (right plot). This limitation arises from its reliance on fixed length scales in the kernel function, which restricts its ability to adapt to varying scales within the data. Consequently, the Gaussian process model struggles to capture finer-scale details and variations inherent in the multi-scale function accurately. In contrast, the generalized Gaussian process model (middle plot) shows markedly better performance in capturing the multi-scale nature of the data, even with sparse observations. This improvement is attributed to the model's flexibility in learning length scales according to the data characteristics, enabling it to more accurately represent different scales and capture subtle variations that the Gaussian process model misses. As observation density increases (bottom row), both models show improved accuracy. However, the Gaussian process model remains unable to capture fine-scale details accurately, even with dense observations spaced at a quarter of the smallest scale. In contrast, the generalized Gaussian process model nearly captures all scales and demonstrates better predictive performance.

The comparison is more clearly illustrated in Fig. 8, which presents the predicted function values along the diagonal of the input domain, specifically where $x_1 = x_2$. In the case of sparse observations (left plot), the Gaussian process model can only capture the first and second scales, but misses finer details. In contrast, the generalized Gaussian process model performs significantly better, closely aligning with the ground truth by accurately representing nearly all present scales. This trend continues with dense observations (right plot), where the Gaussian process model improves slightly by capturing up to the third scale but still lacks precision in representing finest-scale variations. The generalized Gaussian process model, however, consistently captures all four scales with high accuracy, demonstrating its superior ability to model complex multi-scale features in the data. This comparison clearly highlights the enhanced flexibility and accuracy of the generalized Gaussian process model compared to the Gaussian process model in handling multi-scale functions.

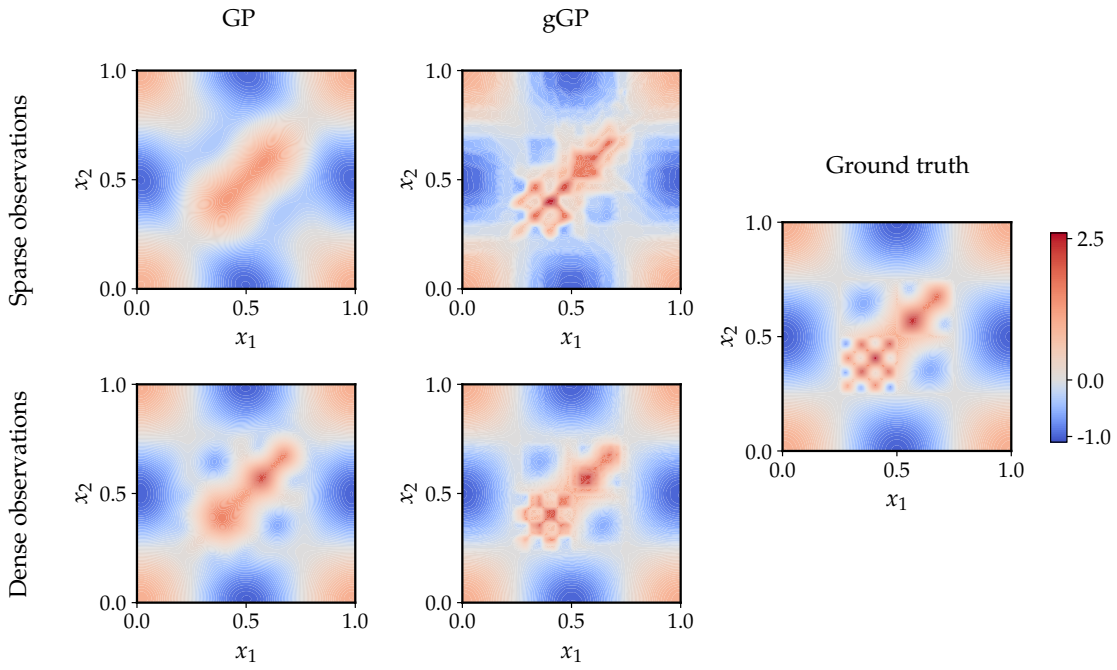


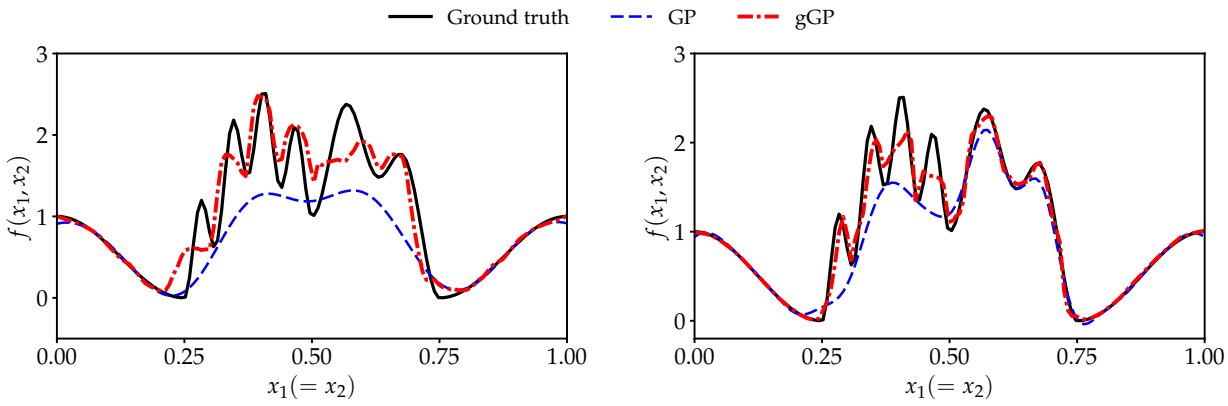
Figure 7: Comparison of regression performance for the two-dimensional multi-scale trigonometric function using Gaussian process (GP) and generalized Gaussian process (gGP) models. The top row displays the results with sparse observations on a 16×16 uniform grid, while the bottom row presents the results with dense observations on a 32×32 uniform grid. The GP captures, at best, the first two or three length scales in both scenarios, whereas the gGP consistently captures all four scales with higher precision.

3.2. Ensemble neural filter (EnNF) for sequential data assimilation

We numerically investigate the assimilation performance of the ensemble neural filter using two chaotic dynamical models: the Lorenz-63 and the Lorenz-96 models, which are commonly used to validate data assimilation algorithms [42]. Our results demonstrate that the ensemble neural filter can mimic the filtering performance of the ensemble Kalman filter when trained on data generated by the latter. Moreover, it exhibits excellent assimilation performance, even with an ensemble size as small as two, and maintains consistent performance across various small ensemble sizes, outperforming the ensemble Kalman filter method. The experimental setups for the Lorenz-63 and Lorenz-96 models primarily follow those described in the literature [43] and [44], respectively.

3.2.1. Assimilation experiment on the Lorenz-63 model

Dynamical model The Lorenz-63 model, introduced by Lorenz [45], is a simplified mathematical model for describing atmospheric convection dynamics. The state vector in this model consists of three variables, represented as $\mathbf{z}(t) = (z_1(t), z_2(t), z_3(t))$, whose dynamics are governed by the following set of three coupled



(a) Regression with sparse observations

(b) Regression with dense observations

Figure 8: Comparison of regression results for the two-dimensional multi-scale trigonometric function along the diagonal ($x_1 = x_2$) of the input domain using the Gaussian process (GP) and generalized Gaussian process (gGP): (a) predictions with sparse observations and (b) predictions with dense observations. The GP captures the first two scales with sparse observations and up to the third scale when using dense observations. In contrast, the gGP consistently captures all four scales, closely aligning with the ground truth.

ordinary differential equations (ODEs):

$$\begin{aligned} \frac{\mathrm{d}z_1}{\mathrm{d}t} &= \sigma(z_2 - z_1), \\ \frac{\mathrm{d}z_2}{\mathrm{d}t} &= z_1(\rho - z_3) - z_2, \\ \frac{\mathrm{d}z_3}{\mathrm{d}t} &= z_1 z_2 - \beta z_3, \end{aligned} \tag{18}$$

where $\beta = 8/3$, $\rho = 28$, and $\sigma = 10$ are fixed model parameters commonly used to produce chaotic solutions towards the well-known Lorenz attractor. This ODE system is integrated using a fourth-order explicit Runge–Kutta method, with a fixed step size of $\Delta t = 0.01$. For twin experiment testing, we set the baseline initial condition slightly deviated from the true initial condition to evaluate the performance of data assimilation methods in correcting state estimates under initial condition uncertainties.

