Data-Centric Engineering (2025), 6: e5
doi:10.1017/dce.2024.21

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

Discretization-independent surrogate modeling of physical fields around variable geometries using coordinate-based networks

James Duvall and Karthik Duraisamy

Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI, USA
Corresponding author: James Duvall; Email: jamesduv@umich.edu

Received: 28 June 2023; **Revised:** 18 April 2024; **Accepted:** 17 May 2024

Keywords: aerodynamics; hypernetworks; neural networks; surrogate model

Abstract

Numerical solutions of partial differential equations require expensive simulations, limiting their application in design optimization, model-based control, and large-scale inverse problems. Surrogate modeling techniques aim to decrease computational expense while retaining dominant solution features and characteristics. Existing frameworks based on convolutional neural networks and snapshot-matrix decomposition often rely on lossy pixelization and data-preprocessing, limiting their effectiveness in realistic engineering scenarios. Recently, coordinate-based multilayer perceptron networks have been found to be effective at representing 3D objects and scenes by regressing volumetric implicit fields. These concepts are leveraged and adapted in the context of physical-field surrogate modeling. Two methods toward generalization are proposed and compared: design-variable multilayer perceptron (DV-MLP) and design-variable hypernetworks (DVH). Each method utilizes a main network which consumes pointwise spatial information to provide a *continuous representation* of the solution field, allowing discretization independence and a decoupling of solution and model size. DV-MLP achieves generalization through the use of a design-variable embedding vector, while DVH conditions the main network weights on the design variables using a hypernetwork. The methods are applied to predict steady-state solutions around complex, *parametrically defined geometries* on *non-parametrically-defined meshes*, with model predictions obtained in less than a second. The incorporation of random Fourier features greatly enhanced prediction and generalization accuracy for both approaches. DVH models have more trainable weights than a similar DV-MLP model, but an efficient batch-by-case training method allows DVH to be trained in a similar amount of time as DV-MLP. A vehicle aerodynamics test problem is chosen to assess the method's feasibility. Both methods exhibit promising potential as viable options for surrogate modeling, being able to process snapshots of data that correspond to different mesh topologies.

Impact statement

Many existing machine learning techniques for surrogate modeling of spatial fields are unable to handle variable mesh topologies and associated unstructured meshes. Further, many techniques also process entire solution snapshots as inputs and outputs, presenting scaling difficulties when applied to very large engineering simulation datasets. The deep learning methods presented in this work allow for fast surrogate models of the solution fields to be constructed via training on heterogeneous data snapshots that may involve variable geometries and mesh topologies. The solution is constructed as a continuous field. Demonstrations are shown on a vehicle aerodynamics application, and the performance is enhanced by leveraging different approaches for training and the use of random Fourier features.

1. Introduction

High-fidelity numerical simulations are ubiquitous in engineering design and analysis but are often prohibitively expensive in design applications. Data-driven and machine-learning surrogate-modeling techniques offer an interesting alternative, particularly in situations where model accuracy may be acceptably traded for computational savings. However, many existing methods face limitations when confronted with unstructured and varying mesh topologies across the parameter or design-variable space. This confines such methods to problems that can be defined with a shared discretization or requires additional lossy interpolation to map solutions onto consistent meshes. These limitations pose a significant challenge for problems involving multiscale phenomena, commonly found in fluid and structural mechanics, where solutions frequently contain regions with large gradients and tightly clustered mesh cells. Additionally, the domains may contain intricate and varying geometric features among solution instances. In such scenarios, interpolating the solutions to a common and often Cartesian mesh results in unacceptable loss of critical information and fidelity.

Many advances in machine learning have been driven by methods which approximate mappings involving high-dimensional spaces, with the cardinality of input or output spaces $\sim \mathcal{O}\left(10^2 - 10^5\right)$ (Krizhevsky et al., 2012; Lecun et al., 1998; Ronneberger et al., 2015; Vincent et al., 2008). This includes matrix-decomposition-based methods (Kutz et al., 2016; Willcox and Peraire, 2002) and deep-learning autoencoder techniques (Bhatnagar et al., 2019; Thuerey et al., 2020; Xu and Duraisamy, 2020) used in scientific applications. Computational limitations arise when applied to scientific or engineering simulations of scale, where the mesh-cell cardinality N may be in the tens-of-millions or even billions in supercomputing settings (Arroyo et al., 2021), with the total degrees-of-freedom for a solution state an even-larger multiple of N. This requires mapping between high-dimensional snapshots in autoencoder-style models, and decomposing even larger snapshot matrices for projection-based ROMs, dynamic mode decomposition, or related Koopman methods (Schmid, 2022).

A recent line of research instead approximates infinite-dimensional (continuous) functions by mapping between lower-dimensional spaces using simple coordinate-based fully connected multi-layer perceptron (MLP) neural networks. That is, instead of mapping between snapshots over the full domain of the problem, for example, with a nonautoassociative autoencoder convolutional neural network (CNN) (Bhatnagar et al., 2019), a mapping is regressed between inputs and outputs at each individual point in space. The resulting key distinction is that coordinate inputs $\mathbf{x}$ are taken *pointwise*, for example, as a single physical-coordinate tuple (x, y, z), instead of entire-solution snapshots. A driving application is object and scene representation for rendering in computer graphics by which objects are represented by continuous implicit fields such as the signed-distance function (SDF) zero-level set (Davies et al., 2020; Park et al., 2019), SDF decision boundary (Mescheder et al., 2019), or as a density/differential-opacity along a light ray (Mildenhall et al., 2021).[1] This concept is known by several names, including *coordinate-based networks*, *neural fields*, *neural implicits*, and *implicit neural representations* (Xie et al., 2022), the latter of which provides a framework which encompasses and generalizes the methods, extending them to other problem scenarios including recovery of typical supervised learning problems. Using a continuous representation is key to attaining discretization independence when applied to scientific simulation data. Every point in each mesh may be included separately, eliminating the need for lossy interpolation of solution data onto a common Cartesian mesh.

In this work, coordinate-based MLPs are applied to the prediction of partial differential equation (PDE) solutions with variable geometry. The predictive models must be able to concurrently handle unstructured data and variation in physical design and operating conditions, as described by design-variable vector $\boldsymbol{\mu}$, to be useful in design optimization or other engineering tasks. The physical coordinate inputs are augmented to include an evaluation of the SDF to provide global information about the domain at each point; this may be viewed as a form of concatenation-based *local* conditioning as the SDF is a function of space over the relevant domains. In addition to the SDF, the network predictions are *globally* conditioned upon the

[1] See Equation 3 for SDF definition.

design variables $\boldsymbol{\mu}$. Several different techniques for conditioning neural network predictions are used in the literature (Dumoulin et al., 2018; Park et al., 2019; Perez et al., 2018; Xie et al., 2022) with concatenation-based conditioning and the use of hypernetworks (Ha et al., 2016) explored here. The method which utilizes concatenation-based conditioning is termed design-variable multilayer perceptron (DV-MLP), while design-variable hypernetworks (DVH) of course utilize a hypernetwork structure. Most conditioning schemes involve learning an additional embedding to condition the networks upon, whereas here, the design variables and SDF are used instead; both of which are known once a design is selected, simplifying the overall scheme.

The outline of the article is as follows: Background information relating to existing discretization-dependent methods along with approaches for generalization and conditioning of predictions are given in the remainder of the introduction, Sections 1.1–1.3. The proposed methods are treated in greater detail in Section 2. Numerical experiments for 2D vehicle aerodynamics with complex, realistic automotive shapes are given in Sections 3 and 4. Section 3 provides a baseline result for single and multiple vehicle speeds, and Section 4 explores the effects of using random Fourier features, where accuracy and generalization properties are improved. Conclusions follow in Section 5, with additional information given in the Appendix.

1.1. Discretization-dependent methods

As a classical dimensionality-reduction technique, proper orthogonal decomposition (POD) has been used to construct surrogate and reduced-order models (Benner et al., 2015; Dolci and Arina, 2016; Salmoiraghi et al., 2018; Willcox and Peraire, 2002). Despite many attractive properties, conventional POD implementations process discretized data and require the use of a fixed topology mesh across *all parameter regimes*, fixing the number of degrees-of-freedom. This is restrictive in many engineering problems in which various solution features (e.g., relative motion of bodies, crack propagation, etc.) may emerge in different regions of parameter space. Further, data may be available from multiple sources with varying discretization and mesh topologies. Other snapshot-based methods inherit these disadvantages, including autoencoders and their variants.

CNNs have been used to construct surrogate models for both steady-state (Bhatnagar et al., 2019; Guo et al., 2016; Ronneberger et al., 2015; Tangsali et al., 2021; Thuerey et al., 2020) and time-varying parametric problems (Hasegawa et al., 2020; Xu and Duraisamy, 2020) by including an additional time-advance model such as a long short-term memory (LSTM) or temporal-convolutional network. However, CNNs place even greater restrictions on the discretization than POD-based methods, requiring inputs and outputs to be defined on regular Cartesian grids of consistent dimension for all parameter regimes. Overcoming this restriction requires interpolation from the computational mesh to a Cartesian grid overlain on the problem domain, equivalent to pixelization. The interpolation results in a number of undesirable effects, including a reduced-fidelity representation of the domain geometry, and a loss of information in regions of tightly clustered mesh points, such as within boundary layers, shocks, and wakes. CNN-based models may then be conceptualized as image-to-image mappings, where researchers Guo et al. (2016) note improved results when using a signed-distance field as the network input in contrast to a binarized representation, inspiring its use as an additional input feature in this work for coordinate-based networks.

Another more problematic but surprisingly overlooked issue is that the memory requirements for 3D convolutions, commonly implemented on a single GPU, are not affordable for typical resolutions in realistic engineering problems. When taking mini-batch training into consideration, even storing the output of one single hidden layer (a 5-dimensional tensor), requires memory typically on the order of $O(10) - O(10^2)$ GB for a 3D Cartesian field with 40 million cells. As a result, most reported works using 3D CNNs for engineering problems are limited to below 5-6 million degrees of freedom (Mohan et al., 2019; Santos et al., 2020) and often still require lossy interpolation (Gao et al., 2022).

1.2. Hypernetworks and methods for conditioning neural networks

Hypernetworks constitute a metamodeling approach where one neural network is used to generate the weights of another main network (Ha et al., 2016) and are part of a broader class of proposed techniques

where network weights are conditioned on model inputs or features (Bertinetto et al., 2016; Jaderberg et al., 2015; Jia et al., 2016). Hypernetworks were originally applied to convolutional and recurrent neural networks for image- and natural-language-processing tasks, with the goal of reducing the number of trainable parameters while maintaining or improving model accuracy. The use cases for hypernetworks have largely been the domain of computer science but lately have been applied to scientific machine learning in some instances. Pan et al., 2023 leveraged a hypernetwork structure known as neural implicit flow (NIF) to learn latent representations from turbulence on arbitrary meshes in a scheme similar to DVH used here. HyperPINNs apply hypernetworks to physics-informed neural networks (PINNs) (Raissi et al., 2019) for parametric PDE solutions of 1D-viscous Burgers and the Lorenz system, with improved accuracy seen over baseline PINN models despite a smaller main network (de Avila Belbute-Peres et al., 2021).

Other methods of conditioning include the use of feed-forward encoders, auto-decoders (Park et al., 2019), concatenation-based conditioning, and feature-wise transformations (Dumoulin et al., 2018; Perez et al., 2018). Hypernetworks are the most general conditioning method, as all the other methods may be derived from a hypernetwork, which outputs only a portion of the network weights (Xie et al., 2022). Further, the conditioning may be *global* or *local.* Global conditioning uses a single embedding for an entire instance, whereas local conditioning uses an embedding which is itself a function of space over the domain of interest. In this work, a combination of the two is used, where the inclusion of the SDF coordinate may be viewed as local conditioning, while the use of design variables $\boldsymbol{\mu}$ is a form of global conditioning.

1.3. Other discretization-independent methods

In addition to neural fields, some graph neural networks (GNNs), point-cloud networks, and operator regression methods also allow for discretization independence. GNNs have been developed to generalize convolutional-like models to problems defined on non-Euclidean domains, or with non-regular Cartesian structure. GNNs may be classified as either spectral (Bruna et al., 2014; Defferrard et al., 2016; Henaff et al., 2015; Kipf and Welling, 2017) or spatial (Duvenaud et al., 2015) approaches, although the two may be generalized by the message-passing graph neural network (MPGNN) (Gilmer et al., 2017). Spectral GNNs require a consistent discretization among all instances similar to POD, while MPGNNs allow the meshes to vary. MPGNNs have been used for body-force predictions of aerodynamic flows (Ogoke et al., 2021). Additionally, MPGNNs are used as a sub-component for certain learning and prediction schemes, with a focus on PDEs in either a mesh-based (Pfaff et al., 2020; Xu et al., 2021) or mesh-free scenario (Sanchez-Gonzalez et al., 2020) in addition to some neural operator methods discussed later. Additionally, Geodesic CNNs can handle varying mesh topologies and have been used for aerodynamics and optimization (Baque et al., 2018), along with heat transfer through porous media (Mallya et al., 2023).

Point cloud neural networks are useful in situations in which the data is available in the form of unstructured point clouds, such as the raw output of a LIDAR unit or other three-dimensional sensor. As such they are often seen in the context of autonomous vehicles or robotic vision. PointNet (Qi, Su, et al., 2017) and PointNet++ (Qi, Yi, et al., 2017) are architectures designed for point clouds and are used for scene recognition, classification, and segmentation tasks. PointNet++ has been adapted in the context of predicting viscous, incompressible flows over 2D shapes lying on unstructured meshes (Kashefi et al., 2021).

Another class of relevant techniques capable of handling unstructured data includes operator-regression methods, such as those based on DeepONet (Cai et al., 2021; Lu et al., 2021; Wang et al., 2021), Neural Operator (Kovachki et al., 2023; Li, Kovachki, Azizzadenesheli, Liu, Bhattacharya, et al., 2020; Li, Kovachki, Azizzadenesheli, Liu, Stuart, et al., 2020), and GMLS Nets (Trask et al., 2019). DeepONet's structure may be viewed as a partial hypernetwork for just the output layer and is thus similar to what is pursued here with DVHs, with another major difference corresponding to hypernetwork (branch network) inputs. The DeepONet branch network takes as input a sampling of input functions across the domain, and in fact DeepONet is conceived in 1D with reference to an operator representation

theorem, which further assumes the sensors have consistent sampling locations which span the domain. Unstructured meshes with varying topology among instances generally do not have a shared set of points to place sensors among all instances, complicating the use of DeepONet in the present scenario.

2. Methods

Engineering models are typically expressed as systems of PDEs describing the design under consideration, with the highest-fidelity versions termed the full-order model (FOM). Quantities of interest (QoIs) relevant to assessing the design are extracted from numerically generated FOM solutions. In the context of design, the FOM may be parameterized by vector $\boldsymbol{\mu} \in \mathbb{R}^{n_\mu}$ so that important and defining design elements may be varied and the QoI response measured. Often, these variations include the shape or geometry of some problem element, and these geometric design variables may be collected and represented as $\boldsymbol{\mu}_{\text{geo}} \in \mathbb{R}^{n_{\mu_{\text{geo}}}}$. These parameters are often used with and defined in the context of a separate parametric geometry model, relating the design variables to the fine details of the physical design. In this work, the design variables $\boldsymbol{\mu}$ are used in place of a learned representation in order to condition predicted PDE solutions $\mathbf{q}(\mathbf{x};\boldsymbol{\mu}) \in \mathbb{R}^{n_q}$. Two methods of global conditioning are used, including concatenation-based conditioning and the use of dense hypernetworks. Additionally, the signed-distance function $f_{\text{sdf}}(\mathbf{x};\Omega)$ or minimum distance function $\phi(\mathbf{x};\Omega)$ are used as inputs to the main network as an implicit representation of the domain geometry, along with the spatial coordinates $\mathbf{x} \in \mathbb{R}^{n_x}$. This may be viewed as a form of local concatenation-based conditioning.