Likelihood model The state of the Lorenz-63 system is observed directly every $\Delta t_{\text{obs}} = 0.5$ time units (i.e., every 50 integration steps). For any observation step k, the likelihood model is defined by:

$$\mathbf{d}(t_k) = \mathsf{H}\mathbf{z}^*(t_k) + \boldsymbol{\epsilon}(t_k), \tag{19}$$

where t_k is the k-th observation time given by $t_k = k\Delta t_{\text{obs}}$, $\mathbf{z}^*$ represents the true solution, $\mathbf{d}$ denotes the observation, and $\boldsymbol{\epsilon}$ represents the observational noise. In this experiment, $\mathsf{H} \in \mathbb{R}^{2\times 3}$ is a linear observation operator that selects the state variables z_1 and z_3 from the full state vector, and $\boldsymbol{\epsilon}$ is assumed to be Gaussian with zero mean and covariance $\mathbf{C}_{dd,i,i} = (0.1\,\mathbf{d}_i^* + 0.05)^2$ for $i \in \{1, 2\}$, where the synthetic observation $\mathbf{d}^* = \mathsf{H}\mathbf{z}^*$ and $\mathbf{d}_i^*$ refers to its i-th component.

Experiment setup Given the true initial condition of $z_1(0) = -8.5, z_2(0) = -7.0, z_3(0) = 27.0$, we generate the true solution and a sequence of synthetic observations. The baseline initial condition is set slightly off from the true initial condition at $z_1(0) = -8.0, z_2(0) = -9.0, z_3(0) = 28.0$. The initial prior ensemble, $\mathbf{Z}^{\mathrm{f}}(0)$, is generated by sampling from a Gaussian distribution with mean equal to the baseline initial condition and covariance $\mathbf{C}_{zz} = \mathrm{diag}(0.4, 2.0, 1.4)$.

With the initial prior ensemble generated, the dynamical model available, and both the synthetic observations and likelihood model defined, the ensemble Kalman filter is applied to correct the state estimates whenever an observation is made. Specifically, during each assimilation window, each realization $\mathbf{z}_i$ is independently propagated according to Eq. (18), and then updated based on the ensemble statistics and its corresponding perturbed observation $\mathbf{d}_i$, as described by Eq. (12). The updated ensemble then serves as the starting point for the next data assimilation window. We implement the ensemble Kalman filter using an ensemble size of $N = 50$ to ensure that the calculated ensemble error covariances closely approximate the true error covariances.

Data generation We conduct ten separate runs of the ensemble Kalman filter on the Lorenz-63 model to generate data for training the ensemble neural filter. Each run begins with a different initial prior ensemble and spans a training period of $T = 75$ time units that covers 150 assimilation windows. As such, this process generates 1,500 pairs of prior and posterior ensembles of full state vectors for training purposes. However, as stated above, we assume each state variable in a prior ensemble shares the same update operator, which allows its independent update without involving other state variables. Thus, we have 4,500 input-output data pairs for training ensemble neural filter, given the state vector has three variables. Specifically, the input is an ensemble consisting of three components: a prior state variable, predicted observation, and observation, i.e., $\left\{\left(z_{i,j}^{\mathrm{f}}, \mathsf{H}\mathbf{z}_i^{\mathrm{f}}, \mathbf{d}_i\right)\right\}_{i=1}^{50}$ for $j \in \{1, 2, 3\}$; the output is an ensemble of the corresponding posterior state variable, i.e., $\left\{z_{i,j}^{\mathrm{a}}\right\}_{i=1}^{50}$. With the training dataset, we train the ensemble neural filter on an NVIDIA V100 GPU for 2,000 epochs using the open-source machine learning framework PyTorch [46].

Assimilation result The trained ensemble neural filter exhibits excellent assimilation performance, even with the smallest ensemble size of $N = 2$, as shown in Fig. 9. This figure illustrates the dynamics of each state variable in the ensemble mean, $\overline{\mathbf{z}}(t) := \frac{1}{N}\sum_{i=1}^{N} \mathbf{z}_i(t)$, across two different time periods, each spanning 16 assimilation windows. The first period (left column) corresponds to the initial phase within the training period, while the second period (right column) falls within the forecasting period after the training period and represents temporal extrapolation at a later stage. It can be seen that, in both periods, the baseline dynamics of each state variable significantly deviates from the ground truth due to the minor initial condition deviations and the chaotic nature of the Lorenz-63 system. Despite the observations, the ensemble Kalman filter fails to correct the baseline state effectively; sometimes updating the state even further from

the observations or resulting in minimal updates. This is attributed to the small ensemble size, which results in erroneous approximations of the true covariances, an important component in the ensemble Kalman filter update equation. In contrast, the results from ensemble neural filter closely match the ground truths, as it bypasses the need for explicit covariance approximations.

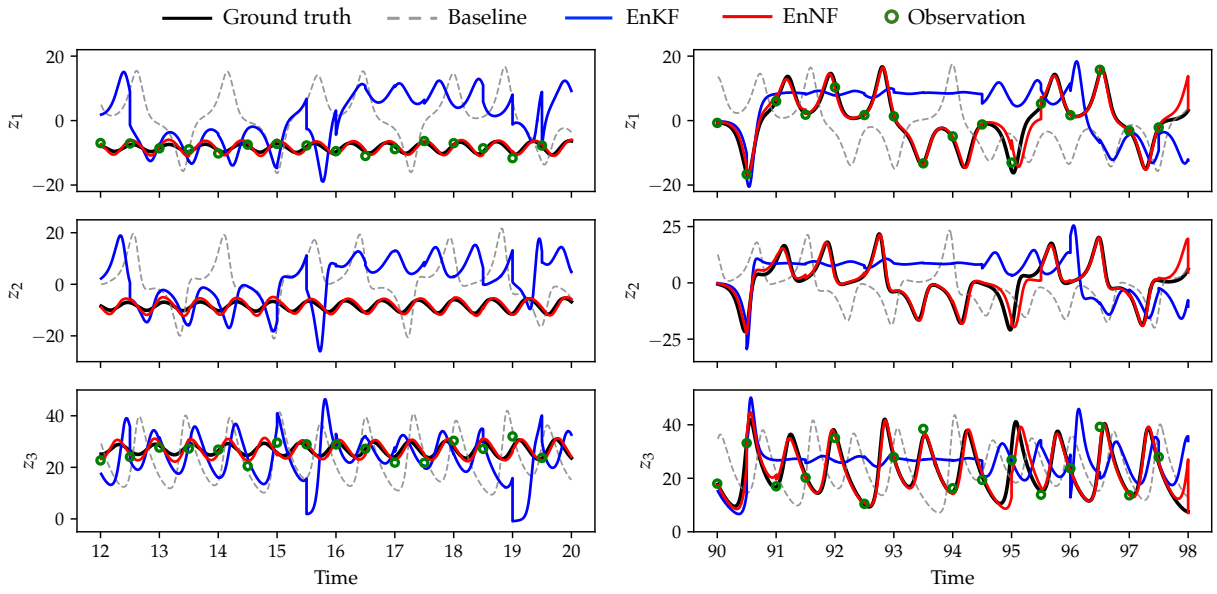


Figure 9: Comparison of state estimation for the Lorenz-63 model using ensemble Kalman filter (EnKF) and ensemble neural filter (EnNF) with the smallest ensemble size of $N = 2$. The left panels show the dynamics of three state variables, z_1, z_2, and z_3, over 16 initial assimilation windows, while the right panels continue the display over another 16 windows after the process has been going on for a while. Each panel compares the results of the baseline, EnKF and EnNF against the ground truth. The baseline result deviates significantly from the ground truth due to the initial condition deviation and the absence of data assimilation. The EnKF result shows limited improvement due to the small ensemble size. In contrast, the EnNF result is very close to the ground truth, demonstrating its superior assimilation performance. Note that the results for both EnKF and EnNF represent the behavior of the ensemble mean $\overline{\mathbf{z}}(t)$.

Moreover, the trained ensemble neural filter demonstrates consistent assimilation performance across various small ensemble sizes, as illustrated in Fig. 10. Specifically, Fig. 10(a) compares the model trajectories using the ensemble Kalman filter and ensemble neural filter against the ground truths across four different ensemble sizes: $N = 2, 4, 8$ and 16. It can be seen that, as the ensemble size increases, the trajectory from ensemble Kalman filter noticeably converges towards the ground truth, indicating improved assimilation performance with larger ensembles. In contrast, the trajectory of ensemble neural filter consistently aligns closely with the ground truth across all four ensemble sizes, highlighting its robustness under varying uncertainty conditions. This superior assimilation performance is further evidenced in Fig. 10(b), which shows the assimilation errors, represented by relative root mean square error (RMSE), for both methods across

eight ensemble sizes ranging from 2 to 16. The relative RMSE is computed following the equation:

$$\text{relative RMSE} = \sqrt{\frac{\sum_{k=1}^{N_a} \left\|\overline{\mathbf{z}}^{\mathrm{a}}(t_k) - \mathbf{z}^*(t_k)\right\|^2}{\sum_{k=1}^{N_a} \left\|\mathbf{z}^*(t_k)\right\|^2}}, \tag{20}$$

where N_a represents the total number of assimilation windows, $\|\cdot\|$ indicates the ℓ^2-norm, and $\overline{\mathbf{z}}^{\mathrm{a}}$ is the posterior ensemble mean obtained by either ensemble Kalman filter or ensemble neural filter. Here, the assimilation error is calculated over a span of 200 assimilation windows, including the initial 150 in the training period and the subsequent 50 in the extrapolation period. For each setting, defined by a specific assimilation method and ensemble size, the error represents the average metric from five runs with different initial priors, ensuring statistical robustness and reliability. It is clear that the assimilation errors from ensemble Kalman filter are initially high with very small ensembles but decrease rapidly as the ensemble size increases. In contrast, ensemble neural filter maintains lower values across all ensemble sizes, further demonstrating its superior assimilation performance. Notably, the errors from ensemble neural filter remain stable and low throughout the entire interval, highlighting its effectiveness and robustness in handling smaller ensembles compared to the ensemble Kalman filter. This characteristic makes ensemble neural filter particularly valuable in scenarios where computational resources are limited.

3.2.2. Assimilation experiment on the Lorenz-96 model

Dynamical model The Lorenz-96 model is a widely used dynamical model developed to study fundamental issues related to mid-latitude atmospheric and weather predictability [47]. Unlike the Lorenz-63 model, which describes the fluid convection between hot and cold plates using three variables, the Lorenz-96 model extends to higher dimensions, making it a popular testbed for data assimilation algorithms in a more complex yet controlled environment. The state vector in this model consists of m ($m \geq 4$) variables, represented as $\mathbf{z}(t) = (z_1(t), \ldots, z_m(t))$, whose dynamics are given by a set of nonlinear ODEs:

$$\frac{\mathrm{d}z_i}{\mathrm{d}t} = (z_{i+1} - z_{i-2})\, z_{i-1} - z_i + F, \quad i = 1, 2, \ldots, m, \tag{21}$$

with periodic boundary conditions $z_{-1} = z_{m-1}$, $z_0 = z_m$, and $z_{m+1} = z_1$. Here, m is the number of spatial points in the discretization, z_i denotes the state variable at the i-th location, and F represents a constant forcing parameter. In this experiment, we use $n = 24$ and $F = 8$, a commonly used configuration to induce fully chaotic dynamics. The well-defined ODE system is integrated using a fourth-order explicit Runge–Kutta method, with a fixed step size of $\Delta t = 0.01$.

Likelihood model The state of the Lorenz-96 system is observed directly every $\Delta t_{\mathrm{obs}} = 0.2$ time units (i.e., every 20 integration steps), considering its increased complexity compared to the Lorenz-63 model. The likelihood model follows the same form as that used for the Lorenz-63 system. Here, we recall it for

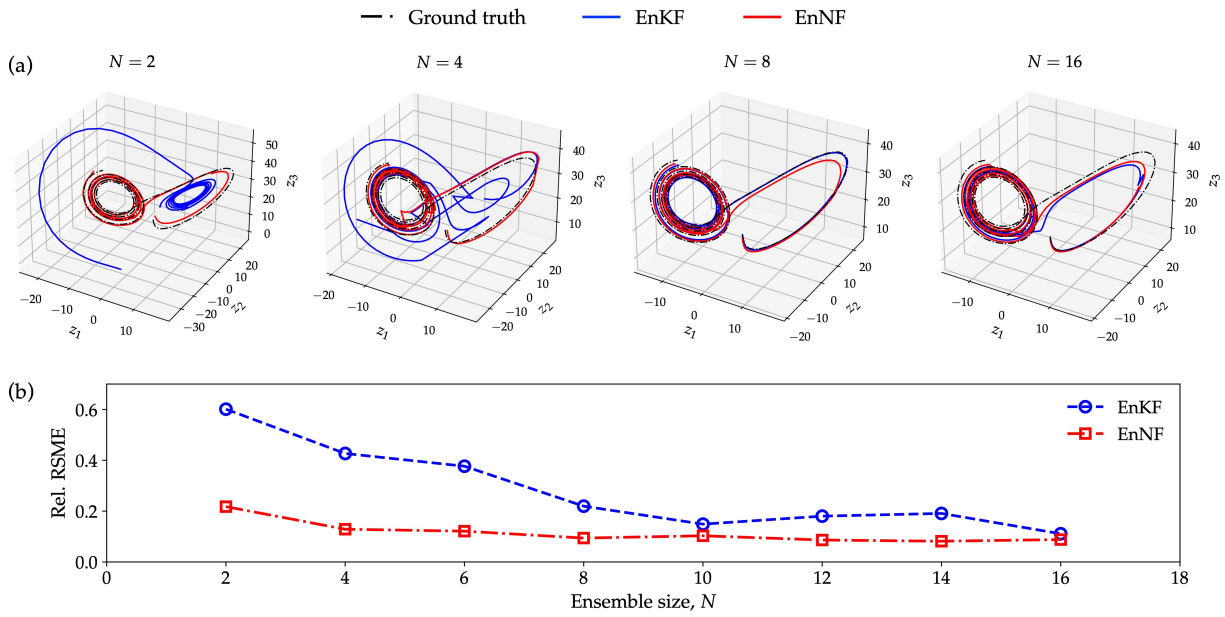


Figure 10: Comparison of assimilation performance of ensemble Kalman filter (EnKF) and ensemble neural filter (EnNF) for the Lorenz-63 model across various ensemble sizes. Panel (a) shows the ensemble mean trajectories using EnKF and EnNF under four ensemble sizes over 20 assimilation windows during the process. The trajectory from EnKF gradually converges towards the ground truth with increasing ensemble size, while the trajectory from EnNF consistently remains in close proximity to the ground truth. Panel (b) quantitatively compares the assimilation errors of both methods across eight ensemble sizes from 2 to 16. The EnNF consistently demonstrates lower errors compared to EnKF, highlighting its superior assimilation performance and robustness, especially with smaller ensemble sizes.

completeness:

$$\mathbf{d}(t_k) = \mathsf{H}\mathbf{z}^*(t_k) + \boldsymbol{\epsilon}(t_k).$$

For the Lorenz-96 model, the linear observation operator $\mathsf{H} \in \mathbb{R}^{8\times 24}$ selects the state variables at eight equidistant locations from the full state vector, specifically z_i with $i = 3, 6, \ldots, 24$. The Gaussian observational noise $\boldsymbol{\epsilon}$ has zero mean and covariance $\mathbf{C}_{dd,i,i} = (0.05\,\mathbf{d}_i^* + 0.1)^2$ for $i \in \{1, 2, \ldots, 8\}$.

Experiment setup To generate a valid initial condition, we start at $t = -5$ with the equilibrium state $z_i = F$ for $i = 1, \ldots, 24$, and introduce a small perturbation to the 20th state variable by setting $z_{20} = F + 0.01$. We then run the ODE integrator up to $t = 0$ and use the solution at $t = 0$ as the true initial condition. With this true initial condition, we generate the true solution and a sequence of synthetic observations. For twin experiment testing, the baseline initial condition is set slightly off from the true initial condition by adding a random noise drawn from $\mathcal{N}(\mathbf{0}, \mathbf{I}_{24})$. The initial prior ensemble, $\mathbf{Z}^{\mathrm{f}}(0)$, is then generated by sampling from a Gaussian distribution with mean equal to the baseline initial condition and covariance $\mathbf{C}_{zz} = \mathbf{I}_{24}$.

With the initial prior ensemble generated, the dynamical model available, and both the synthetic ob-

servations and likelihood model specified, we apply the ensemble Kalman filter for state estimate under initial condition uncertainties, following a similar procedure as described for the Lorenz-63 model. For the Lorenz-96 model, we use an ensemble size of $N = 100$, given its significantly higher dimensionality. Note that this ensemble size does not result in optimal assimilation performance according to the literature [42]; however, we use this size for proof of concept and computational manageability.