Consider FOMs for steady-state, parametric PDEs, written generally as

$$\mathcal{R}(\mathbf{x}, \mathbf{q}(\mathbf{x}), \ldots; \boldsymbol{\mu}) = 0, \quad \mathbf{x} \in \Omega\left(\boldsymbol{\mu}_{\text{geo}}\right), \tag{2.1}$$

with boundary conditions prescribed as

$$\mathcal{B}(\mathbf{x}, \mathbf{q}(\mathbf{x}), \ldots; \boldsymbol{\mu}) = 0, \quad \mathbf{x} \in \partial\Omega\left(\boldsymbol{\mu}_{geo}\right), \tag{2.2}$$

where $\mathbf{q}$ is the solution state, $\Omega/\partial\Omega \subset \mathbb{R}^{n_x}$ are the problem domain/boundary which are implicit functions of $\boldsymbol{\mu}_{\text{geo}}$, $\mathbf{x} \in \Omega$ are the spatial coordinates, $\mathcal{R}$ is the PDE operator, and $\mathcal{B}$ is the boundary-condition operator. The geometric design variables $\boldsymbol{\mu}_{\text{geo}}$ are included as part of $\boldsymbol{\mu}$, which may also contain PDE coefficients or numerical values of boundary or operating conditions. In this context, the elements of $\boldsymbol{\mu}$ apply to the entire physical domain over which the FOM is solved, in contrast to problem scenarios involving spatially distributed parameters, such as a thermal conductivity or another physical property.

2.1. Shape and scene representation via coordinate-based neural networks

Given a set of points representing a surface $\mathcal{S} = \partial\mathcal{V}$ of an object $\mathcal{V} \in \mathbb{R}^m$ in m-dimensional physical space, the signed-distance function may be defined as

$$f_{\text{sdf}}(\mathbf{x};\mathcal{S}) \triangleq \begin{cases} \phi(\mathbf{x},\mathcal{S}) & \mathbf{x} \notin \mathcal{V} \\ 0 & \mathbf{x} \in \mathcal{S}, \\ -\phi(\mathbf{x},\mathcal{S}) & \mathbf{x} \in \mathcal{V} \end{cases} \tag{2.3}$$

where

$$\phi(\mathbf{x},\mathcal{S}) \triangleq \inf_{\mathbf{y} \in \mathcal{S}} d(\mathbf{x},\mathbf{y}) \tag{2.4}$$

is a minimum-distance function (MDF), and $d(\cdot,\cdot)$ is the Euclidean-distance function. Stated simply, the SDF is the minimum distance between the field point $\mathbf{x}$ and the surface $\mathcal{S}$ in consideration. It takes positive values for points outside the object ($\mathbf{x} \notin \mathcal{V}$), negative values for points inside ($\mathbf{x} \in \mathcal{V}$), and is identically zero on the surface.

Coordinate-based MLPs have been used to represent 3D objects for rendering tasks. The object's surface is implicitly represented within a volumetric field; including as the zero-level-set of a directly regressed signed-distance field (Davies et al., 2020; Park et al., 2019) or decision boundary (interior/exterior) (Chen and Zhang, 2019; Mescheder et al., 2019), or as an emitted radiance and density/differential-opacity field (Mildenhall et al., 2021). Many of these methods include loss terms describing a rendering process, such that the entire image generation process contributes to the loss during training. This concept may be generalized by *implicit neural representations*, introduced along with sin-activation SIREN networks in by Sitzmann, Martel, et al. (2020), from which defining Equations (2.5) and (2.6) are taken. In this setting, a function of interest Φ with input coordinates $\mathbf{x}$, written $\Phi : \mathbf{x} \rightarrow \Phi(\mathbf{x})$, is defined by a set of constraints $\mathcal{C}$,

$$\mathcal{C}_m\left(\mathbf{x}, \mathbf{a}(\mathbf{x}), \Phi, \nabla_{\mathbf{x}}\Phi, \nabla^2_{\mathbf{x}}\Phi, \ldots\right) = 0, \mathbf{x} \in \Omega_m,\ m = 1, \ldots, M \tag{2.5}$$

which optionally depend on the function values Φ, function gradients $\nabla_{\mathbf{x}}\Phi$, and additional quantities $\mathbf{a}(\mathbf{x})$ which are needed to compute the constraints. When a neural network N with parameters θ is used to approximate Φ, then this is referred to as an *implicit neural representation.* To train the neural-network approximation, a loss function with M terms is defined by penalizing deviation from the constraints,

$$\mathcal{L}(\theta) = \int_{\Omega} \sum_{m=1}^{M} 1_{\Omega_m}(\mathbf{x}) \| \mathcal{C}_m(\theta, \mathbf{x}, \mathbf{a}(\mathbf{x}), \ldots) \| \, d\mathbf{x}, \tag{2.6}$$

where the indicator function 1_{Ω_m} activates over valid locations within the domain Ω_m. A key distinction between this and other methods is that coordinate inputs $\mathbf{x}$ are taken *pointwise*, for example as a single physical-coordinate-tuple (x, y, z), instead of as entire-solution snapshots. Surprisingly, many problems may be cast in this form, including as-discussed surface representation and variations on classic deep-learning problems, such as classifying MNIST hand-written digits. The approaches proposed in this work may be viewed through this lens where the only constraint is that the predictions match the data, recovering basic supervised regression. However, this does not directly align with the main thrust of implicit neural representations, where an implicit function is regressed and additional constraints are imposed.

A popular approach to generalization across instances involves autodecoders such as DeepSDF, which uses a form of concatenation-based conditioning, where concurrently learned embedding-vector $\mathbf{z}_j$ is concatenated with spatial coordinates $\mathbf{x}$ as network input (Park et al., 2019). This is a powerful concept, as it allows for a single network to predict the SDF for many shapes, but it may come at the expense of blurred fine-object details (Davies et al., 2020), along with the need to solve an optimization problem for $\mathbf{z}_j$ in order to make predictions for an unseen case. An alternative to embedding vector concatenation is weight-encoding (Davies et al., 2020), where a neural network is overfit to each shape separately without an embedding vector. In this scenario, the weights themselves are viewed as the embedding. To make predictions for a rendering, the weights for the shape under consideration are loaded and used. This method may provide greater accuracy but has the additional complications of training a separate network for each object and for loading and unloading weights. Additionally, this does not allow one to make predictions for objects outside of the training set. This leads naturally to the concept of applying hypernetworks to the coordinate-based MLPs as an alternative to training networks separately, an idea explored briefly by Sitzmann, Chan, et al. (2020).

2.2. Problem setup

Denote the solution snapshot for a single instance j of the FOM as

$$\mathcal{D}_j \triangleq \left\{ \{\mathbf{q}_i | \mathbf{x}_i\}_{i=1}^{n_j}, \boldsymbol{\mu}_j \right\}, \tag{2.7}$$

where the solution output-input pairs are defined at n_j spatial (mesh) locations. Considering a dataset

$$D \triangleq \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{n_D}\} \tag{2.8}$$

containing n_D snapshots, a distinctive feature of this approach is that each snapshot may correspond to a solution domain with different spatial extent and discretization, with varying number and location of mesh points. Models are sought which can approximate the solution snapshots stored in D, without interpolation of ground-truth data or prediction. In other words, given the generative factors or design variables for a problem $\boldsymbol{\mu} \in \mathcal{M} \subset \mathbb{R}^{n_\mu}$, predict the system state $\mathbf{q}$ at any location $\mathbf{x} \in \Omega(\boldsymbol{\mu})$. Denote the input space as $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^{n_x}$, and the output space as $\mathbf{q} \in \mathcal{Q} \subset \mathbb{R}^{n_q}$.

The desired model should generalize across solution instances, and thus approximate the mapping $f : \mathcal{M} \times \mathcal{X} \to \mathcal{Q}$, without direct knowledge of the system state. Note that this is in contrast to initial-value problems, where the initial state to be integrated forward in time is necessary. The problems considered may contain complex and variable geometry, the input space is augmented to include an additional minimum-distance function coordinate, defined as

$$\mathbf{x}' \triangleq \left[\mathbf{x}^T,\ \phi(\mathbf{x};\boldsymbol{\mu})\right]^T. \tag{2.9}$$

In the scenarios considered, all mesh-distances are positive, so the use of MDF and SDF is equivalent. This defines an augmented input space, $\mathcal{X}' \subset \mathbb{R}^{n_x+1}$, where $\mathbf{x}' \in \mathcal{X}'$, and in turn, the desired mapping is

$$f : \mathcal{M} \times \mathcal{X}' \to \mathcal{Q}. \tag{2.10}$$

The model approximation is then written as $f(\mathbf{x}';\boldsymbol{\mu}) \approx \hat{\mathbf{q}}(\mathbf{x}';\boldsymbol{\mu})$ or just $\hat{\mathbf{q}}$ in compact notation. Each design variable $\boldsymbol{\mu}$ defines a spatial solution field over a domain, thus it is natural to condition the models upon the design variables.

2.3. Method 1: design-variable MLP (DV-MLP)

DV-MLP uses concatenation-based conditioning to account for different solution instances, where the augmented coordinates $\mathbf{x}'$ are concatenated with the design variables $\boldsymbol{\mu}$. This is a from of global conditioning as the same $\boldsymbol{\mu}$ is used for all spatial locations for a given solution field. The main network (denoted as N_m with weights θ_m) and resulting prediction are written as

$$\hat{\mathbf{q}}(\mathbf{x}',\boldsymbol{\mu}) = N_m(\mathbf{x}',\boldsymbol{\mu};\theta_m). \tag{2.11}$$

Simple fully connected layers are used, where the hidden state of each layer has the same dimension or number of nodes, and no skip or recurrent connections are used. See Figure 1a for a schematic.

2.4. Method 2: design-variable hypernetworks (DVH)

DVH generates the weights and biases of the main network θ_m using a dense hypernetwork which takes as input the design variables $\boldsymbol{\mu}$. Davies et al. (2020) noted an autodecoder trained to represent many shapes had marginally worse performance representing the fine-grain details of the objects as compared to overfitting a network on each case separately. In an effort to avoid this loss in fine-grain detail and to avoid training a separate model for each case, a hypernetwork is used to generate main network weights θ_m for each solution instance. The hypernetwork is written as

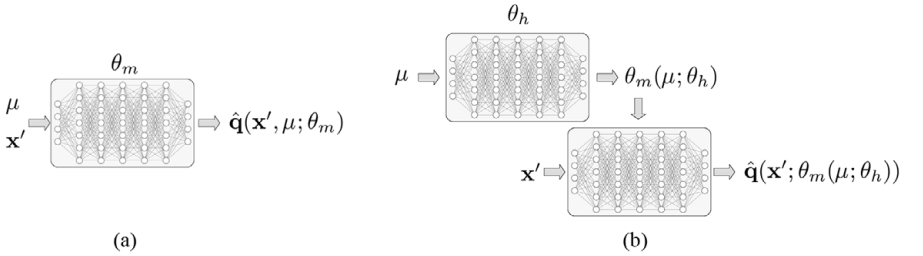


Figure 1. *Network schematics for (a): DV-MLP and (b) DVH.*

$$\theta_m(\boldsymbol{\mu}) = N_h(\boldsymbol{\mu};\theta_h), \tag{2.12}$$

and the main-network prediction written as

$$\hat{\mathbf{q}}(\mathbf{x}';\theta_m(\boldsymbol{\mu})) = N_m(\mathbf{x}';\theta_m(\boldsymbol{\mu})), \tag{2.13}$$

where θ_h and θ_m are the weights and biases of the hypernetwork and main network. In the following experiments, simple MLPs are used for both the main network and hypernetwork. All of the weights and biases contained in θ_m are generated at once as one large vector which is sliced and reshaped as required. The training loss depends on the main-network predictions, but only the hypernetwork weights θ_h are adjusted during training; the loss gradients are back-propagated through the hypernetwork. See Figure 1b for a schematic. The use of a hypernetwork in this way is a form of global conditioning upon the design variables $\boldsymbol{\mu}$.

2.4.1. Network-size scaling considerations

Dense hypernetworks have an important scaling consideration relating the number of trainable hypernetwork parameters in θ_h to the size of the main network θ_m. Consider a main network with L_m hidden layers each with a hidden dimension H, and an analogous hypernetwork with hidden dimension H except for the final hidden layer which has dimension H_L. Since all of the main network weights are generated at once, the output layer of the hypernetwork has roughly H_L times as many weights as the main network. That is,

$$\dim(\theta_h) \propto \dim(\theta_m) H_L \propto L_m H^2 H_L, \tag{2.14}$$

showing that the total number of hypernetwork weights is linear in main-network depth L_m, quadratic in main-network hidden dimension H, and linear in hypernet-final-hidden dimension H_L. This is intuitive given the use of dense layers throughout, but Equation 2.14 is developed in more detail in Appendix Section A.1. Once a main network architecture is chosen with L_m and H selected, then H_L is the remaining term to be selected which drives the number of weights, which of course must be managed for the given training hardware and problem at hand, suggesting selecting $H_L < H$. This leads to an encoder-like interpretation for the hypernetwork, where the final hidden state is an H_L-dimensional embedding for a linear neural-network generator; the hypernetwork output layer. The interpretation exists regardless, even if $H_L \geq H$, but the bottleneck structure brings it out and is reminiscent of autoencoder-style models.

The scaling relationship of Equation 2.14 also highlights a difference between this and many other hypernetwork models. Frequently, the goal is to reduce the number of trainable parameters in a given main network while retaining predictive accuracy, while here, the differences in accuracy and generalization are studied between vector-embedded and weight-embedded coordinate-based networks. In the following experiments, H_L is chosen to be simply $H_L = H = 50$, across main and hypernetworks, meaning that the size of the DVH model is roughly 50 times larger than the DV-MLP model, as seen in Section A.2 of the Appendix. This is taken into account by comparing the time and resources needed to train each type of model while also assessing predictive accuracy. Further, the training methods which follow in Section 2.4.2 are found to have a large impact on the required training time. Differing hypernetwork architectures which reduce the number of trainable parameters are possible but are outside the scope of this paper.

2.4.2. Training considerations

Equation 2.14 implies that the computational complexity of training DVH models may follow a similar scaling relative to training DV-MLP models. This is investigated by considering two approaches for training DVH models, with differences relating to details of model evaluation. Consider a minibatch j consisting of N spatial locations drawn from each of M solution instances. Represent the minibatch as a tuple of tensors, $\left(\mathbf{M}_j, \mathbf{X}_j, \mathbf{Q}_j\right)$, where tensor $\mathbf{M}_j$ holds hypernetwork inputs, $\mathbf{X}_j$ holds main network inputs,

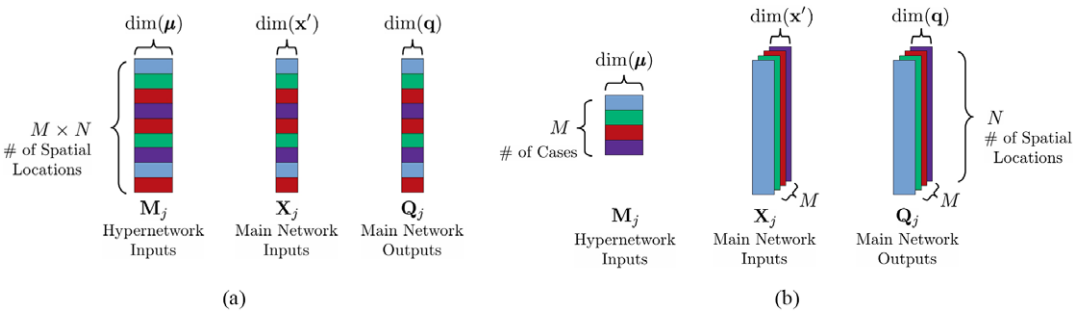


Figure 2. *Illustrating the difference between (a) fully-mixed batches and (b) batch-by-case training minibatches by considering the shape and dimension of the training arrays for a single batch j. Colors correspond to data from a given case.*

and $\mathbf{Q}_j$ holds the target solution variables. The tensor shapes vary between the following methods, which are described below and represented in Figure 2.