Data generation We conduct 30 separate runs of the ensemble Kalman filter on the Lorenz-96 model to generate data for training ensemble neural filter, with each run starting with a different initial prior ensemble and spanning a training period of $T = 30$ time units that covers 150 assimilation windows. This process generates 4,500 pairs of prior and posterior ensembles of full state vectors for training purposes. Given that the model state consists of 24 variables, and assuming each prior state variable shares the same update operator, we have 108,000 training data pairs, with the input and output represented as $\left\{\left(z_{i,j}^{\mathrm{f}}, \mathsf{H}\mathbf{z}_i^{\mathrm{f}}, \mathbf{d}_i\right)\right\}_{i=1}^{100}$ and $\left\{z_{i,j}^{\mathrm{a}}\right\}_{i=1}^{100}$ for $j \in \{1, 2, \ldots, 24\}$, respectively.

However, unlike the training strategy used for the Lorenz-63 case, in this case, the ensemble neural filter is initially trained using the entire dataset for 2000 epochs and then fine-tuned for each state variable using their respective sub-datasets for a few hundred epochs. This adjustment is necessary because an ensemble neural filter with the same architecture, in terms of the number of layers and neurons, as used in the Lorenz-63 case may not sufficiently handle the filtering complexities here. Alternatively, training a larger-scale ensemble neural filter with more layers and neurons may directly address this issue.

Assimilation result The trained ensemble neural filter also demonstrates good assimilation performance on this more complex model, even with the smallest ensemble size $N = 2$, as shown in Fig. 11. Specifically, we compare the dynamics of three state variables, z_6, z_{12}, and z_{20}, from ensemble Kalman filter and ensemble neural filter against the ground truth over two distinct time periods, with the first period (left column) representing an initial phase within the training period and the other (right column) temporal extrapolation at a later stage, both covering 25 assimilation windows. Ensemble neural filter performs reasonably well under this extreme small-ensemble condition, while ensemble Kalman filter falls short. The ensemble mean from ensemble neural filter virtually matches the ground truth during the initial period and remains near the true solution in the forecasting period despite some discrepancies due to the limited training dataset. By employing a larger dataset collected over a sufficiently long period, the trained ensemble neural filter could potentially enhance its assimilation performance for temporal extrapolation.

Additionally, the trained ensemble neural filter exhibits robust and consistent assimilation performance across various small ensemble sizes, which is illustrated in Fig. 12. We compare the results of ensemble Kalman filter and ensemble neural filter against the ground truth using both qualitative and quantitative methods in Fig.12(a) and (b), respectively. Fig.12(a) shows the traveling waves in the Lorenz-96 model within

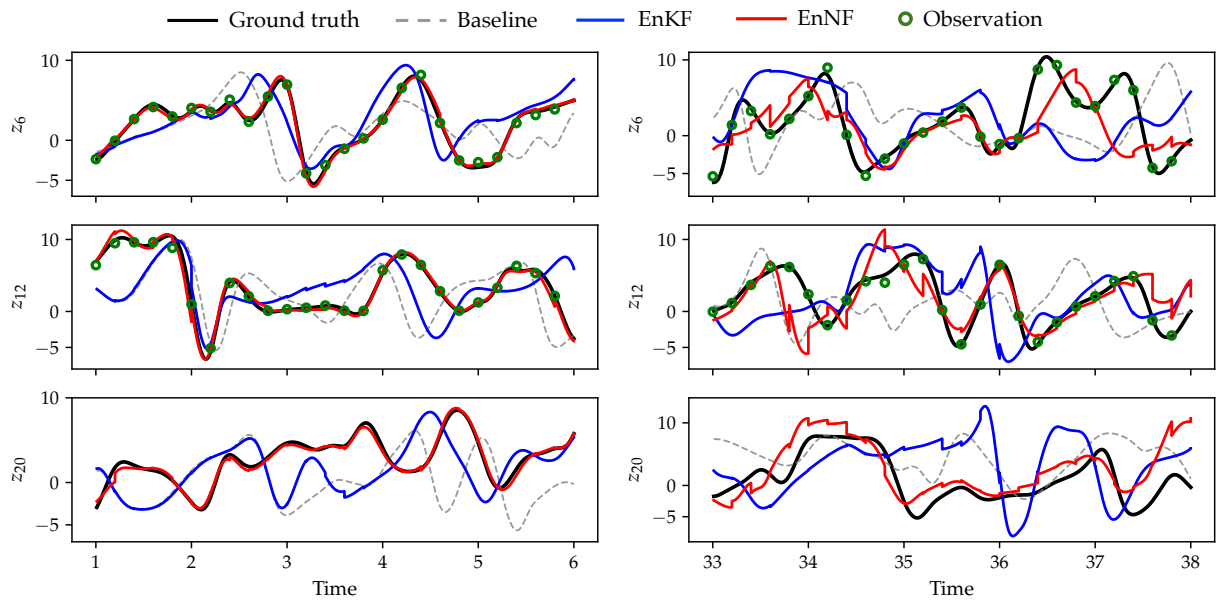


Figure 11: Comparison of state estimation for the Lorenz-96 model using ensemble Kalman filter (EnKF) and ensemble neural filter (EnNF) with the smallest ensemble size of $N = 2$. The left panels show the dynamics of three state variables, z_6, z_{12}, and z_{20}, over 25 initial assimilation windows, while the right panels continue the display over another 25 windows after the process has been going on for a while. Each panel compares the results of the baseline, EnKF and EnNF against the ground truth. The baseline result deviates evidently from the ground truth due to the initial condition deviation and the absence of data assimilation. The EnKF result shows limited improvement due to the small ensemble size. In contrast, the EnNF result is significantly closer to the ground truth, demonstrating its superior assimilation performance under this extremely small ensemble size. Note that the results for both EnKF and EnNF represent the dynamics of the three state variables from the ensemble mean $\overline{\mathbf{z}}(t)$.

the initial 50 assimilation windows by plotting the ensemble mean values (with 24 components) over time. The traveling waves from ensemble Kalman filter differ significantly from the ground truth for three ensemble sizes ($N = 8, 16$, and 32), with the $N = 16$ case showing clear extreme estimates. In contrast, the traveling waves from ensemble neural filter closely match the ground truth for all three ensemble sizes, demonstrating superior assimilation performance. Fig.12(b) further illustrates this by comparing the assimilation errors for both methods across eight ensemble sizes ranging from 2 to 64. As with the Lorenz-63 model, the error is calculated over 200 assimilation windows—150 during the training period followed by 50 during the extrapolation period. Each assimilation error is averaged over five runs with different initial priors to ensure statistical robustness and reliability. For smaller ensemble sizes ($N \leq 16$), the assimilation errors for ensemble Kalman filter consistently exceed 100%. Beyond this point, further increasing the ensemble size gradually reduces assimilation errors to a lower level. In contrast, the assimilation errors for ensemble neural filter remain stable and low across the entire range, highlighting its superior effectiveness and robustness in handling small ensembles compared to ensemble Kalman filter.

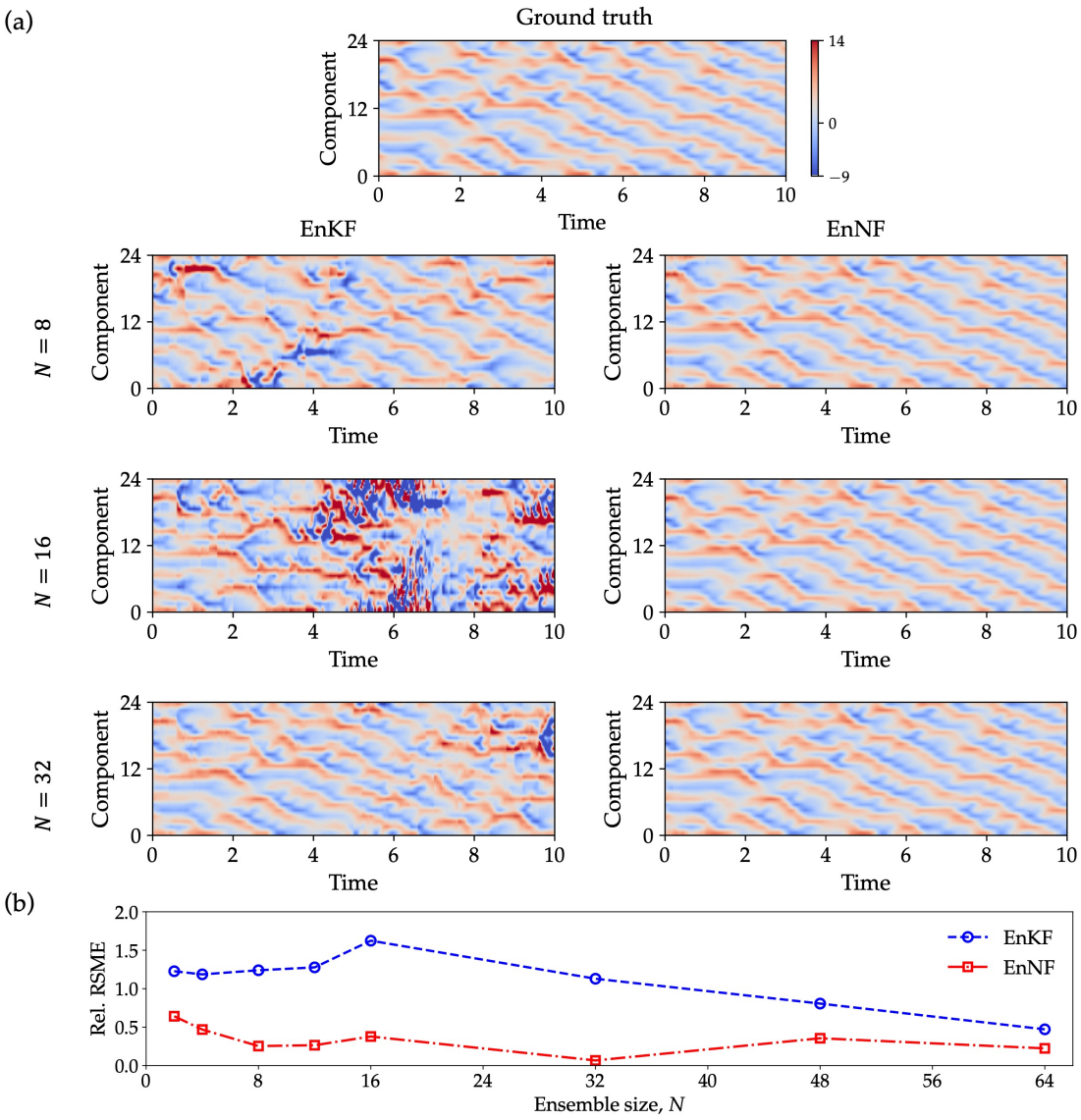


Figure 12: Comparison of assimilation performance of ensemble Kalman filter (EnKF) and ensemble neural filter (EnNF) for the Lorenz-96 model across various ensemble sizes. Panel (a) shows the traveling waves, i.e., ensemble mean of the system over time, from EnKF and EnNF under three ensemble sizes over the initial 50 assimilation windows. The waves from EnKF exhibit clear deviations from the ground truth, while those from EnNF show visual identity. Panel (b) quantitatively compares the assimilation errors of both methods across eight ensemble sizes ranging from 2 to 64. The EnNF consistently demonstrates lower errors, highlighting its superior assimilation performance and robustness for small ensemble sizes in the Lorenz-96 model.

4. Conclusion

Approximate Bayesian inference is typically approached through deterministic and stochastic methods, each with significant limitations: deterministic methods often rely on predefined functional structures, while stochastic methods require large sample sizes for reliability. To address these challenges, we introduce the

BI-EqNO, an equivariant neural operator framework for enhancing various approximate Bayesian inference methods. BI-EqNO transforms prior distributions into corresponding posterior distributions through data-driven training and ensures necessary permutation equivariance between their representations.

We have rigorously evaluated this framework in two important scenarios: regression and sequential data assimilation, demonstrating its efficacy and broad applicability. In regression, the generalized Gaussian process (gGP) outperforms traditional Gaussian process in learning functions characterized by abrupt changes and multiple scales. This enhanced performance is attributable to its ability to learn covariances directly from the data, free from the restriction of predefined kernel structures. In sequential data assimilation, the ensemble neural filter (EnNF) is trained to reproduce the filtering performance of the ensemble Kalman filter and demonstrates superior performance under small-ensemble settings in both Lorenz-63 and Lorenz-96 models.

Despite these promising results, further investigation is required to apply both the generalized Gaussian process and ensemble neural filter to complex, real-world scenarios. For the generalized Gaussian process, integrating essential physical constraints, such as frame independence, is crucial when modeling physical systems. Adapting such physics-informed generalized Gaussian process to real-world challenges, particularly in large-scale geophysical contexts with asymmetric and non-stationary behaviors, necessitates systematic exploration. Additionally, training the ensemble neural filter in high-dimensional systems presents substantial challenges due to significant memory demands. Advancements using modern machine learning architectures, such as Transformers [48, 49], are essential for enhancing its scalability and performance.

Acknowledgments

XHZ is supported by the U.S. Air Force under agreement number FA865019-2-2204. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. ZRL and HX are funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2075 – 390740016. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech). We appreciate the valuable discussions with members of SimTech, particularly Wolfgang Nowak, Steffen Staab, Tim Schneider, and Flex Frizen.

Appendix A. Mathematical constraints of generalized Gaussian process

In designing the generalized Gaussian process, it is essential to preserve the mathematical constraints inherent in the conditional posterior distribution of Gaussian process regression. For completeness, we restate the mean and covariance:

$$\begin{aligned} \mathsf{m}_{\mathcal{D}}(X^*) &= \mathsf{m}(X^*) + \mathsf{K}(X^*, X)\left[\mathsf{K}(X, X) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1}\left[Y - \mathsf{m}(X)\right], \\ \mathsf{K}_{\mathcal{D}}(X^*, X^*) &= \mathsf{K}(X^*, X^*) - \mathsf{K}(X^*, X)\left[\mathsf{K}(X, X) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1}\mathsf{K}(X, X^*). \end{aligned} \tag{A.1}$$

BI-EqNO, an equivariant neural operator framework for enhancing various approximate Bayesian inference methods. BI-EqNO transforms prior distributions into corresponding posterior distributions through data-driven training and ensures necessary permutation equivariance between their representations.

We have rigorously evaluated this framework in two important scenarios: regression and sequential data assimilation, demonstrating its efficacy and broad applicability. In regression, the generalized Gaussian process (gGP) outperforms traditional Gaussian process in learning functions characterized by abrupt changes and multiple scales. This enhanced performance is attributable to its ability to learn covariances directly from the data, free from the restriction of predefined kernel structures. In sequential data assimilation, the ensemble neural filter (EnNF) is trained to reproduce the filtering performance of the ensemble Kalman filter and demonstrates superior performance under small-ensemble settings in both Lorenz-63 and Lorenz-96 models.

Despite these promising results, further investigation is required to apply both the generalized Gaussian process and ensemble neural filter to complex, real-world scenarios. For the generalized Gaussian process, integrating essential physical constraints, such as frame independence, is crucial when modeling physical systems. Adapting such physics-informed generalized Gaussian process to real-world challenges, particularly in large-scale geophysical contexts with asymmetric and non-stationary behaviors, necessitates systematic exploration. Additionally, training the ensemble neural filter in high-dimensional systems presents substantial challenges due to significant memory demands. Advancements using modern machine learning architectures, such as Transformers [48, 49], are essential for enhancing its scalability and performance.

Acknowledgments

XHZ is supported by the U.S. Air Force under agreement number FA865019-2-2204. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. ZRL and HX are funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2075 – 390740016. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech). We appreciate the valuable discussions with members of SimTech, particularly Wolfgang Nowak, Steffen Staab, Tim Schneider, and Flex Frizen.

Appendix A. Mathematical constraints of generalized Gaussian process

In designing the generalized Gaussian process, it is essential to preserve the mathematical constraints inherent in the conditional posterior distribution of Gaussian process regression. For completeness, we restate the mean and covariance:

$$\begin{aligned} \mathsf{m}_{\mathcal{D}}(X^*) &= \mathsf{m}(X^*) + \mathsf{K}(X^*, X)\left[\mathsf{K}(X, X) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1}\left[Y - \mathsf{m}(X)\right], \\ \mathsf{K}_{\mathcal{D}}(X^*, X^*) &= \mathsf{K}(X^*, X^*) - \mathsf{K}(X^*, X)\left[\mathsf{K}(X, X) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1}\mathsf{K}(X, X^*). \end{aligned} \tag{A.1}$$

The equations suggest that the mean vector and covariance matrix are both permutation invariant to the observation data (X, Y) and permutation equivariant to the test data points X^*.