- **Method 1: Fully-Mixed Batches** Minibatches are created, which consist of points from different cases, with both the hypernetwork and main network forward-propagated for each data point, meaning hypernetwork input vector $\boldsymbol{\mu}$ is tiled across the mesh. In this scenario, all tensors have two axes, $\dim(\mathbf{M}_j) = (M \times N) \times n_\mu$, $\dim(\mathbf{X}) = (M \times N) \times n_{x'}$, and $\dim(\mathbf{Q}) = (M \times N) \times n_q$ as shown in Figure 2a. Let $\mathcal{C}_M/\mathcal{C}_H$ be the cost for each the main and hypernetwork, then forward propagating the minibatch is proportional to

$$\mathcal{C}_{FM} \propto (M \times N)\mathcal{C}_H + (M \times N)\mathcal{C}_M \tag{2.15}$$

  The data are *fully mixed* by shuffling over all locations and solution instances used in building the minibatch. The batch size then corresponds to the number of spatial locations where predictions are sought.
- **Method 2: Batch-by-Case** This method takes advantage of the fact that design variables $\boldsymbol{\mu}$ apply to an entire solution instance and that the hypernetwork is a neural-network generator. A single forward pass of the hypernetwork is combined with multiple forward passes of the main network for a given number of spatial locations, all coming from the same solution instance, defined by $\boldsymbol{\mu}$. In this scenario, the design-variable tensor has two axes, $\dim(\mathbf{M}_j) = M \times n_\mu$, while the other tensors have three; $\dim(\mathbf{X}) = M \times N \times n_{x'}$ and $\dim(\mathbf{Q}) = M \times N \times n_q$, as represented in 2b. The complexity of forward-propagating the minibatch scales as

$$\mathcal{C}_{BC} \propto M \times \mathcal{C}_H + (M \times N)\mathcal{C}_M. \tag{2.16}$$

  The batch size is then the number of cases per mini-batch, where the same number of spatial locations N are evaluated for each case in each minibatch.

Comparing first terms between Equations 2.15 and 2.16 shows the batch-by-case first-term complexity is reduced by a factor of $N \times \mathcal{C}_H$ due to the many fewer expensive hypernetwork calls. The actual computational complexity will be measured through profiling in later sections.

3. Numerical experiments I: vehicle aerodynamics

External vehicle aerodynamics are considered, with parametric vehicle shapes lying on unstructured, nonparametric meshes. The incompressible RANS equations were solved using Star CCM+ with the k-ε turbulence model. The dataset—generated by General Motors, Inc.—consists of 2D slices along the vehicle centerline for 124 unique vehicle shapes at speeds of 90 and 130 kilometers-per-hour

Table 1. *Description of entries in geometric design-variable-vector $\boldsymbol{\mu}_{\text{geo}}$ for the 2D vehicle aerodynamics dataset*

Design variable	Units	Range	Design variable	Units	Range
Backlight angle	Degrees	[25,57]	Windshield Angle	Degrees	[57,63]
Face lip angle	Degrees	[0,5]	Hood Front Angle	Degrees	[10,20]
Angle of approach	Degrees	[15,25]	Angle of Departure	Degrees	[15,25]
Vehicle length	mm	[3800,4900]	Floor to Roof Height	mm	[1448,1788]

(kph). The incompressible RANS equations are shown and discussed in greater detail in Appendix Section A.5, including a discussion of nondimensional flow variables. The simulations utilize unstructured polyhedral meshes of varying size, with an example mesh shown in Figure 5a. Each vehicle shape is parameterized by 8 geometric parameters as summarized in Table 1, with all 124 shapes overlain on one set of axes in Figure 5b. The vehicle designs were selected using Latin hypercube sampling, and random subsets of the dataset are chosen for training and validation.

The drag coefficient is an important quantity commonly used in assessing aerodynamic designs and is given by

$$C_D = \frac{F_D}{\frac{1}{2}\rho_\infty u_\infty^2 A}, \tag{3.1}$$

where F_D is the drag force, consisting of pressure-force and skin-friction components, subscript ∞ corresponds to freestream conditions, and A is the frontal area. Only three of the design variables, the windshield angle, vehicle length, and floor-to-roof height, are significantly correlated with C_D, as shown in Figure 3. The Pearson correlation coefficient for each is given on the plots, with a p-value $p < 0.05$ as the criteria for determining significance. The floor-to-roof height shows the strongest correlation, with taller vehicles having greater drag.

The flow field structure for all vehicles and both speeds is typical for a time-averaged vehicle wake. Dominant features include a region of high pressure in front of the vehicle as the flow stagnates upon the grille, a low pressure region on and above the vehicles roof, and a large separated wake behind the vehicle as the pressure recovers to free-stream values, with a decaying free-shear layer emanating from the vehicle's rear-upper surface. The pressure forces on the vehicle contribute strongly to the drag, and thus, examining the minimum and maximum pressures in the field gives insight to the variation in the dataset. Scatter plots of the min/max dimensional and nondimensional static pressure are shown in Figure 4. When

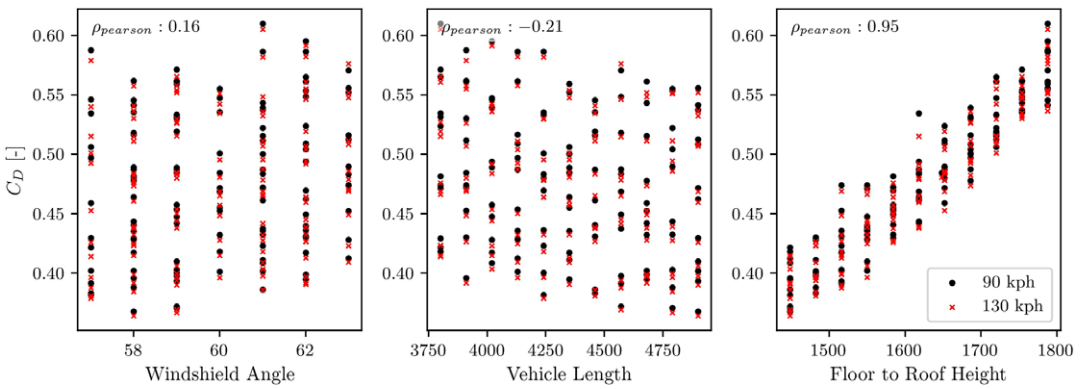


Figure 3. *The drag coefficient versus significantly correlated design variables of windshield angle, vehicle length, and floor-to-roof height. The Pearson correlation coefficient is reported in each subplot.*

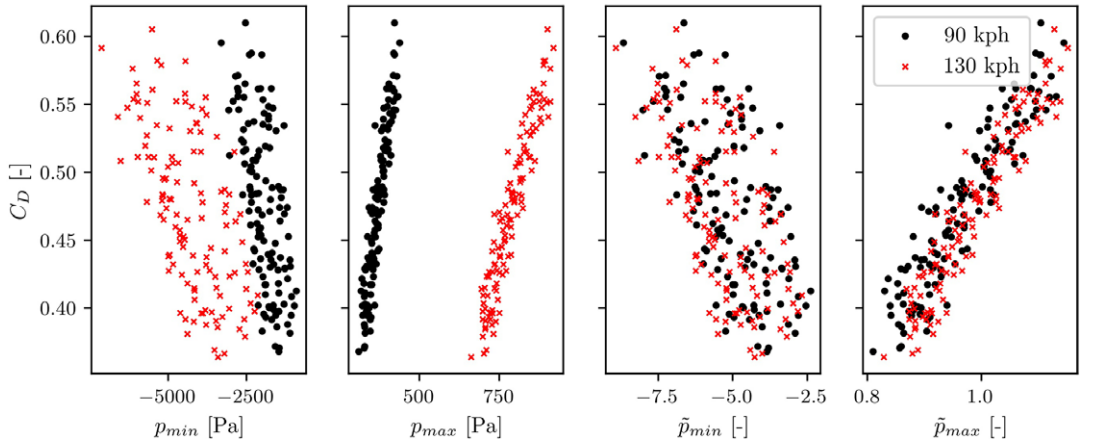


Figure 4. *The drag coefficient versus max/min dimensional static pressure, and max/min nondimensional static pressure.*

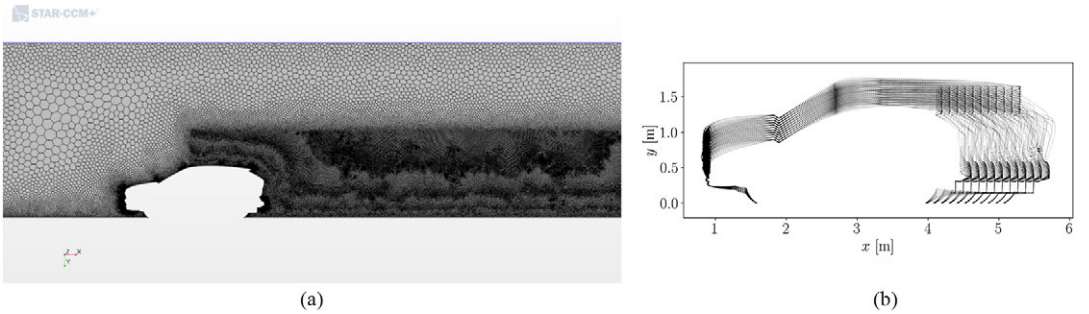


Figure 5. *Pertaining the training dataset, (a) an example unstructured CFD mesh and (b) a composite image of all 124 vehicle shapes overlain on the same axes*

dimensional static pressure is considered, the two left-most plots, the data are grouped by vehicle speed. However, when nondimensional pressure is used, as with the two right-most plots, the distributions collapse and one trend is observed for both vehicle speeds.

The computational fluid dynamics (CFD) meshes vary in size and topology, with the number of cells ranging from 108,748 to 115,751. In all sections, a spatial batch size of 54,000 points is used, resulting in two minibatches per case for a total of 108,000 mesh points used for each case. Mesh points are randomly dropped from each solution as required to make the minibatches, corresponding to 0.7%–6.7% of the points being left out for each training case. Reported training-error metrics include all mesh points, even if they were dropped from the training set, with further discussion of this in Section 3.2. The effect of varying spatial batch sizes is discussed in the Appendix, Section A.6.

The spatial input quantities are

$$\mathbf{x}' = [x, y, \phi(x,y)]^T \in \mathbb{R}^3, \tag{3.2}$$

and the predicted state is

$$\mathbf{q} = [p(x,y), u(x,y), v(x,y)]^T \in \mathbb{R}^3. \tag{3.3}$$

The design-variable vectors differ slightly depending on the number of speeds considered and are given as

$$\boldsymbol{\mu} = \begin{cases} \boldsymbol{\mu}_{\text{geo}} \in \mathbb{R}^8 & n_{\text{speeds}} = 1, \\ \left[\text{Re}, \boldsymbol{\mu}_{\text{geo}}^T\right]^T \in \mathbb{R}^9 & n_{\text{speeds}} > 1. \end{cases} \tag{3.4}$$

All spatial and predicted state quantities in $\mathbf{x}'$ and $\mathbf{q}$ are nondimensionalized according to Section A.5 of the Appendix. Nondimensional quantities and dimensional analysis are widely used in fluid mechanics, leading to satisfying similarity solutions and increased interpretability. Following nondimensionalization, all input and output vectors are min–max normalized component-wise using training-set stats before use with the neural networks. Details on this normalization and other implementation details related to training and initialization are given in Appendix Section A.3. All reported errors are in the fully-dimensional units of the state, with the exception of training curves which correspond to fully-normalized quantities; see Section A.4 of the Appendix for error-metric definitions. Without normalization, the loss and loss-gradients may be biased toward the state quantities with the largest unit-scale. Normalization eases this problem and ensures that smaller output quantities are not ignored during training.

3.1. Model architecture and training options

In this section, 5-hidden-layer networks are used for both main and hypernetworks, each with a hidden dimension of 50 for all layers. Model architecture summaries are given in Appendix Section A.2, with the number of trainable weights for Section 3 given in Table 2. As noted in Section 2.4.1, the DVH model has roughly $H_L = 50$ times as many trainable weights as the corresponding DV-MLP model, and the computational consequences of this are explored and quantified here through profiling. Three model-method combinations are considered and include DV-MLP with its sole fully-mixed training mode, DVH method 1 (M1) fully mixed batches, and DVH method 2 (M2) batch-by-case. See Section 2.4.2 for greater detail. The average step time, average compute time, and maximum memory usage during training among the models and methods are reported in Table 3, where the spatial batch size is 54,000 points for DV-MLP and

Table 2. *Number of trainable parameters for DV-MLP and DVH models for all baseline results of Section 3, where* $H = H_L = 50$, $L_m = L_H = 5$

	# Trainable weights	
Method	Single speed	Multiple speeds
DV-MLP	10,953	11,003
DVH	548,853	548,903

Table 3. *Comparing training profiles among the models and training methods*

Type	Training method	Precision	Δt step [ms]	Δt compute [ms]	Max. Mem. [GB]
DV-MLP	–	float64	18.1 (0.2)	17.2	1.20
DV-MLP	–	float32	3.7 (0.1)	2.9	0.62
*DV-MLP	–	mixed	3.6 (0.5)	1.9	0.54
DVH	M1 Fully-mixed	float64	490.0 (1.6)	488.9	9.70
DVH	M1 Fullymixed	float32	55.6 (0.2)	54.4	4.85
DVH	M1 Fully mixed	mixed	40.5 (0.8)	37.5	2.66
DVH	M2 Batch-by-case	float64	20.1 (0.4)	18.7	0.72
*DVH	M2 Batch-by-case	float32	4.4 (0.1)	3.3	0.36
*DVH	M2 Batch-by-case	mixed	4.3 (0.4)	2.3	0.28

DVH M1 and a corresponding case batch size of 1 for DVH M2. All profiled results were obtained using `tensorboard` callbacks. Three different levels of precision are compared for each method, including double precision (float64), single precision (float32), and mixed precision (combination of float32 and float16). The reported values are averaged over the first 100 minibatches for 100 epochs of training, and the Δt step standard deviation is given in parentheses.

There are several important takeaways from these profiled results. First, the precision has a large impact on the step-times and memory requirements, which decrease massively between float64 and float32 in all instances. Subsequent smaller improvements are seen moving between float32 and mixed precision. Next, M2 batch-by-case training decreases the step time and memory requirements by roughly an order of magnitude as compared to M1 fully mixed for a given precision. Further, the DVH M2 step times are comparable to the DV-MLP step times and consume less memory despite the much greater parameter count. Two average times are given, Δt step, which includes all operations in a single optimizer update, and Δt compute, which includes only GPU operations per update. Rows marked with an asterisk (*) in Table 3 may benefit from improvements in the data pipeline, as the overhead associated with those operations becomes appreciable as the Δt step times decrease.

When comparing the accuracy of the resulting fully trained models, it was found that using single precision generally improved the mean-relative-L2 error (MRL2E) by roughly 1–8 percentage points as compared to mixed precision. The use of double precision negligibly improved the predictive performance beyond that seen with single precision, and in some cases, single precision performed best overall. Thus, given the greatly improved step times from double to single precision, and the smaller improvements afforded by using mixed precision, all models in later sections are trained using single precision unless otherwise stated.

A piecewise learning-rate schedule was used for all experiments, consisting of periods of constant and exponentially decaying learning rates. This was devised to combat large variation around training-loss plateaus seen in some instances and stabilize DVH M1 training dynamics which showed large training-loss fluctuations. Additionally, several influential, state-of-the-art works in machine learning have used a similar scheme whereby the learning rate is manually decreased by a factor of 10 when plateaus in the training loss are observed, a process repeated up to three times (He et al., 2016; Krizhevsky et al., 2012; Simonyan and Zisserman, 2014). This aided model convergence, and it is now commonplace to start training with a large initial learning rate which is decreased during a subsequent annealing period. The large initial learning rate is thought to aid generalization (Y. Li et al., 2019), while it has been shown that the learning rate is proportional to the variance of training-loss fluctuations around a local minima (Murata et al., 1996), providing a greater rationale for an annealing period. To define the learning rate at optimizer-step i, written as $\alpha^{(i)}$, denote the number of optimizer steps with constant learning rate as s_c, the decay rate r, and decay steps as s_d, then

$$\alpha^{(i)} = \begin{cases} \alpha_0 & \text{if } i < s_c \\ \alpha_0 \times r^{(i-s_c)/s_d} & \text{if } i \geq s_c \end{cases} \tag{3.5}$$

defines the learning rate schedule. The interpretation is that the learning rate will decrease by a factor of r every s_d steps when the exponential term is active. To convert between optimizer steps and dataset iterations or epochs, use

$$s = n_{\text{epochs}} \times n_{\text{updates per epoch}}.$$

It is commonplace to use train-validation-test splits when training and evaluating neural networks. The training set is used to perform optimizer updates, and the validation set is used to assess generalization and/or over-fitting during training, while the test set is completely unobserved until after training is complete. In this work, due to a limited number of available solutions, only training and validation groups are used, without cross-validation or hyperparameter tuning.

3.2. Single vehicle speed

Training information is given in Table 4 where s_t is the total number of training steps, and the values shown in parentheses are the corresponding number of epochs. Note that DVH M1 uses $\alpha_0 = 5 \times 10^{-5}$

Table 4. *Dataset and training options, where s_t is the total number of optimizer steps, and values in parentheses are dataset iterations or epochs*

Dataset options		Learning rate schedule	
vehicle speed	90 kph	α_0	1×10^{-3}
# training cases	99	r	0.1
# validation cases	25	s_t	1,386,000 (7000)
spatial batch size	54,000	s_c	198,000 (1000)
case batch size	1	s_d	594,000 (3000)

instead of $\alpha_0 = 1 \times 10^{-3}$ as reported in the table due to large instabilities which frequently resulted in training-loss divergence with higher learning rates.