Permutation invariance to observation data To demonstrate that the mean vector $\mathsf{m}_{\mathcal{D}}(X^*)$ is invariant under permutations of the observation data, consider a permutation $\mathcal{P}$ of the observation data. Let the reordered input and target values be $X' = \mathcal{P}X$ and $Y' = \mathcal{P}Y$. The covariance matrix and the cross-covariance with the test points will transform as:

$$\mathsf{K}(X', X') = \mathcal{P}\mathsf{K}(X, X)\mathcal{P}^\top \quad \text{and} \quad \mathsf{K}(X^*, X') = \mathsf{K}(X^*, X)\mathcal{P}^\top. \tag{A.2}$$

The posterior mean vector under this permutation is given by:

$$\mathsf{m}'_{\mathcal{D}}(X^*) = \mathsf{m}(X^*) + \mathsf{K}(X^*, X') \left[\mathsf{K}(X', X') + \sigma_\epsilon^2 \mathbf{I}\right]^{-1} \left[Y' - \mathsf{m}(X')\right]. \tag{A.3}$$

Substituting the transformed variables:

$$\mathsf{m}'_{\mathcal{D}}(X^*) = \mathsf{m}(X^*) + \mathsf{K}(X^*, X)\mathcal{P}^\top \left[\mathcal{P}\mathsf{K}(X, X)\mathcal{P}^\top + \sigma_\epsilon^2 \mathbf{I}\right]^{-1} \left[\mathcal{P}Y - \mathsf{m}(\mathcal{P}X)\right]. \tag{A.4}$$

Using the orthogonality of permutation matrices ($\mathcal{P}^{-1} = \mathcal{P}^\top$):

$$\mathsf{m}'_{\mathcal{D}}(X^*) = \mathsf{m}(X^*) + \mathsf{K}(X^*, X)\mathcal{P}^\top\mathcal{P} \left[\mathsf{K}(X, X) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1} \mathcal{P}^\top\mathcal{P} \left[Y - \mathsf{m}(X)\right]. \tag{A.5}$$

Substituting $\mathcal{P}^\top\mathcal{P} = \mathbf{I}$ into the expression:

$$\mathsf{m}'_{\mathcal{D}}(X^*) = \mathsf{m}(X^*) + \mathsf{K}(X^*, X) \left[\mathsf{K}(X, X) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1} \left[Y - \mathsf{m}(X)\right]. \tag{A.6}$$

Thus, $\mathsf{m}'_{\mathcal{D}}(X^*) = \mathsf{m}_{\mathcal{D}}(X^*)$. This confirms that the posterior mean vector is invariant under permutations of the observation data.

Permutation equivariance to test points To demonstrate that the mean vector $\mathsf{m}_{\mathcal{D}}(X^*)$ is equivariant under permutations of the test points, consider a permutation $\mathcal{Q}$ of the test points. Let the reordered test points be $X^{*\prime} = \mathcal{Q}X^*$. The cross-covariance with the observation points will transform as:

$$\mathsf{K}(X^{*\prime}, X) = \mathcal{Q}\mathsf{K}(X^*, X). \tag{A.7}$$

The posterior mean vector under this permutation is given by:

$$\mathsf{m}_{\mathcal{D}}(X^{*\prime}) = \mathsf{m}(X^{*\prime}) + \mathsf{K}(X^{*\prime}, X) \left[\mathsf{K}(X, X) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1} \left[Y - \mathsf{m}(X)\right]. \tag{A.8}$$

Substituting the transformed variables:

$$\mathsf{m}_{\mathcal{D}}(X^{*\prime}) = \mathsf{m}(\mathcal{Q}X^*) + \mathcal{Q}\mathsf{K}(X^*, X) \left[\mathsf{K}(X, X) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1} \left[Y - \mathsf{m}(X)\right]. \tag{A.9}$$

Since $\mathsf{m}(\mathcal{Q}X^*)$ is the reordered prior mean vector, and $\mathcal{Q}\mathsf{K}(X^*, X)$ represents the reordered cross-covariance matrix:

$$\begin{aligned}\mathsf{m}_{\mathcal{D}}(X^{*\prime}) &= \mathcal{Q}\mathsf{m}(X^*) + \mathcal{Q}\mathsf{K}(X^*, X)\left[\mathsf{K}(X, X) + \sigma_\epsilon^2\mathbf{I}\right]^{-1}[Y - \mathsf{m}(X)] \\ &= \mathcal{Q}\left[\mathsf{m}(X^*) + \mathsf{K}(X^*, X)\left[\mathsf{K}(X, X) + \sigma_\epsilon^2\mathbf{I}\right]^{-1}[Y - \mathsf{m}(X)]\right]\end{aligned} \tag{A.10}$$

Thus, $\mathsf{m}_{\mathcal{D}}(X^{*\prime}) = \mathcal{Q}\mathsf{m}_{\mathcal{D}}(X^*)$. This demonstrates that the posterior mean vector is equivariant with respect to the permutation $\mathcal{Q}$ of the test points.

Similarly, it can be shown that the covariance matrix $\mathsf{K}_{\mathcal{D}}(X^*, X^*)$ is permutation invariant to the observation data and permutation equivariant to the test data points. This can be demonstrated as follows:

(1) Under reordered observations, $X' = \mathcal{P}X$ and $Y' = \mathcal{P}Y$, the covariance matrix remains invariant:

$$\mathsf{K}'_{\mathcal{D}}(X^*, X^*) = \mathsf{K}_{\mathcal{D}}(X^*, X^*). \tag{A.11}$$

(2) Under reordered test points, $X^{*\prime} = \mathcal{Q}X^*$, the covariance matrix transforms equivariantly:

$$\mathsf{K}_{\mathcal{D}}(X^{*\prime}, X^{*\prime}) = \mathcal{Q}\mathsf{K}_{\mathcal{D}}(X^*, X^*)\mathcal{Q}^\top. \tag{A.12}$$

Appendix B. Mathematical properties of ensemble neural filter

In designing the ensemble neural filter (EnNF), we preserve two fundamental properties from the ensemble Kalman filter (EnKF) to ensure efficient training and application:

1. Uniform update operation for each state variable.

2. The update relies solely on information from the observation space.

In the ensemble Kalman filter framework, updated ensemble realizations are confined to the subspace spanned by the prior ensemble [22]. Specifically, an updated ensemble realization is a linear combination of prior ones, expressed in matrix form as:

$$\mathbf{Z}^{\mathrm{a}} = \mathbf{Z}^{\mathrm{f}} + \mathbf{Z}^{\mathrm{f}}\mathbf{W}, \tag{B.1}$$

where $\mathbf{Z}^{\mathrm{f}} = \left(\mathbf{z}_1^{\mathrm{f}}, \ldots, \mathbf{z}_N^{\mathrm{f}}\right) \in \mathbb{R}^{n\times N}$ denotes the prior ensemble, $\mathbf{Z}^{\mathrm{a}} = (\mathbf{z}_1^{\mathrm{a}}, \ldots, \mathbf{z}_N^{\mathrm{a}}) \in \mathbb{R}^{n\times N}$ represents the posterior ensemble, and $\mathbf{W} \in \mathbb{R}^{N\times N}$ is the weight matrix.

Consider the j-th state variable across all realizations. We define the vector of the j-th state variable from the prior ensemble as:

$$\mathbf{z}_{:,j}^{\mathrm{f}} = \left(z_{1,j}^{\mathrm{f}}, z_{2,j}^{\mathrm{f}}, \ldots, z_{N,j}^{\mathrm{f}}\right), \tag{B.2}$$

and similarly for the posterior ensemble:

$$\mathbf{z}_{:,j}^{\mathrm{a}} = \left(z_{1,j}^{\mathrm{a}}, z_{2,j}^{\mathrm{a}}, \ldots, z_{N,j}^{\mathrm{a}}\right). \tag{B.3}$$

From the linear update scheme (B.1), the update for the j-th state variable across all realizations is:

$$\mathbf{z}^{\mathrm{a}}_{:,j} = \mathbf{z}^{\mathrm{f}}_{:,j} + \mathbf{z}^{\mathrm{f}}_{:,j}\mathbf{W} \quad \text{for} \quad j \in \{1, \ldots, n\}, \tag{B.4}$$

indicating that each state variable follows the same update mechanism.

The weight matrix $\mathbf{W}$ is determined by the following equation derived from Eq. (12):

$$\mathbf{W} = \mathbf{Y}^{\top} \left(\overline{\mathbf{C}}_{yy} + \overline{\mathbf{C}}_{dd}\right)^{-1} \left(\mathbf{D} - \mathcal{H}(\mathbf{Z}^{\mathrm{f}})\right), \tag{B.5}$$

where $\mathbf{Y}$ represent the normalized anomaly matrix of the predicted observations $\mathcal{H}(\mathbf{Z}^{\mathrm{f}})$:

$$\mathbf{Y} = \frac{1}{N-1}\left(\mathcal{H}(\mathbf{z}^{\mathrm{f}}_1) - \overline{\mathcal{H}(\mathbf{Z}^{\mathrm{f}})}, \ldots, \mathcal{H}(\mathbf{z}^{\mathrm{f}}_N) - \overline{\mathcal{H}(\mathbf{Z}^{\mathrm{f}})}\right), \tag{B.6}$$

with

$$\overline{\mathcal{H}(\mathbf{Z}^{\mathrm{f}})} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{H}(\mathbf{z}^{\mathrm{f}}_i). \tag{B.7}$$

Therefore, the weight matrix $\mathbf{W}$ is solely derived from the observations and the predicted observations, utilizing only the information from the observation space.