DV-MLP and DVH models are trained using Adam optimizer (Kingma and Ba, 2014) with default options and single precision. In the case of DVH, both training methods described in Section 2.4.2 were employed, where DVH M1/M2 correspond to methods 1 and 2, respectively. Table 5 presents error metrics, revealing that DVH M2 performs best across the board, with both DVH methods having very similar and lesser errors as compared to DV-MLP. DVH M2 slightly outperforms DVH M1 while requiring significantly less training time, approximately 4.4 ms per optimizer step versus 40.5 ms, as shown in Table 3.

Each model's ability to infill or predict a training-group case at truncated spatial locations may be evaluated since training cases had points randomly dropped out or truncated during minibatch construction. Error metrics were calculated separately for the truncated and retained locations, with the case-wise mean-absolute error (MAE) for truncated locations plotted against that for retained locations in Figure 6 for DV-MLP and DVH M2. The dashed line is the identity line $y(x) = x$, meaning points which lie above the dashed line correspond to truncated-point errors which are larger than those for the retained points, and the opposite for points below the line. The points are generally located near the line, some above and some below, indicating that the performance is very similar between the retained and truncated groups. This is quantified in Table 6 where the mean error metrics for retained and truncated points are given separately, while the training errors reported in Table 5 include all points for the instance. This shows that the error metrics are nearly identical between the two groups for all models, demonstrating that the models are effective in this in-filling scenario.

Pressure-field predictions near a validation-group vehicle are shown in Figure 7 for DV-MLP and DVH M2, with predictions for all field variables over the full domain shown in the Appendix,

Table 5. *Summary of training and validation error metrics at a vehicle speed of 90 kph*

$\hat{\mathbf{q}}_i$	Network type	RMSE (train/val)	MRL2E (train/val)
	DV-MLP	11.5/12.6	4.75%/5.12%
p [Pa]	DVH M1	8.7/11.5	3.60%/4.61%
	DVH M2	**8.1/9.9**	**3.34%/3.99%**
	DV-MLP	0.66/0.74	2.85%/3.18%
u [m/s]	DVH M1	0.59/0.68	2.56%/2.92%
	DVH M2	**0.56/0.60**	**2.41%/2.59%**
	DV-MLP	0.41/0.49	9.94%/11.9%
v [m/s]	DVH M1	0.34/0.47	8.28%/11.3%
	DVH M2	**0.27/0.38**	**6.72%/9.22%**

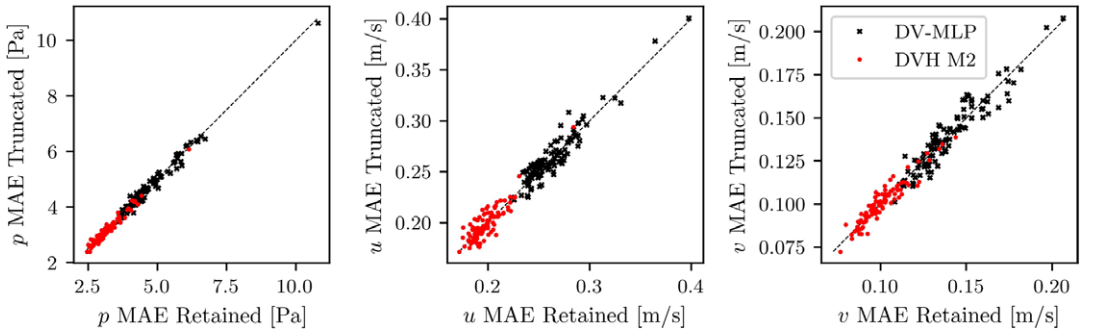


Figure 6. *Mean absolute error over points truncated from training cases versus those retained and used in training for DV-MLP and DVH M2 predictions for each field variable.*

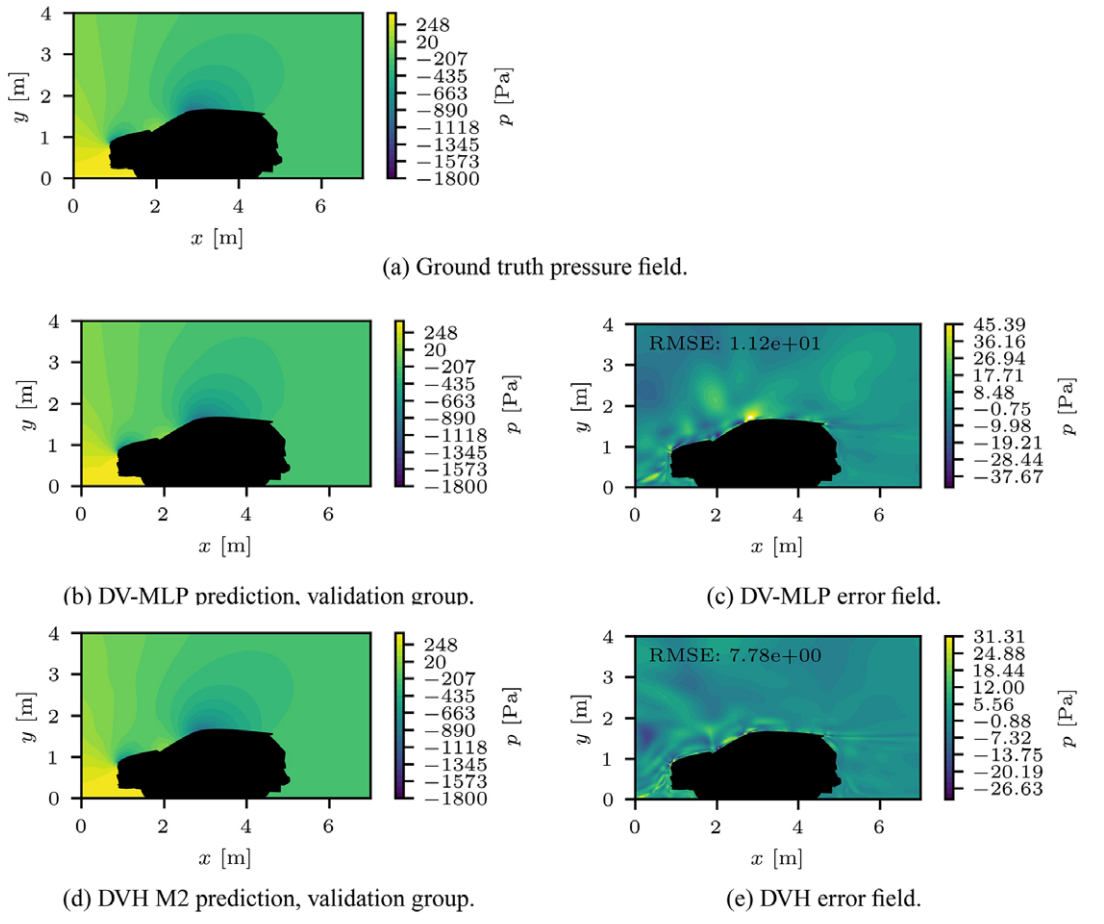
Table 6. *Comparing training-case error metrics between points that are retained and actually used in training versus those which are truncated*

$\hat{\mathbf{q}}_i$	Network type	RMSE (Retained/truncated)	MRL2E (Retained/truncated)
	DV-MLP	11.5/11.3	4.75%/4.66%
p [Pa]	DVH M1	8.7/8.5	3.60%/3.49%
	DVH M2	**8.1/8.0**	**3.34%/3.27%**
	DV-MLP	0.66/0.66	2.85%/2.86%
u [m/s]	DVH M1	0.59/0.60	2.55%/2.58%
	DVH M2	**0.56/0.56**	**2.41%/2.42%**
	DV-MLP	0.41/0.40	9.94%/9.88%
v [m/s]	DVH M1	0.34/0.34	8.27%/8.20%
	DVH M2	**0.27/0.28**	**6.72%/6.76%**

Section A.7. The error contour colorbars are limited to $\pm 4 \times$ RMSE centered on the average case error in order to see more fine-grain detail. Infrequent, comparably large errors obscure these details in many instances by skewing the colorbar, and points where the error is outside of this band are left white. The pressure field predictions match the ground truth well, with the main features captured including the high-pressure in front, the low-pressure due to flow acceleration on the roof, and the slowly recovering pressure in the wake seen in the full-domain plots. Some distortion of the contour lines is present in the predictions. The largest errors are seen near the vehicle surface, near the ground in front, and at locations along the free-shear layer and in the wake. The accuracy in predicted pressure drag coefficients are reported in Section 4.2.

3.3. Multiple speeds and generalization: low-data regime

The goal in situations such as surrogate-based design optimization is to obtain an accurate and generalizable surrogate using the least amount of training data and resources. The effect of the training dataset size on the generalization and convergence properties is explored by varying the number of available training cases. Two vehicle speeds of 90 and 130 kph are used, giving a total of 248 available solution instances. DV-MLP and DVH are compared while the number of training cases is varied from 5 to 199 instances, corresponding to training fractions ranging from 0.02 to 0.8. Training method 2 batch-by-case is used to train DVH models given that similar or better accuracy may be obtained as compared to fully-mixed training in a fraction of the time, as demonstrated for a single vehicle speed in Section 3.2. The dataset and training options are given in Table 7. The number of epochs remains fixed, resulting in varying



(a) Ground truth pressure field.

(b) DV-MLP prediction, validation group.

(c) DV-MLP error field.

(d) DVH M2 prediction, validation group.

(e) DVH error field.

Figure 7. *Validation-group instance (a) ground-truth pressure field, (b) DV-MLP prediction, (c) DV-MLP error, (d) DVH prediction, and (e) DV-MLP error. Error colorbars are limited to* $\pm 4 \times$ *RMSE centered on the average error for the instance.*

Table 7. *Dataset and training options, where* s_t *is the total number of optimizer steps, and values in parentheses are dataset-iterations or epochs*

Dataset options		Learning rate schedule	
Vehicle speeds	90, 130 kph	α_0	1×10^{-3}
Spatial Batch size	54,000	r	0.1
# Cases total	248	s_t	Equation 22 (3000)
Case batch size	1	s_c	Equation 22 (1000)
–	–	s_d	Equation 22 (1000)

learning rate schedule entries as the number of training cases is adjusted. Given a training fraction f_{train}, the number of training cases is chosen to be $\text{ceil}(f_{\text{train}} n_{\text{cases}})$, then the number of steps is given as

$$s = n_{\text{epochs}} \times \text{ceil}(f_{\text{train}} n_{\text{cases}}) \times n_{\text{batches per case}}. \tag{3.6}$$

The number of epochs are given in Table 7 in parentheses and $n_{\text{batches per case}} = 2$.

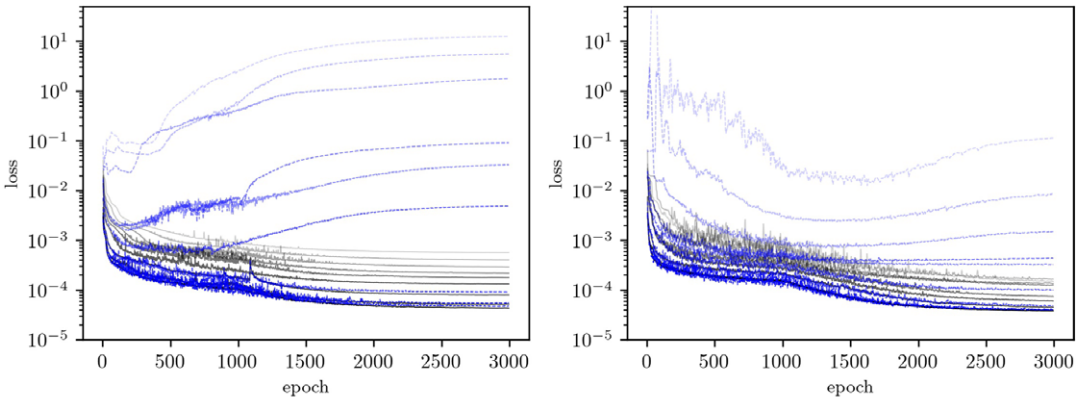


Figure 8. *Training (solid) and validation (dashed) losses during training as the amount of training data is varied from 5 to 199 cases, where darker lines correspond to more data, for (a) DV-MLP and (b) DVH. The curves have been smoothed using a moving average with a window length of 3 epochs for DV-MLP and 5 epochs for DVH.*

In the low-data regime, it is evident that neither model demonstrates strong generalization capabilities, leading to a substantial disparity between training and validation losses. The effect is more pronounced for DV-MLP, as illustrated by the training curves in Figure 8. In the figure, the solid lines represent the training loss, the dashed lines the validation loss, and darker lines correspond to a greater number of training cases. It is worth noting that the training loss reaching a plateau at a higher value with less data can be attributed, in part, to the fact that the number of epochs remains consistent rather than the number of optimizer updates. This should be kept in mind while interpreting the results.

During training the best weights based on the validation loss are saved along with the final weights. Figure 9a shows the MRL2E versus the number of training cases using the final weights for each flow quantity. In the very low-data regime, a large gap between training and validation losses is seen for both models, with DV-MLP exhibiting poorer performance. As the number of cases is increased, this gap is closed more quickly for DVH than DV-MLP. When 199 cases are used, the models perform similarly, with a slight advantage observed for DVH. Similar plots are generated using the validation best weights, as shown in Figure 9b. In this scenario, there is a smaller gap between the training and validation losses for each model. However, a persistent gap between DVH and DV-MLP remains across all training fractions considered, most notable for the y-velocity v.

Dimensional error metrics computed using the validation-best weights with 199 training instances, corresponding to the rightmost point in Figure 9b, are reported for both DV-MLP and DVH in Table 8. This shows that DVH performs best across the board, as expected. The RMSEs with two vehicle speeds are larger than those reported for a single speed in Table 5, and this is due in part to the 130 kph solutions having higher pressures and velocities than at 90 kph. To dig into this, the nondimensional and dimensional error metrics are broken out by vehicle speed instead of the training group for DVH in Table 9. This reveals that the nondimensional errors compare similarly for each vehicle speed but with 130 kph errors being slightly larger. The dimensional pressure errors for 90 kph lie between the training and validation errors for DVH M2 at a single speed, while the velocity component errors are slightly larger.

DVH pressure field predictions for a single vehicle shape at both 90 and 130 kph are shown in Figure 10, where the 90 kph case is from the training set while the 130 kph case is from the validation set. Good agreement between ground truth and prediction is seen at both speeds. Additional plots for velocities u and v are shown in the Appendix Section A.7.2 with similar good agreement.