In applying an equivariant neural operator for the update, denoted as $\mathcal{F}_{\mathrm{E}} : \mathbf{Z}^{\mathrm{f}} \mapsto \mathbf{Z}^{\mathrm{a}}$, the neural operator $\mathcal{F}_{\mathrm{E}}$ also adheres to these two principles. Thus, we have

$$\mathcal{F}_{\mathrm{E}} : \mathbf{z}^{\mathrm{f}}_{:,j} \mapsto \mathbf{z}^{\mathrm{a}}_{:,j} \quad \text{for} \quad j \in \{1, \ldots, n\}. \tag{B.8}$$

The operator $\mathcal{F}_{\mathrm{E}}$ exclusively depends on the observations $\{\mathbf{d}_i\}_{i=1}^{N}$ and the predicted observations $\{\mathcal{H}(\mathbf{z}^{\mathrm{f}}_i)\}_{i=1}^{N}$. Consequently, the ensemble neural filter defines a permutation equivariant mapping:

$$\mathcal{F}_{\mathrm{E}} : \left\{\left(z^{\mathrm{f}}_{i,j}, \mathcal{H}(\mathbf{z}^{\mathrm{f}}_i), \mathbf{d}_i\right)\right\}_{i=1}^{N} \mapsto \left\{z^{\mathrm{a}}_{i,j}\right\}_{i=1}^{N} \tag{B.9}$$

for all $j \in \{1, \ldots, n\}$.

References

[1] K. Duraisamy, G. Iaccarino, H. Xiao, Turbulence modeling in the age of data, Annual review of fluid mechanics 51 (2019) 357–377.

[2] H. Xiao, J.-L. Wu, J.-X. Wang, R. Sun, C. Roy, Quantifying and reducing model-form uncertainties in Reynolds-averaged Navier–Stokes simulations: A data-driven, physics-informed Bayesian approach, Journal of Computational Physics 324 (2016) 115–136.

[3] W. N. Edeling, P. Cinnella, R. P. Dwight, H. Bijl, Bayesian estimates of parameter variability in the k–ε turbulence model, Journal of Computational Physics 258 (2014) 73–94.

[4] X.-L. Zhang, H. Xiao, X. Luo, G. He, Ensemble Kalman method for learning turbulence models from indirect observation data, Journal of Fluid Mechanics 949 (2022) A26.

[5] I. M. Navon, Data assimilation for numerical weather prediction: A review, Data Assimilation for Atmospheric, Oceanic and Hydrologic Applications (2009) 21–65.

[6] A. Carrassi, M. Bocquet, L. Bertino, G. Evensen, Data assimilation in the geosciences: An overview of methods, issues, and perspectives, Wiley Interdisciplinary Reviews: Climate Change 9 (5) (2018) e535.

[7] L. M. Berliner, J. A. Royle, C. K. Wikle, R. F. Milliff, Bayesian methods in the atmospheric sciences, Bayesian Statistics 6 (1999) 83–100.

[8] J. G. Richens, C. M. Lee, S. Johri, Improving the accuracy of medical diagnosis with causal machine learning, Nature Communications 11 (1) (2020) 3923.

[9] R. Collins, N. Fenton, Bayesian network modelling for early diagnosis and prediction of Endometriosis, medRxiv (2020) 2020–11.

[10] D. J. MacKay, Information theory, inference and learning algorithms, Cambridge University Press, 2003.

[11] C. E. Rasmussen, Gaussian processes in machine learning, in: Summer school on machine learning, Springer, 2003, pp. 63–71.

[12] T. P. Minka, A family of algorithms for approximate Bayesian inference, Ph.D. thesis, Massachusetts Institute of Technology (2001).

[13] K. P. Murphy, Probabilistic machine learning: an introduction, MIT press, 2022.

[14] N. J. Gordon, D. J. Salmond, A. F. Smith, Novel approach to nonlinear/non-Gaussian Bayesian state estimation, in: IEE proceedings F (radar and signal processing), Vol. 140, IET, 1993, pp. 107–113.

[15] G. Evensen, Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics, Journal of Geophysical Research: Oceans 99 (C5) (1994) 10143–10162.

[16] X.-H. Zhou, H. Wang, J. McClure, C. Chen, H. Xiao, Inference of relative permeability curves in reservoir rocks with ensemble Kalman method, The European Physical Journal E 46 (6) (2023) 44.

[17] A. Pensoneault, W. F. Krajewski, N. Velásquez, X. Zhu, R. Mantilla, Ensemble Kalman inversion for upstream parameter estimation and indirect streamflow correction: A simulation study, Advances in Water Resources 181 (2023) 104545.

[18] P. Bauer, A. Thorpe, G. Brunet, The quiet revolution of numerical weather prediction, Nature 525 (7567) (2015) 47–55.

[19] E. P. Chassignet, H. E. Hurlburt, E. J. Metzger, O. M. Smedstad, J. A. Cummings, G. R. Halliwell, R. Bleck, R. Baraille, A. J. Wallcraft, C. Lozano, et al., US GODAE: global ocean prediction with the Hybrid Coordinate Ocean Model (HYCOM), Oceanography 22 (2) (2009) 64–75.

[20] A. Chattopadhyay, E. Nabizadeh, E. Bach, P. Hassanzadeh, Deep learning-enhanced ensemble-based data assimilation for high-dimensional nonlinear dynamical systems, Journal of Computational Physics 477 (2023) 111918.

[21] C.-A. Xia, J. Li, M. Riva, X. Luo, A. Guadagnini, Characterization of conductivity fields through iterative ensemble smoother and improved correlation-based adaptive localization, Journal of Hydrology 634 (2024) 131054.

[22] G. Evensen, F. C. Vossepoel, P. J. Van Leeuwen, Data assimilation fundamentals: A unified formulation of the state and parameter estimation problem, Springer Nature, 2022.

[23] A. Damianou, N. D. Lawrence, Deep Gaussian processes, in: Artificial intelligence and statistics, PMLR, 2013, pp. 207–215.

[24] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, J. Sohl-Dickstein, Deep neural networks as Gaussian processes, arXiv preprint arXiv:1711.00165 (2017).

[25] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, S. A. Eslami, Conditional neural processes, in: International conference on machine learning, PMLR, 2018, pp. 1704–1713.

[26] J. Zhang, C. Cao, T. Nan, L. Ju, H. Zhou, L. Zeng, A novel deep learning approach for data assimilation of complex hydrological systems, Water Resources Research 60 (2) (2024) e2023WR035389.

[27] M. Fan, Y. Bai, L. Wang, L. Ding, Combining a fully connected neural network with an ensemble Kalman filter to emulate a dynamic model in data assimilation, IEEE Access 9 (2021) 144952–144964.

[28] J. Brajard, A. Carrassi, M. Bocquet, L. Bertino, Combining data assimilation and machine learning to emulate a dynamical model from sparse and noisy observations: A case study with the Lorenz 96 model, Journal of computational science 44 (2020) 101171.

[29] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nature machine intelligence 3 (3) (2021) 218–229.

[30] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895 (2020).

[31] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Learning maps between function spaces with applications to PDEs, Journal of Machine Learning Research 24 (89) (2023) 1–97.

[32] X.-H. Zhou, J. Han, H. Xiao, Frame-independent vector-cloud neural network for nonlocal constitutive modeling on arbitrary grids, Computer Methods in Applied Mechanics and Engineering 388 (2022) 114211.

[33] J. Han, X.-H. Zhou, H. Xiao, An equivariant neural operator for developing nonlocal tensorial constitutive models, Journal of Computational Physics 488 (2023) 112243.

[34] Z. Li, N. Kovachki, C. Choy, B. Li, J. Kossaifi, S. Otta, M. A. Nabian, M. Stadler, C. Hundt, K. Azizzadenesheli, et al., Geometry-informed neural operator for large-scale 3D PDEs, Advances in Neural Information Processing Systems 36 (2024).

[35] M. Elrefaie, F. Morar, A. Dai, F. Ahmed, Drivaernet++: A large-scale multimodal car dataset with computational fluid dynamics simulations and deep learning benchmarks, arXiv preprint arXiv:2406.09624 (2024).

[36] X.-H. Zhou, J. Han, M. I. Zafar, C. J. Roy, H. Xiao, Neural operator-based super-fidelity: A warm-start approach for accelerating steady-state simulations, arXiv preprint arXiv:2312.11842 (2023).