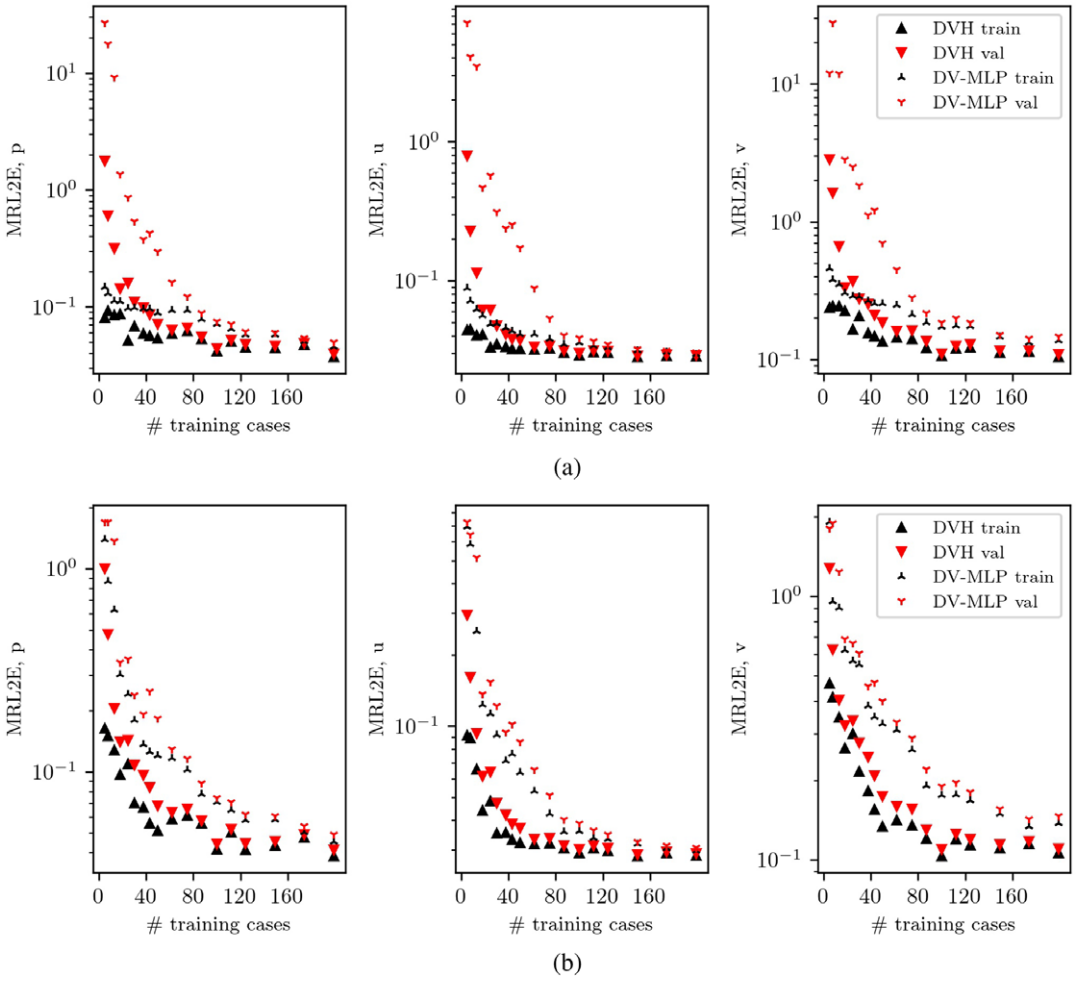


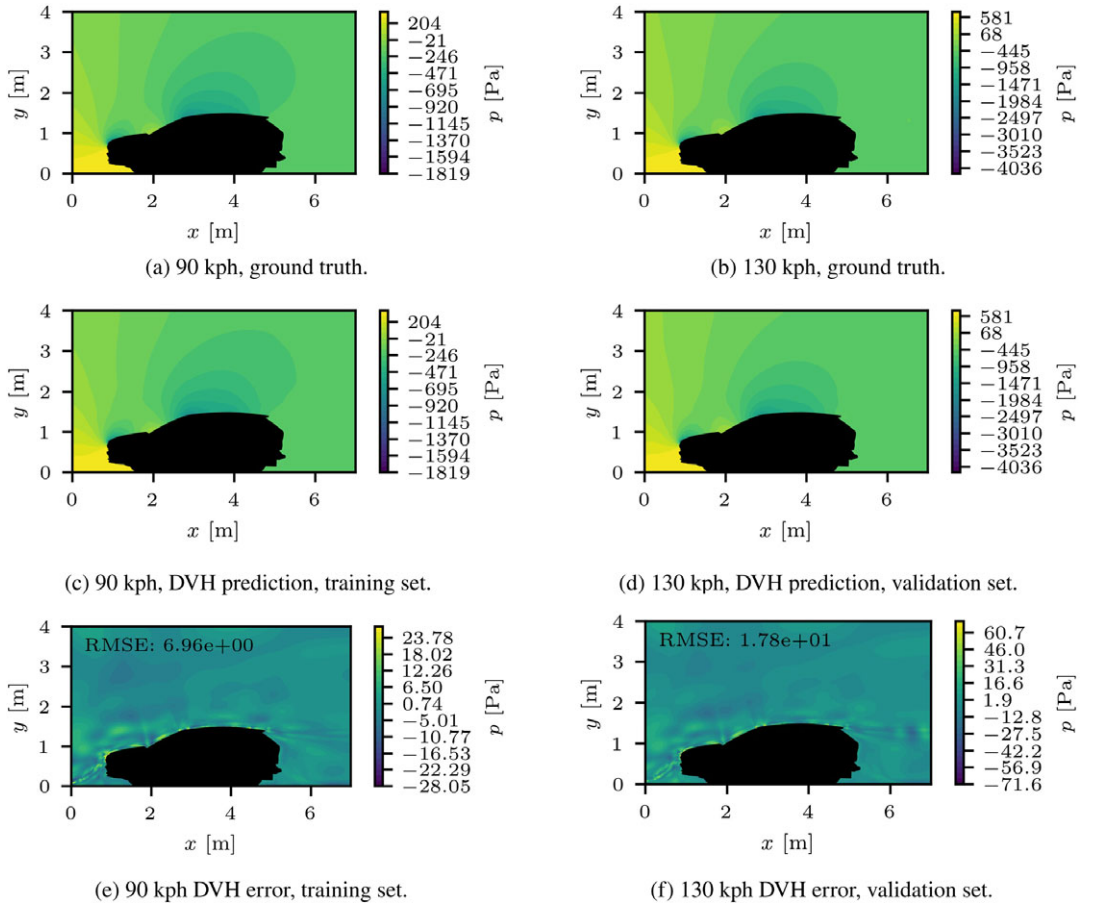
Figure 9. *Comparing trends in predictive error using mean-relative-L2-error (MRL2E), with (a) the final weights, and (b) the best weights per validation loss seen during training. The y-axis is not multiplied by 100%; therefore,* 10^{-1} *corresponds to 10% mean error in the state variable.*

Table 8. *Summary of training and validation error metrics for vehicle speeds of 90 and 130 kph with a training fraction of 0.80*

$\hat{\mathbf{q}}_i$	Network type	RMSE (train/val)	MRL2E (train/val)
p [Pa]	DV-MLP	16.4/18.3	4.37%/4.69%
	DVH M2	**12.9**/**14.8**	**3.43%**/**3.72%**
u [m/s]	DV-MLP	0.78/0.83	2.75%/2.87%
	DVH M2	**0.61**/**0.65**	**2.14%**/**2.23%**
v [m/s]	DV-MLP	0.59/0.63	11.7%/12.2%
	DVH M2	**0.46**/**0.49**	**9.06%**/**9.40%**

Table 9. *Comparing DVH nondimensional and dimensional error metrics computed for each vehicle speed separately with a training fraction of 0.80*

		RMSE		MRL2E	
$\hat{\mathbf{q}}_i$	Error type	90 kph	130 kph	90 kph	130 kph
p [−]/[Pa]	Nondimensional	$2.14 \cdot 10^{-2}$	$2.30 \cdot 10^{-2}$	3.37%	3.61%
	Dimensional	8.2	18.4	3.37%	3.61%
u [−]/[m/s]	Nondimensional	$1.99 \cdot 10^{-2}$	$2.02 \cdot 10^{-2}$	2.15%	2.18%
	Dimensional	0.50	0.73	2.15%	2.18%
v [−]/[m/s]	Nondimensional	$1.48 \cdot 10^{-2}$	$1.54 \cdot 10^{-2}$	9.04%	9.22%
	Dimensional	0.37	0.56	9.04%	9.22%



(a) 90 kph, ground truth.

(b) 130 kph, ground truth.

(c) 90 kph, DVH prediction, training set.

(d) 130 kph, DVH prediction, validation set.

(e) 90 kph DVH error, training set.

(f) 130 kph DVH error, validation set.

Figure 10. *Pressure field ground truth, DVH prediction, and errors at 90 and 130 kph for the same vehicle shape.*

4. Numerical Experiments II: effect of random fourier features

Spectral bias is a noted difficulty in training neural networks, where low-frequency signal content is learned more quickly and readily than high-frequency content. Consequently, the networks exhibit a bias toward low-frequency signal content (Rahaman et al., 2019). This issue may be addressed though the use

of random Fourier features (Tancik et al., 2020) or positional encoding techniques (Mildenhall et al., 2021). With these methods, the input coordinates are processed by sinusoidal terms of varying frequency before being fed into the MLP. Given network inputs $\mathbf{x}$, a Fourier-feature mapping is written as

$$\gamma(\mathbf{x}) = \left[a_1 \cos\left(2\pi \mathbf{b}_1^T \mathbf{x}\right), a_1 \sin\left(2\pi \mathbf{b}_1^T \mathbf{x}\right), \ldots, a_m \cos\left(2\pi \mathbf{b}_m^T \mathbf{x}\right), a_m \sin\left(2\pi \mathbf{b}_m^T \mathbf{x}\right)\right]^T, \tag{4.1}$$

where coefficients a_i, frequency-vectors $\mathbf{b}_i$, and their number m are parameters of the mapping. It has been shown that the simple strategy of setting $a_i = 1$ and drawing each $\mathbf{b}_i$ randomly from a sampling distribution is an effective strategy in practice, with the width σ of the sampling distribution having much greater importance than the distribution shape (Tancik et al., 2020). This width has a major impact on the convergence and generalization properties of the network, with underfitting observed for σ "too small" and overfitting observed for σ "too large." Positional-encoding strategies are generally a special case of Fourier features (Rahimi and Recht, 2007) where the frequencies are specified as a geometric progression and applied along each input dimension x_i separately (Mildenhall et al., 2021; Vaswani et al., 2017). This may be written as

$$\gamma(x_i) = \left[\cos\left(2^0 \pi x_i\right), \sin\left(2^0 \pi x_i\right), \ldots, \cos\left(2^{m-1} \pi x_i\right), \sin\left(2^{m-1} \pi x_i\right)\right], \tag{4.2}$$

where m may be set independently along each input axis. Another similar alternative includes the specific weight-initialization of sin-activated SIREN networks (Sitzmann, Martel, et al., 2020). A downside to random Fourier features is that the width σ and the number of features m must generally be found by trial and error or hyperparameter tuning. In all experiments, a zero-mean truncated isotropic Gaussian sampling distribution is used, $\mathbf{b}_i \sim \mathcal{N}(\mathbf{0}, \sigma \mathbf{I})$, with $\sigma = 3$ for all results here. Samples greater than 2σ from the mean are discarded. The number of features is selected to be $m = 256$, the same number of features chosen in Tancik et al., 2020, despite the smaller hidden dimension of $H = 50$ used in experiments here.

4.1. Model architectures and training options

The options and architectures of Section 3.1 are used again here but with the insertion of a random Fourier feature layer after the input layer of the main network. Note that the Fourier layer does not contain any trainable parameters; the sampled $\mathbf{b}_i$ are fixed. Applying this to DVH is straightforward, and the same set of random Fourier features are used for all main networks. However, Fourier features are not used in the hypernetwork. For DV-MLP, Fourier features are applied only to spatial inputs $\mathbf{x}'$. The resulting output $\gamma(\mathbf{x}')$ is then concatenated with $\boldsymbol{\mu}$ and passed into the MLP. Naively feeding all inputs through the Fourier features resulted in poorly performing models, see Section A.8 of the Appendix.

Insertion of a random Fourier layer with $m = 256$ results in models with more weights as compared to a model without a random Fourier layer. This is because $\dim(\gamma(\mathbf{x}')) = 2m \gg \dim(\mathbf{x}')$, leading to many more parameters in the first layer of the MLP. Models are profiled with the addition of the Fourier-feature layer, as shown in Table 10, with the number of trainable parameters given in Table 11 as compared against Table 2. Double precision is not considered here due to the much greater training time. Additionally only DVH training method 2 batch-by-case is considered due to the lessened training time and resources.

Table 10. *Profiling DV-MLP and DVH models, which use random Fourier features, with single and mixed precision*

Type	Precision	Δt step [ms]	Δt compute [ms]	Max. Mem. [GB]
DV-MLP FF	Single	5.0 (0.1)	4.2	0.85
DV-MLP FF	Mixed	4.1 (0.2)	2.7	0.71
DVH M2 FF	Single	5.4 (0.1)	4.5	0.55
DVH M2 FF	Mixed	5.1 (0.5)	3.1	0.41

Table 11. *Number of trainable parameters for DV-MLP and DVH models for all Fourier-feature results of Section 4, where* $H = H_L = 50$, $L_m = L_H = 5$

	# Trainable Weights	
Method	Single speed	Multiple speeds
DV-MLP	36,403	36,453
DVH	1,846,803	1,846,853

4.2. Single vehicle speed

DV-MLP and DVH using random Fourier features are first trained on a single vehicle speed using the same dataset and training options as given in Table 4. Error metrics are reported in Table 12, with the corresponding percentage improvement in RMSE compared to the models without Fourier features indicated in parentheses. With the use of Fourier features, DVH and DV-MLP exhibit similar performance, and DV-MLP is now best for several entries. For DV-MLP, this is a rather large improvement as the errors are roughly halved for a 35% increase in training step time (from 3.7 ms to 5.0 ms) when using single precision. DVH shows large but less significant improvements, with the training step time increased by 23% (from 4.4 to 5.4 ms) when using single precision.

The pointwise absolute error probability distributions with and without Fourier features are visualized using kernel density estimates, computed using the `FFTKDE` function of the python library KDEpy (Odland, 2018) using the Silverman method for kernel bandwidth selection. Other implementations were found to be very slow by comparison due to the large number of points: ∼11.1 million training and ∼2.8 million validation mesh points per vehicle speed. These are shown in Figure 11(a–c) for DV-MLP and Figure 11(d–f) for DVH. In all instances, the absolute error distributions are narrowed when using Fourier features, meaning smaller errors are more prevalent.

The MRL2E in computing the pressure drag coefficient is shown in Table 13 for both the Fourier models of this section and the non-Fourier models of Section 3.2. DVH M2 FF shows the smallest overall percent error over both the training and validation groups. In general, the Fourier models perform better than the non-Fourier models, with the lone exception of the DVH M2 training group errors.

4.3. Multiple speeds and generalization: low-data regime

The impact of Fourier features on generalization in the low-data regime with multiple vehicle speeds is studied here in an analogous fashion to Section 3.3. Dataset and training options of Table 7 apply. Figure 12a shows the training and validation MRL2E for each output quantity as the number of training instances is varied. As before, in the very-low data regime, there is a large gap between training and

Table 12. *Summary of training and validation error metrics at a vehicle speed of 90 kph for models using a Fourier-feature layer*

		RMSE		MRL2E	
$\hat{\mathbf{q}}_i$	Network type	Train	Val	Train	Val
p [Pa]	DV-MLP FF	5.55 (51.7%)	7.37 (41.3%)	2.31%	2.96%
	DVH M2 FF	**4.53** (44.1%)	**7.26** (26.6%)	**1.88%**	**2.89%**
u [m/s]	DV-MLP FF	0.243 (63.3%)	**0.30** (59.5%)	1.05%	**1.29%**
	DVH M2 FF	**0.236** (57.8%)	0.32 (47.3%)	**1.02%**	1.37%
v [m/s]	DV-MLP FF	**0.20** (49.6%)	**0.27** (44.6%)	**5.04%**	**6.58%**
	DVH M2 FF	0.22 (21.1%)	0.31 (19.5%)	5.32%	7.40%

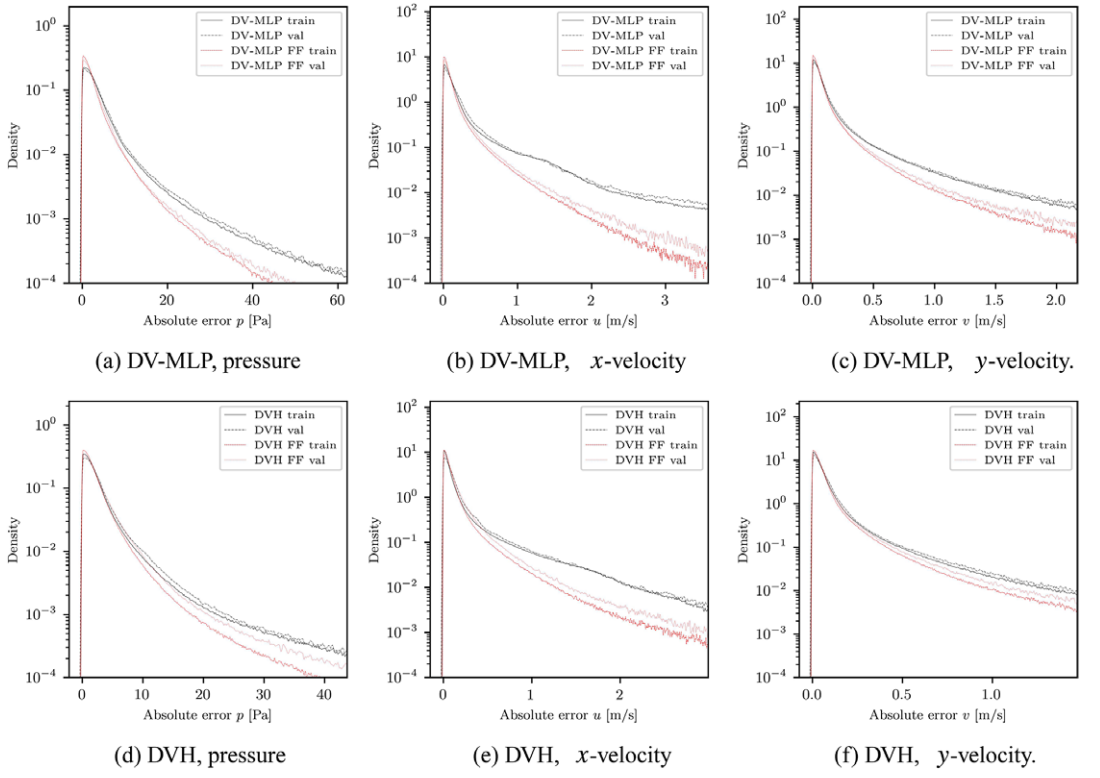


Figure 11. *DV-MLP (a–c) and DVH (d–f) single speed, pointwise absolute error probability distributions with and without random Fourier features, computed using Gaussian kernel density estimates using* `KDEpy` *python library .*

Table 13. *MRL2E (equivalent to mean-absolute-percent error) in predicting the pressure drag coefficient over the training and validation groups for non-Fourier and Fourier-based models for a single vehicle speed of 90 kph*

Network type	Train	Val
DV-MLP	2.66%	2.45%
DVH M1	2.28%	2.72%
DVH M2	1.26%	2.39%
DV-MLP FF	1.43%	1.50%
DVH M2 FF	**0.67%**	**1.30%**

validation losses, with DV-MLP showing a greater disparity. However, as the number of training instances is increased to around 40 the difference between DV-MLP and DVH is greatly reduced, and thereafter, their performance is very similar to one another. Figure 12b shows the corresponding plots using the best weights per validation loss. Without Fourier features there was a persistent gap between DVH and DV-MLP, but when Fourier features are used this gap is more or less eliminated.

Dimensional error metrics computed using the validation-best weights with 199 training instances, corresponding to the rightmost point in Figure 12b, are reported for both DV-MLP and DVH in Table 14. The percentage improvement in RMSE as compared to a model without Fourier features is given in

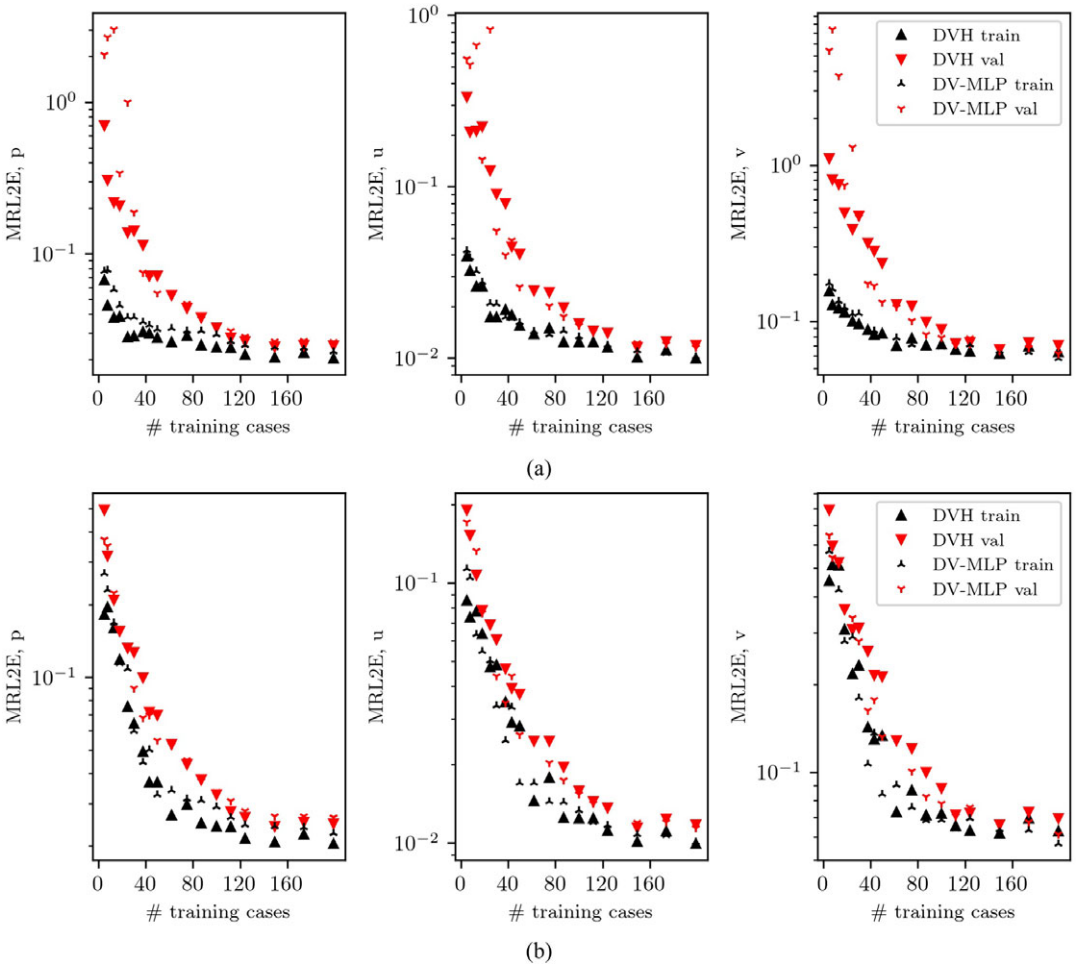


Figure 12. *Comparing trends in predictive error using mean-relative-L2-error (a), with (b) the final weights, and (c) the best weights per validation loss seen during training. The y-axis is not multiplied by 100%, therefore* 10^{-1} *corresponds to 10% mean error in the state variable.*

Table 14. *Summary of training and validation error metrics for vehicle speeds of 90 and 130 kph with a training fraction of 0.80, including the use of Fourier features. The percentage improvement when using Fourier features is given in parentheses*

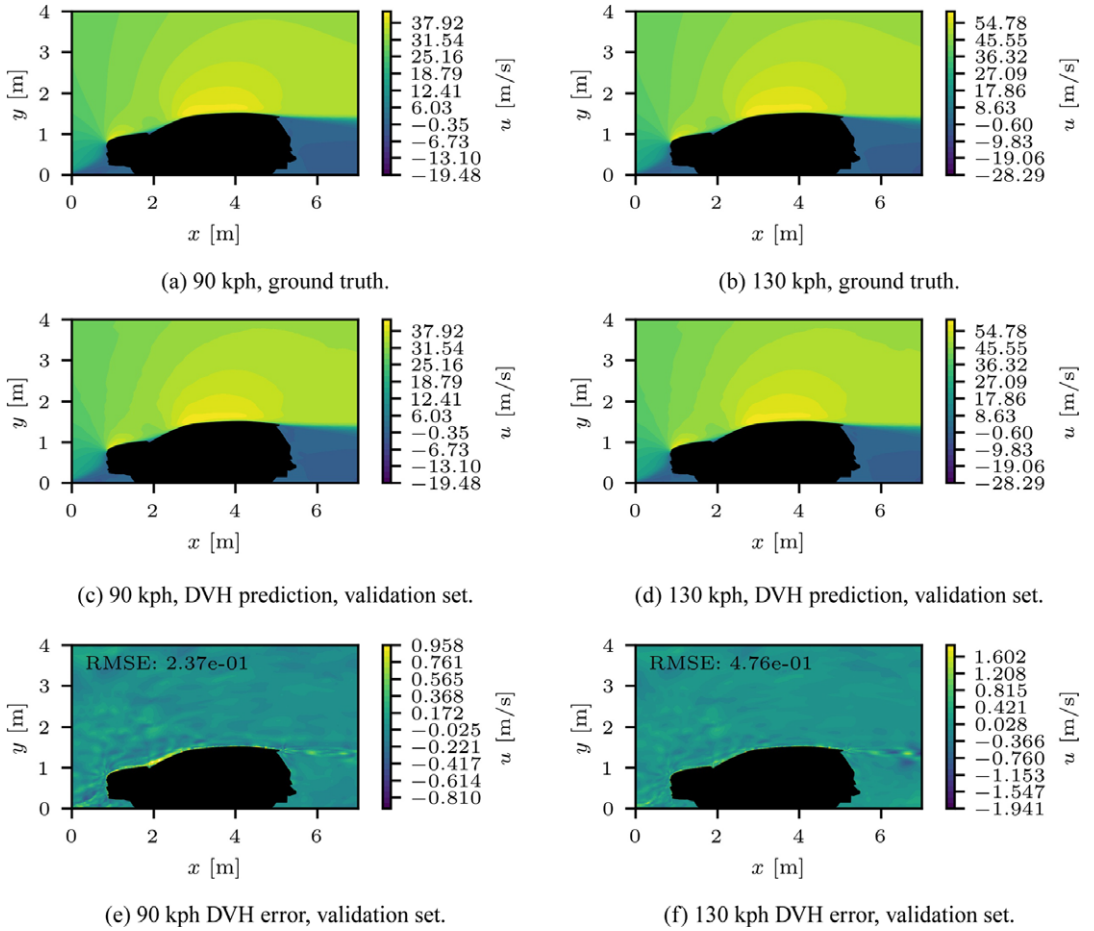
		RMSE		MRL2E	
$\hat{\mathbf{q}}_i$	Network type	Train	Val	Train	Val
p [Pa]	DV-MLP FF	8.4 (48.7%)	10.2 (44.5%)	2.26%	2.63%
	DVH M2 FF	**7.6** (40.9%)	**9.7** (34.4%)	**2.05%**	**2.48%**
u [m/s]	DV-MLP FF	0.29 (63.2%)	**0.33** (59.7%)	1.01%	**1.14%**
	DVH M2 FF	**0.28** (53.4%)	0.34 (47.3%)	**0.99%**	1.17%
v [m/s]	DV-MLP FF	**0.29** (51.2%)	**0.31** (49.9%)	**5.70%**	**6.13%**
	DVH M2 FF	0.32 (30.7%)	0.36 (26.5%)	6.28%	6.95%

parentheses. As with a single vehicle speed, substantial improvements are seen for both DV-MLP and DVH, with the effect larger for DV-MLP. Similarly to the single-speed scenario, DV-MLP now performs best for several entries as well.

DVH predictions and errors of the x-velocity field at both speeds are shown in Figure 13, where neither speed is included in the training dataset. The predicted fields match the ground truth well and capture the dominant flow features including the small recirculating regions in front of the vehicle, acceleration and flow turning over the roof, and a decaying free-shear layer in the wake. Similar plots for the pressure and y-velocity predictions are given in the Appendix, Section A.9.

For further comparison, vertical line probes are placed near the vehicle, with one in front, one through the vehicle's highest point, and two in the wake. The probe in front of the vehicle is offset by 1 m, while those in the wake are offset by 1 and 3 m. The ground truth and DVH predictions with and without Fourier features are interpolated from mesh points to the line probe locations using the `griddata` function from the `scipy.interpolate` library, with the results shown in Figure 14. Generally, the line probe predictions match the ground truth well, though some oscillation is present in the predictions. This is most prevalent for the pressure probes in the vehicle wake, though the effect is more pronounced due to the x-axis limits.

Pointwise absolute error probability distributions for DV-MLP and DVH using Fourier features are visualized in Figure 15 using kernel density estimates, again computed using the `FFTKDE` function of the



(a) 90 kph, ground truth.

(b) 130 kph, ground truth.

(c) 90 kph, DVH prediction, validation set.

(d) 130 kph, DVH prediction, validation set.

(e) 90 kph DVH error, validation set.

(f) 130 kph DVH error, validation set.

Figure 13. *x-velocity ground truth, DVH prediction, and errors at 90 and 130 kph for the same vehicle shape, where neither instance was included in the training set.*

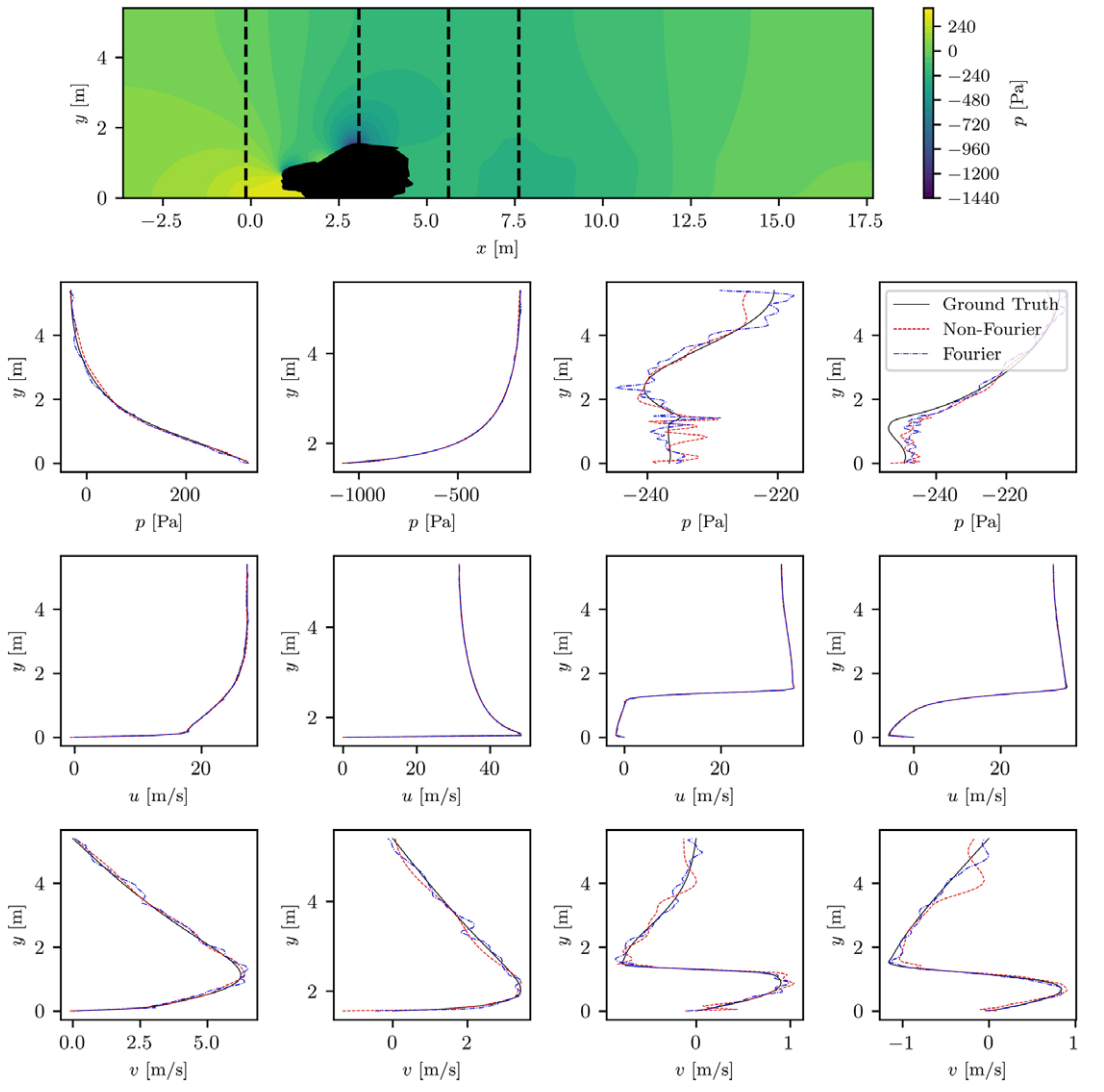


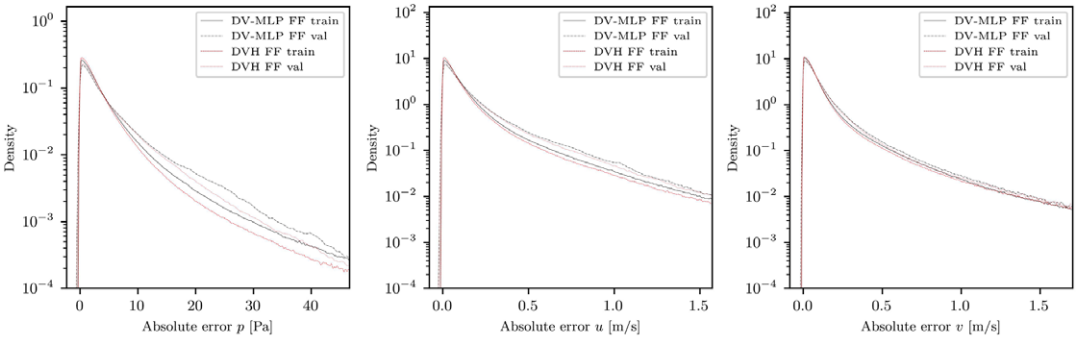
Figure 14. *Line probes comparing baseline and Fourier feature DVH predictions for a validation-group case at 90 kph.*

python library KDEpy (Odland, 2018) using the Silverman method for kernel bandwidth selection. The distribution shapes compare similarly for both network types, showing a peak and gradual trailing off as the errors increase.