[37] K. Azizzadenesheli, N. Kovachki, Z. Li, M. Liu-Schiaffini, J. Kossaifi, A. Anandkumar, Neural operators for accelerating scientific simulations and design, Nature Reviews Physics (2024) 1–9.

[38] M. I. Zafar, J. Han, X.-H. Zhou, H. Xiao, Frame invariance and scalability of neural operators for partial differential equations, arXiv preprint arXiv:2112.14769 (2021).

[39] B. Chen, C. Wang, W. Li, H. Fu, A hybrid Decoder-DeepONet operator regression framework for unaligned observation data, Physics of Fluids 36 (2) (2024).

[40] J. Han, Y. Li, L. Lin, J. Lu, J. Zhang, L. Zhang, Universal approximation of symmetric and anti-symmetric functions, arXiv preprint arXiv:1912.01765 (2019).

[41] C. K. Williams, C. E. Rasmussen, Gaussian processes for machine learning, Vol. 2, MIT press Cambridge, MA, 2006.

[42] A. Spantini, R. Baptista, Y. Marzouk, Coupling techniques for nonlinear ensemble filtering, SIAM Review 64 (4) (2022) 921–953.

[43] C. A. Michelén Ströfer, X.-L. Zhang, H. Xiao, DAFI: An open-source framework for ensemble-based data assimilation and field inversion, Communications in Computational Physics 29 (5) (2021) 1583–1622.

[44] S. E. Ahmed, S. Pawar, O. San, PyDA: A hands-on introduction to dynamical data assimilation with Python, Fluids 5 (4) (2020) 225.

[45] E. N. Lorenz, Deterministic nonperiodic flow, Journal of atmospheric sciences 20 (2) (1963) 130–141.

[46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems 32 (2019).

[47] E. N. Lorenz, Predictability: A problem partly solved, in: Proc. Seminar on predictability, Vol. 1, Reading, 1996.

[48] G. Goel, P. Bartlett, Can a Transformer represent a Kalman filter?, arXiv preprint arXiv:2312.06937 (2023).

[49] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems 30 (2017).

Chapter 6

Conclusions and Future Work

6.1 Summary

This dissertation presents novel neural operator-based methods to address challenges in both Reynolds-averaged Navier–Stokes (RANS) turbulence modeling and simulation acceleration from a fundamental and mathematical perspective—modeling and learning complex nonlocal and nonlinear mappings.

First, to address the limitations of the weak equilibrium assumption in conventional eddy viscosity turbulence models, we develop a fully nonlocal constitutive model using the vector-cloud neural network (VCNN), which models *field-to-point* mappings from the mean velocity fields in the neighborhood to Reynolds stress anisotropies at points of interest [1, 2]. This approach effectively captures the spatial nonlocality inherent in Reynolds stress transport models (RSTMs) and accounts for non-equilibrium turbulence effects arising from Reynolds stress anisotropy transport. Crucially, we embed the essential physical properties required by turbulence and broader constitutive models directly into the design of VCNN, ensuring that the predicted Reynolds stresses are both physically accurate and frame-independent. In addition, through a specialized design, we further ensure that the VCNN remains robust across varying computational settings. Specifically, the model is adaptable to different domain discretizations and maintains permutation invariance, meaning the predicted Reynolds stresses are independent of how neighboring points are arranged in the computational mesh. This method establishes a foundation for developing data-driven, non-equilibrium turbulence models that are computationally robust and have the potential to significantly improve the predictive capabilities of RANS models across a wide range of engineering applications.

Second, to optimize the computational workflow for RANS simulations, we propose using neural operators to provide enhanced initial conditions, thus reducing the number of iterations and accelerating the convergence processes [3]. Although many existing studies attempt to train neural networks as surrogate models to replace numerical simulations for similar acceleration purposes, these approaches may raise trust crises, particularly in applications where high computational accuracy is crucial. Instead of replacing the numerical solver, our method focuses on improving initial conditions, which involves learning *field-to-field* mappings from potential flow fields to near-converged RANS simulation results. We test the effectiveness of our method across scenarios with different levels of nonlinearity, and the results demonstrate that it can achieve at least a twofold acceleration without any loss of accuracy. This performance highlights the robustness of the approach and its potential for broader applications in accelerating RANS simulations while maintaining high-fidelity results.

Third, to address the uncertainty in nonlinear eddy viscosity turbulence models, previous studies have developed ensemble Kalman filter-based methods to infer the models from sparse indirect measurements. However, such methods are inherently limited by the linear and Gaussian assumptions of the ensemble Kalman filter. To overcome these limitations and improve learning performance, we introduce a novel ensemble neural filter that gener-

alizes traditional ensemble-based filters by modeling *ensemble-to-ensemble* mappings using a permutation-equivariant neural operator. This approach enables fully nonlinear data assimilation, allowing the posterior ensemble to extend beyond the subspace spanned by the prior ensemble. The ensemble neural filter is inherently trainable and has the potential to deliver state-of-the-art assimilation performance. Our experiments show that when trained on data generated by the EnKF, the ensemble neural filter accurately reproduces the EnKF's behavior. Moreover, it significantly outperforms the EnKF in scenarios with small ensemble sizes, where conventional methods often struggle due to sampling errors and reduced representational capacity. This capability allows for more accurate inference of nonlinear eddy viscosity models with significantly lower computational costs and holds promise for broader applications in sequential data assimilation tasks.

6.2 Future Direction

Despite the progress achieved in this dissertation, several future research directions are necessary to further enhance the applicability of these methods in practical engineering contexts.

First, although the VCNN has shown promise in modeling non-equilibrium turbulence, it has only been validated in an *a priori* setting. To thoroughly evaluate its advantages over traditional weak-equilibrium turbulence models, the trained VCNN should be coupled with the RANS equations and assessed in *a posteriori* simulations [4]. This integration would enable a more comprehensive evaluation of its performance and robustness across different turbulent flow regimes, thereby establishing its effectiveness for practical turbulence modeling.

Second, the current work on initialization-driven simulation acceleration has been limited to 2D cases. However, many practical applications in aerospace engineering, such as flow over aircraft wings and other aerodynamic surfaces [5], involve complex 3D geometries and turbulent flow features. Extending the neural operator-based warm-start methods to 3D scenarios and validating them in such high-dimensional, realistic settings is crucial to demonstrating their practical effectiveness in accelerating real-world engineering processes.

Third, the development of the ensemble neural filter is still in its early stages and has only been tested on benchmark systems like the Lorenz-63 and Lorenz-96 models. To fully establish its capability in handling complex dynamical systems such as turbulent flows, it is essential to rigorously evaluate the method on more realistic and challenging scenarios. Moreover, additional research is needed to explore how the ensemble neural filter can be further enhanced and implemented to solve inverse problems, including inferring nonlinear eddy viscosity models from sparse indirect measurements. Simultaneously, it is important to investigate and incorporate the latest machine learning advancements, such as Transformers and diffusion models [6, 7], to further elevate the data assimilation performance.

Pursuing these research directions will contribute to a more robust and reliable framework for integrating data-driven models into turbulence modeling and other complex engineering

systems, ultimately improving both the predictive accuracy and computational efficiency of RANS models across a broad range of practical applications.

Bibliography

[1] Xu-Hui Zhou, Jiequn Han, and Heng Xiao. Frame-independent vector-cloud neural network for nonlocal constitutive modeling on arbitrary grids. *Computer Methods in Applied Mechanics and Engineering*, 388:114211, 2022.

[2] Jiequn Han, Xu-Hui Zhou, and Heng Xiao. An equivariant neural operator for developing nonlocal tensorial constitutive models. *Journal of Computational Physics*, 488:112243, 2023.

[3] Xu-Hui Zhou, Jiequn Han, Muhammad I Zafar, Christopher J Roy, and Heng Xiao. Neural operator-based super-fidelity: A warm-start approach for accelerating steady-state simulations. *arXiv preprint arXiv:2312.11842*, 2023.

[4] Ruiying Xu, Xu-Hui Zhou, Jiequn Han, Richard P Dwight, and Heng Xiao. A PDE-free, neural network-based eddy viscosity model coupled with RANS equations. *International Journal of Heat and Fluid Flow*, 98:109051, 2022.

[5] Shamsheer S Chauhan and Joaquim RRA Martins. RANS-based aerodynamic shape optimization of a wing considering propeller–wing interaction. *Journal of Aircraft*, 58(3): 497–513, 2021.

[6] Gautam Goel and Peter Bartlett. Can a Transformer represent a kalman filter? In *6th Annual Learning for Dynamics & Control Conference*, pages 1502–1512. PMLR, 2024.

[7] François Rozet and Gilles Louppe. Score-based data assimilation. *Advances in Neural Information Processing Systems*, 36:40521–40541, 2023.