The MRL2E in predicting the pressure drag coefficient using the Fourier models of this section and the non-Fourier models of Section 3.3 are shown in Table 15. In general, the Fourier models slightly outperform the non-Fourier models, with the lone exception being the validation set using DVH without Fourier features.

5. Summary and conclusions

The past few years have witnessed significant activity in the use of neural networks to develop surrogate representations of physical fields (e.g. Bhatnagar et al., 2019; Guo et al., 2016; Xu and



(a) Pressure (b) x-velocity (c) y-velocity.

Figure 15. *Comparing DV-MLP and DVH pointwise absolute error probability distributions using random Fourier features, computed using Gaussian kernel density estimates.*

Table 15. *MRL2E (equivalent to mean-absolute-percent error) in predicting the pressure drag coefficient over the training and validation groups for non-Fourier and Fourier-based models for multiple vehicle speeds of 90 and 130 kph*

Network type	Train	Val
DV-MLP	2.65%	3.30%
DVH	1.20%	**1.54%**
DV-MLP FF	1.90%	2.29%
DVH FF	**1.05%**	1.58%

Duraisamy, 2020) on a given discretized mesh. The present work seeks the development of surrogate models on unseen meshes, mesh topologies, and geometries as a *continuous field*, allowing learning and prediction on meshes with arbitrary discretization and topology. This is achieved through the use of coordinate-based neural networks, which map between low-dimensional spaces, inspired by a line of research involving scene and object representation for rendering tasks. Two methods of global conditioning were compared, namely concatenation-based conditioning as in DV-MLP, and conditioning the network weights through a hypernetwork as with DVH. Input features include spatially varying quantities, collected in $\mathbf{x}'$, and non-spatially-varying design variables, collected in $\boldsymbol{\mu}$. The spatial coordinates $\mathbf{x}$ are paired with a MDF evaluation $\phi(\mathbf{x};\boldsymbol{\mu})$ to implicitly encode the geometry, acting as a form of local conditioning.

The utilization of batch-by-case training for DVH models significantly reduced the computational cost associated with training. This approach led to a substantial decrease in step times and memory consumption, approximately by an order of magnitude, compared to fully-mixed training when considering a backend precision policy. This training method enabled DVH models with significantly more parameters to be trained in a comparable amount of time as DV-MLP models. The effectiveness of DVH and DV-MLP was evaluated in the context of a challenging vehicle aerodynamics problem, utilizing realistic vehicle shapes with solutions lying on unstructured meshes of variable topology. Baseline results indicated that DVH consistently outperformed DV-MLP by a few percentage points, while also demonstrating superior convergence and generalization properties in the low data regime.

By incorporating a random Fourier features layer to process the spatially varying inputs $\mathbf{x}'$, the RMSEs were significantly reduced by approximately 40–60% with greater improvements seen for DV-MLP as

compared to DVH. In this scenario, DV-MLP results were improved so substantially that the performance gap between DVH and DV-MLP was essentially closed, including the convergence and generalization properties in the low-data regime. Both DVH and DV-MLP exhibited strong generalization capabilities when multiple vehicle speeds were considered, with minimal disparities between training and validation errors. The pressure field errors were near 2%, x-velocity errors around 1%, and y-velocity errors around 6–7%. The pressure drag coefficients were also very well predicted, with the best models have an error of 1–2%. The main features of the flow fields were well-captured, with the largest errors often clustering in regions close to the vehicle, in the fine details of the grill, and along the free-shear layer of the wake. Line probes showed some oscillation in the network predictions compared to the ground truth, consistent with discrepancies seen in the contour levels. Several researchers have noted the effectiveness of positional encoding techniques in a variety of problem scenarios (Mildenhall et al., 2021; Vaswani et al., 2017; Zhong et al., 2019). The success of the techniques may be explained using Neural Tangent Kernel theory (Jacot et al., 2018), where the use of random Fourier features results in a stationary, shift-invariant kernel with a tunable bandwidth controlled by sampled frequency vectors $\mathbf{b}_i$ (Tancik et al., 2020). However, this result is in the context of dense, nearly uniform sampling of input coordinates without an additional signed-distance input, and further study is warranted in the current scenario with unstructured, non-uniform meshes.

The results suggest that both methods can be accurate and effective alternatives to CNNs for surrogate modeling of PDE solution fields over complex geometries and arbitrary mesh topologies. It is again emphasized that CNNs typically require a fixed grid topology and have a large memory footprint for 3D problems since the entire grid is an input. In contrast, coordinate-based networks take pointwise information and design variables as inputs, allowing model size and memory requirements to be decoupled from the solution field degrees-of-freedom.

Acknowledgements. Special thanks to Dr. Shaowu Pan of Rensselaer Polytechnic Institute who helped guide the early stages of this research and contributed to important discussion relating to memory requirements for CNNs.

Data availability statement. Code for model implementations may be found at https://github.com/jamesduv/DISMv2. The vehicle aerodynamics dataset is proprietary to General Motors and cannot be made public.

Author contribution. J.D.: Conceptualization, data curation, formal analysis, investigation, methodology, software, validation, visualization, writing—original draft. K.D.: Conceptualization, funding acquisition, methodology, project administration, resources, supervision, writing—review & editing.

Funding statement. This work is funded by General Motors, Inc. under a contract titled "Deep Learning and Reduced Order Modeling for Automotive Aerodynamics," and by Advanced Research Projects Agency-Energy (ARPA-E) DIFFERENTIATE program under the project "Multi-source Learning-accelerated Design of High-efficiency Multi-stage Compressor," in collaboration with Raytheon Technologies Research Center (RTRC). Computing resources were provided in part by the NSF via grant 1531752 MRI: Acquisition of Conflux, A Novel Platform for Data-Driven Computational Physics.

Competing interest. The authors declare none.

References

Abadi M, **Agarwal A**, **Barham P**, **Brevdo E**, **Chen Z**, **Citro C**, **Corrado G**, **Davis A**, **Dean J**, **Devin M**, **Ghemawat S**, **Goodfellow I**, **Harp A**, **Irving G**, **Isard M**, **Jia Y**, **Jozefowicz R**, **Kaiser L**, **Kudlur M**, … **Zheng X**. (2015) TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Software available from tensorflow.org]. https://www.tensorflow.org/

Arroyo CP, **Dombard J**, **Duchaine F**, **Gicquel L**, **Martin B**, **Odier N and Staffelbach G** (2021) Towards the large-eddy simulation of a full engine: Integration of a 360 Azimuthal degrees fan, compressor and combustion chamber. part i: Methodology and initialisation. *Journal of the Global Power and Propulsion Society* (May), 1–16. https://doi.org/10.33737/jgpps/133115

Baque P, **Remelli E**, **Fleuret F and Fua P** (2018) Geodesic convolutional shape optimization. *International Conference on Machine Learning*, *80*, 472–481.

Benner P Gugercin S and Willcox K (2015) A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Review 57*(4), 483–531. https://doi.org/10.1137/130932715

Bertinetto L, **Henriques JF**, **Valmadre J**, **Torr P and Vedaldi A** (2016) Learning feed-forward one-shot learners. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Red Hook, NY: Curran Associates, pp. 523–531.

Bhatnagar S, **Afshar Y**, **Pan S**, **Duraisamy K and Kaushik S** (2019) Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics 64*(2), 525–545. https://doi.org/10.1007/s00466-019-01740-0

Bruna J, **Zaremba W**, **Szlam A and LeCun Y** (2014) Spectral networks and locally connected networks on graphs. https://doi.org/10.48550/arXiv.1312.6203

Cai S, **Wang Z**, **Lu L**, **Zaki TA and Karniadakis GE** (2021) Deepm&mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *Journal of Computational Physics 436*, 110296. https://doi.org/10.1016/j.jcp.2021.110296

Chen Z and Zhang H (2019) Learning implicit fields for generative shape modeling. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA: CVF, pp. 5939–5948.

Davies T, **Nowrouzezahrai D and Jacobson A** (2020) On the effectiveness of weight-encoded neural implicit 3D shapes. *arXiv preprint arXiv:2009.09808*. https://doi.org/10.48550/arXiv.2009.09808

de Avila Belbute-Peres F, **Chen Y-f and Sha F** (2021) Hyperpinn: Learning parameterized differential equations with physics-informed hypernetworks. *The Symbiosis of Deep Learning and Differential Equations*, *690*.

Defferrard M, **Bresson X and Vandergheynst P** (2016) Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Red Hook, NY: Curran Associates, pp. 3844–3852.

Dolci V and Arina R (2016) Proper orthogonal decomposition as surrogate model for aerodynamic optimization. *International Journal of Aerospace Engineering 2016*. https://doi.org/10.1155/2016/8092824

Dumoulin V, **Perez E**, **Schucher N**, **Strub F**, **Vries H d**, **Courville A. and Bengio Y** (2018) Feature-wise transformations. *Distill*. https://distill.pub/2018/feature-wise-transformations. https://doi.org/10.23915/distill.00011

Duvenaud DK, **Maclaurin D**, **Iparraguirre J**, **Bombarell R**, **Hirzel T**, **Aspuru-Guzik A and Adams RP** (2015) Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the 28th International Conference on Neural Information Processing Systems—Volume 2*. Cambridge, MA: MIT Press, pp. 2224–2232.

Gao F, **Zhang Z**, **Jia C**, **Zhu Y**, **Zhou C and Wang J** (2022) Simulation and prediction of threedimensional rotating flows based on convolutional neural networks. *Physics of Fluids 34*(9), 095116. https://doi.org/10.1063/5.0113030

Gilmer J, **Schoenholz SS**, **Riley PF**, **Vinyals O and Dahl GE** (2017) Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning—Volume 70*. Sydney, NSW: JMLR, pp. 1263–1272.

Glorot X and Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington (Eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (pp. 249–256). PMLR. https://proceedings.mlr.press/v9/glorot10a.html

Guo X, **Li W, and Iorio F** (2016) Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 481–490. https://doi.org/10.1145/2939672.2939738

Ha D, **Dai A and Le QV** (2016) Hypernetworks. *arXiv preprint arXiv:1609.09106*. https://doi.org/10.48550/arXiv.1609.09106

Hasegawa K, **Fukami K**, **Murata T and Fukagata K** (2020) Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes. *Theoretical and Computational Fluid Dynamics 34*, 367–383.

He K, **Zhang X**, **Ren S and Sun J** (2016) Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* pp. 770–778. https://doi.org/10.1109/CVPR.2016.90

Henaff M, **Bruna J and LeCun Y** (2015) Deep convolutional networks on graph-structured data. https://doi.org/10.48550/arXiv.1506.05163

Jacot A, **Gabriel F and Hongler C** (2018) Neural tangent Kernel: Convergence and Generalization in Neural Networks, pp. 8580–8589. https://proceedings.neurips.cc/paper/8076-neural-tangen-kernel-convergence-and-generalization-in-neural-networks.pdf

Jaderberg M, **Simonyan K**, **Zisserman A and Kavukcuoglu K** (2015) Spatial transformer networks. *Advances in Neural Information Processing Systems 28*. https://proceedings.neurips.cc/paper_files/paper/2015/file/33ceb07bf4eeb3da587e268d663aba1a-Paper.pdf

Jia X, **De Brabandere B**, **Tuytelaars T and Gool LV** (2016) Dynamic filter networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon and R. Garnett (eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2016/file/8bf1211fd4b7b94528899de0a43b9fb3-Paper.pdf

Kashefi A, **Rempe D and Guibas LJ** (2021) A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries. *Physics of Fluids*, *33*(2).https://doi.org/10.1063/5.0033376

Kingma DP and Ba J (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. https://doi.org/10.48550/arXiv.1412.6980

Kipf TN and Welling M (2017) Semi-Supervised Classification with Graph Convolutional Networks. https://doi.org/10.48550/arXiv.1609.02907

Kovachki NB, **Li Z**, **Liu B**, **Azizzadenesheli K**, **Bhattacharya K**, **Stuart AM and Anandkumar A** (2023) Neural operator: Learning maps between function spaces with applications to PDES. *Journal of Machine Learning Research 24*(89), 1–97. http://jmlr.org/papers/v24/21-1524.html

Krizhevsky A, **Sutskever I and Hinton GE** (2012) Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25*. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Kutz JN, **Brunton SL**, **Brunton BW and Proctor JL** (2016) *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. SIAM.

Lecun Y, **Bottou L**, **Bengio Y and Haffner P** (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Li Y, **Wei C and Ma T** (2019) Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training neural Networks. https://proceedings.neurips.cc/paper/2019/file/bce9abf229ffd7e570818476ee5d7dde-Paper.pdf

Li Z Kovachki N, **Azizzadenesheli K**, **Liu B**, **Bhattacharya K**, **Stuart A and Anandkumar A** (2020) Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*. https://doi.org/10.48550/arXiv.2010.08895

Li Z, **Kovachki N**, **Azizzadenesheli K**, **Liu B**, **Stuart A**, **Bhattacharya K and Anandkumar A** (2020) Multipole graph neural operator for parametric partial differential equations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*. Red Hook, NY: Curran Associates.

Lu L, **Jin P**, **Pang G**, **Zhang Z and Karniadakis GE** (2021) Learning nonlinear operators via deep-onet based on the universal approximation theorem of operators. *Nature Machine Intelligence 3*(3), 218–229. https://doi.org/10.1038/s42256-021-00302-5

Mallya N, **Baqué P**, **Yvernay P**, **Pozzetti A**, **Fua P and Haussener S** (2023) Geodesic convolutional neural network characterization of macro-porous latent thermal energy storage. *ASME Journal of Heat and Mass Transfer 145*(5), 052902. https://doi.org/10.1115/1.4056663

Mescheder L, **Oechsle M**, **Niemeyer M**, **Nowozin S and Geiger A** (2019) Occupancy networks: Learning 3D reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4460–4470. https://openaccess.thecvf.com/content_CVPR_2019/papers/Mescheder_Occupancy_Networks_Learning_3D_Reconstruction_in_Function_Space_CVPR_2019_paper.pdf

Mildenhall B, **Srinivasan PP**, **Tancik M**, **Barron JT**, **Ramamoorthi R and Ng R** (2021) Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM 65*(1), 99–106. https://doi.org/10.1145/3503250

Mohan A, **Daniel D**, **Chertkov M and Livescu D** (2019) Compressed convolutional lstm: An efficient deep learning framework to model high fidelity 3D turbulence. *arXiv preprint arXiv:1903.00033*. https://api.semanticscholar.org/CorpusID:119353217

Murata N, **Müller K-R**, **Ziehe A and Amari S-i** (1996) Adaptive on-line learning in changing environments. Mozer M, Jordan M and Petsche T. (eds.), *Advances in Neural Information Processing Systems, 9*. https://proceedings.neurips.cc/paper_files/paper/1996/file/0e095e054ee94774d6a496099eb1cf6a-Paper.pdf

Odland T (2018) *Tommyod/kdepy: Kernel density estimation in python* (Version v0.9.10). Zenodo. https://doi.org/10.5281/zenodo.2392268

Ogoke F, **Meidani K**, **Hashemi A and Farimani AB** (2021) Graph convolutional networks applied to unstructured flow field data. *Machine Learning: Science and Technology 2*(4), 045020. https://doi.org/10.1088/2632-2153/ac1fc9

Pan S, **Brunton SL and Kutz JN** (2023) Neural implicit flow: A mesh-agnostic dimensionality reduction paradigm of spatio-temporal data. *Journal of Machine Learning Research 24*(41), 1–60. http://jmlr.org/papers/v24/22-0365.html

Park JJ, **Florence P**, **Straub J**, **Newcombe R and Lovegrove S** (2019) Deepsdf: learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 165–174. https://doi.org/10.1109/CVPR.2019.00025

Perez E, **Strub F**, **de Vries H**, **Dumoulin V and Courville A** (2018) Film: visual reasoning with a general conditioning layer. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. New Orleans, LA: AAAI Press, p. 483.

Pfaff T, **Fortunato M**, **Sanchez-Gonzalez A and Battaglia P W** (2020) Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*. https://doi.org/10.48550/arXiv.2010.03409

Qi CR, **Su H**, **Mo K and Guibas LJ** (2017) Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Honolulu, HI: IEEE, pp. 652–660.

Qi CR, **Yi L**, **Su H and Guibas LJ** (2017) Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems 30*. https://proceedings.neurips.cc/paper_files/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf

Rahaman N, **Baratin A**, **Arpit D**, **Draxler F**, **Lin M**, **Hamprecht F**, **Bengio Y and Courville A** (2019) On the spectral bias of neural networks. *International Conference on Machine Learning* 5301–5310. http://proceedings.mlr.press/v97/rahaman19a/rahaman19a.pdf

Rahimi A and Recht B (2007) *Random Features for Large-Scale Kernel Machines*. Platt J, Koller D, Singer Y and Roweis S (Eds.). *20*. https://proceedings.neurips.cc/paper_files/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf

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

Ronneberger O, **Fischer P and Brox T** (2015) u-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI2015*, pp. 234–241. https://doi.org/10.1007/978-3-319-24574-4_28

Salmoiraghi F, **Scardigli A**, **Telib H and Rozza G** (2018) Free-form deformation, mesh morphing and reduced-order methods: Enablers for efficient aerodynamic shape optimisation. *International Journal of Computational Fluid Dynamics 32*(4–5), 233–247. https://doi.org/10.1080/10618562.2018.1514115

Sanchez-Gonzalez A, **Godwin J**, **Pfaff T**, **Ying R**, **Leskovec J, and Battaglia P** (2020) Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 8459–8468. https://proceedings.mlr.press/v119/sanchez-gonzalez20a.html

Santos JE, **Xu D**, **Jo H**, **Landry CJ**, **Prodanovic M and Pyrcz MJ** (2020) Poreflow-net: A 3d convolutional neural network to predict fluid flow through porous media. *Advances in Water Resources 138*, 103539. https://doi.org/10.1016/j.advwatres.2020.103539

Schmid PJ (2022) Dynamic mode decomposition and its variants. *Annual Review of Fluid Mechanics 54*, 225–254. https://doi.org/10.1146/annurev-fluid-030121-015835

Simonyan K and Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. https://doi.org/10.48550/arXiv.1409.1556

Sitzmann V, **Chan E**, **Tucker R**, **Snavely N and Wetzstein G** (2020) Metasdf: Meta-learning signed distance functions. In Larochelle H, Ranzato M, Hadsell R, Balcan M and Lin H (eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc, pp. 10136–10147. https://proceedings.neurips.cc/paper_files/paper/2020/file/731c83db8d2ff01bdc000083fd3c3740-Paper.pdf

Sitzmann V, **Martel J**, **Bergman A**, **Lindell D and Wetzstein G** (2020) Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems 33*, 7462–7473. https://proceedings.neurips.cc/paper_files/paper/2020/file/53c04118df112c13a8c34b38343b9c10-Paper.pdf

Tancik M, **Srinivasan P**, **Mildenhall B**, **Fridovich-Keil S**, **Raghavan N**, **Singhal U**, **Ramamoorthi R**, **Barron J and Ng R** (2020) Fourier features let networks learn high frequency functions in low dimensional domains. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*. Red Hook, NY: Curran Associates, p. 632.

Tangsali K, **Krishnamurthy VR and Hasnain Z** (2021) Generalizability of convolutional encoder-decoder networks for aerodynamic flow-field prediction across geometric and physical-fluidic variations. *Journal of Mechanical Design 143*(5). https://doi.org/10.1115/1.4048221

Thuerey N, **Weißenow K**, **Prantl L and Hu X** (2020) Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal 58*(1), 25–36. https://doi.org/10.2514/1.J058291

Trask N, **Patel RG**, **Gross BJ and Atzberger PJ** (2019) Gmls-nets: A framework for learning from unstructured data. *arXiv preprint arXiv:1909.05371*. https://doi.org/10.48550/arXiv.1909.05371

Vaswani A, **Shazeer N**, **Parmar N**, **Uszkoreit J**, **Jones L**, **Gomez AN**, **Kaiser Ł and Polosukhin I** (2017) Attention is all you need. In Guyon I, Luxburg U, Bengio S, Wallach H, Fergus R, Vishwanathan S and Garnett R (eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Vincent P, **Larochelle H**, **Bengio Y and Manzagol P-A** (2008) Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*. pp. 1096–1103.

Wang S, **Wang H and Perdikaris P** (2021) Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science Advances 7*(40), eabi8605. https://doi.org/10.1126/sciadv.abi8605

Willcox K and Peraire J (2002) Balanced model reduction via the proper orthogonal decomposition. *AIAA Journal 40*(11), 2323–2330. https://doi.org/10.2514/2.1570

Xie Y, **Takikawa T**, **Saito S**, **Litany O**, **Yan S**, **Khan N**, **Tombari F**, **Tompkin J**, **sitzmann V and Sridhar S** (2022) Neural fields in visual computing and beyond. *Computer Graphics Forum 41*(2), 641–676. https://doi.org/10.1111/cgf.14505

Xu J and Duraisamy K (2020) Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics. *Computer Methods in Applied Mechanics and Engineering 372*, 113379. https://doi.org/10.1016/j.cma.2020.113379

Xu J, **Pradhan A and Duraisamy K** (2021) Conditionally parameterized, discretization-aware neural networks for mesh-based modeling of physical systems. *Advances in Neural Information Processing Systems 34*, 1634–1645. https://openreview.net/forum?id=0yMGEUQKd2D

Zhong ED, **Bepler T**, **Davis JH and Berger B** (2019) Reconstructing continuous distributions of 3d protein structure from cryo-em images. In *International Conference on Learning Representations*. https://api.semanticscholar.org/CorpusID:204806091

Appendix

A.1. Network scaling: further details

Considering a main network with L_m hidden layers, each with hidden dimension H, the total number of weights in the main network is

$$\dim(\theta_m) = (Hn_{x'} + H) + (L_m - 1)\left(H^2 + H\right) + \left(n_q H + n_q\right), \tag{A.1}$$

where the first term corresponds to the first hidden layer, the second term to all remaining hidden layers, and the final term the output layer. With low-dimensional input and output spaces, typically $n_{x'}, n_\mu, n_q << H$, thus quadratic H^2 terms dominate, leading to simplifying approximations

$$\dim(\theta_m) \approx (L_m - 1)H^2 + \mathcal{O}(H) \approx L_m H^2 + \mathcal{O}(H). \tag{A.2}$$

Next consider a hypernetwork with L_h hidden layers, where the first $L_h - 1$ layers also have a hidden dimension of H, while the final hidden layer has dimension H_L. The total number of weights in the hypernetwork is

$$\dim(\theta_h) = \left(Hn_\mu + H\right) + (L_h - 2)\left(H^2 + H\right) + (H_L H + H_L) + (\dim(\theta_m)H_L + \dim(\theta_m)), \tag{A.3}$$

where the terms are again in forward-propagation order. Retaining quadratic terms leads to the approximation

$$\dim(\theta_h) \approx (L_h - 2)H^2 + H_L H + \dim(\theta_m)(H_L + 1) + \mathcal{O}(H). \tag{A.4}$$

The third term in this expression typically dominates, corresponding to the hypernetwork output layer, revealing the important scaling consideration: a dense hypernetwork model will have roughly H_L times as many trainable weights as a similar DV-MLP model. Writing this proportionality, and substituting Equation A.2 gives the final result:

$$\dim(\theta_h) \propto \dim(\theta_m)H_L \propto L_m H^2 H_L, \tag{A.5}$$

corresponding to Equation 2.14 of the main text.

A.2. Model architecture summaries

A summary of the model architectures used is given in Table 16, with details on model inputs and outputs summarized in Table 17

In Section 3, the architectures are exactly as described in Table 16. When random Fourier features are applied in Section 4, the only difference is that spatial inputs $\mathbf{x}'$ are fed through a random Fourier layer before entering the main network. The number of weights in each scenario are given in the respective sections.

A.3. Neural network implementation details

Model implementation and training. All models are implemented via Python 3.X classes using Tensorflow v2.X (Abadi et al., 2015) and are trained using Nvidia RTX A6000 48 GPUs. The DV-MLP implenentation uses off-the-shelf Tensorflow-Keras sequential models, constructed using the Keras functional API. DVH models subclass Tensorflow-Keras Models (`tf.keras.Model`), overwriting the `call` method as required, depending on the batching method used. The DVH implementation uses a Tensorflow-Keras sequential model for the hypernetwork, required during class instantiation. Tensorflow-Keras models are useful as they provide high-level abstraction and contain easy-to-use functions for common tasks, such as training models and saving/loading network weights. All model weights are initialized using the Glorot-uniform weight-initialization scheme (Glorot and Bengio, 2010) and trained using calls to `tf.keras.Model.fit()` or custom training loops. Adam optimizer with default

Table 16. *Details on the structure of the networks used in the numerical experiments*

	Main/spatial network		Hypernetwork	
Method	# Hidden Layers	# Nodes/layer	#Hidden Layers	# Nodes/layer
DV-MLP	5	50	-	-
DVH	5	50	5	50

Table 17. *Summary of the network inputs and outputs for all sections*

	Main network		Hypernetwork	
Method	Inputs	Outputs	Inputs	Outputs
DV-MLP	$\mathbf{x}', \boldsymbol{\mu}$	$\hat{\mathbf{q}}$	-	-
DVH	$\mathbf{x}'$	$\hat{\mathbf{q}}$	$\boldsymbol{\mu}$	θ_m

settings is used for all numerical experiments. Mixed or single precision is used in training all models, and model checkpoints are used to save the model weights corresponding to the best training and validation losses obtained as training progresses. A simple mean-squared-error (MSE) loss function is used, written for a single minibatch as

$$\mathcal{L}(\theta) = \frac{1}{N_{tot}} \sum_{i=1}^{n_c} \sum_{m=1}^{n_i} \|\hat{\mathbf{q}}(\mathbf{x}_m^i, \boldsymbol{\mu}^i; \theta) - \mathbf{q}(\mathbf{x}_m^i, \boldsymbol{\mu}^i)\|_2^2, \tag{A.6}$$

where N_{tot} is the total number of mesh points in the minibatch.

$$N_{tot} = \sum_{i=1}^{n_c} n_i. \tag{A.7}$$

Normalization. All inputs and outputs are min-max normalized using the statistics of the training group, on a signal-by-signal basis, so that they lie approximately in the range $[0, 1]$. Some members of the validation group may be slightly above or below this range if they are smaller than the smallest element of the training set or larger than the largest element of the training set. Vectors $\mathbf{x}/\boldsymbol{\mu}/\mathbf{q}$ are normalized component-wise. The formula for computing the normalization is

$$\bar{r}^j = \frac{r^j - \min(\mathbf{r})}{\max(\mathbf{r}) - \min(\mathbf{r})}, \tag{A.8}$$

where r is an element of $\mathbf{x}$, $\mathbf{q}$, or $\boldsymbol{\mu}$ from either the training or validation group. Vector $\mathbf{r}$ is the collection of all instances of r from the training dataset, r^j is dimensional, and $\bar{r}^j$ is the normalized quantity. The predictions are fully dimensionalized for computing errors and plotting by rearranging Equation A.8 for r^j. If a nondimensional input or output is used, the nondimensionalization must must also be undone to achieve a fully dimensional result.

A.4. Error metrics

The error metrics of root-mean-squared error (RMSE) and mean-absolute error (MAE) are computed by averaging across all n_j mesh points in each case, and then averaging across all n_c cases. The RMSE for the kth component of the state is then defined as

$$\mathrm{RMSE}_k \triangleq \frac{1}{n_c} \sum_{j=1}^{n_c} \sqrt{\frac{1}{n_j} \sum_{m=1}^{n_j} \left(\hat{\mathbf{q}}_k(\mathbf{x}_m^j, \boldsymbol{\mu}^j; \theta) - \mathbf{q}_k(\mathbf{x}_m^j, \boldsymbol{\mu}^j)\right)^2}. \tag{A.9}$$

The mean-absolute error (MAE) is computed analogously as

$$\mathrm{MAE}_k \triangleq \frac{1}{n_c} \sum_{j=1}^{n_c} \frac{1}{n_j} \sum_{m=1}^{n_j} |\hat{\mathbf{q}}_k(\mathbf{x}_m^j, \boldsymbol{\mu}^j; \theta) - \mathbf{q}_k(\mathbf{x}_m^j, \boldsymbol{\mu}^j)|. \tag{A.10}$$

Both RMSE and MAE have units consistent with the predicted quantities, making them more intuitive than MSE alone. They provide similar measures, though the RMSE penalizes larger errors more than the MAE.

The mean-relative-L2-error (MRL2E) is also reported. To ease notation, gather all predictions for case j in matrix $\hat{\mathbf{Q}}^j \in \mathbb{R}^{n_j \times n_q}$, and all ground-truth in matrix $\mathbf{Q}^j \in \mathbb{R}^{n_j \times n_q}$. Then, $\hat{\mathbf{Q}}^j[:, k]$ is the full snapshot for the kth component of the predicted state, for the jth case. Then, the MRL2E for the kth state component may be expressed as

$$\mathrm{MRL2E}_k \triangleq \frac{1}{n_c} \sum_{j=1}^{n_c} \frac{\|\hat{\mathbf{Q}}^j[:, k] - \mathbf{Q}^j[:, k]\|_2}{\|\mathbf{Q}^j[:, k]\|_2} (\times 100\%). \tag{A.11}$$

The final multiplication by 100% is placed in parentheses as this operation is not always performed. The distinction is made clear in context by reporting the value with or without the percentage symbol.

A.5. RANS equations

The Reynolds Averaged Navier Stokes (RANS) equations are derived by ensemble averaging the Navier Stokes equations and substituting the Reynolds-decomposed state variables. This decomposition separates the state variables into mean (ensemble averaged) and fluctuating components $q = \overline{q} + q'$, where q is a generic state variable, $\overline{q}$ is the mean, and q' is the fluctuating component. In the incompressible limit, the steady RANS equations may be written as

$$\nabla \cdot \overline{\mathbf{u}} = 0 \tag{A.12}$$

$$\rho \overline{u}_j \frac{\partial \overline{u}_i}{\partial x_j} = \frac{\partial}{\partial x_i} \left[\mu \left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i} \right) - \overline{p} \delta_{ij} - \rho \overline{u_i' u_j'} \right], \tag{A.13}$$

where velocity vector $\mathbf{u} = u + v + w\hat{\mathbf{k}}$. Nondimensional flow quantities and inputs are used, denoted by a tilde $\tilde{\cdot}$. The Reynolds number Re is an important nondimensional similarity parameter describing the relative importance of inertial and viscous forces in a flow, and since it has no dimensional counterpart the tilde is omitted.

$$\mathrm{Re} = \frac{\rho|\mathbf{u}|L}{\mu} = \frac{|\mathbf{u}|L}{\nu} \tag{A.14}$$

Free-stream flow conditions are used to define the Reynolds number, using the vehicle length L_v as the length scale. The freestream dynamic pressure is written as $q_\infty = \frac{1}{2}\rho|\mathbf{u}_\infty|^2$, then the nondimensional form of other relevant quantities are given as

$$\tilde{x} = \frac{x}{L_v}; \tilde{y} = \frac{y}{L_v}; \tilde{\phi} = \frac{\phi}{L_v} \tag{A.15}$$

$$\tilde{p} = \frac{p - p_\infty}{q_\infty} \tag{A.16}$$

$$\tilde{u}_i = \frac{u_i}{|\mathbf{u}_\infty|}. \tag{A.17}$$

The nondimensional pressure of Equation A.16 is equivalent to the pressure coefficient c_p, and for incompressible flows the expression is further simplified by using the gauge pressure; taking $p_\infty = 0$. This is allowed because only derivatives of pressure enter into the incompressible momentum equation, Equation A.13. If a compressible flow were considered this equivalency would not hold, as pressure enters directly into the compressible energy equation.

A.6. Effect of spatial batch size

A spatial batch size of 54,000 points is used for all results presented in the main text, but this is a relatively large batch size compared to many other existing works. Here, the effect of varying the spatial batch size on non-Fourier DVH and DV-MLP model predictive performance and training time is quantified, with spatial batch sizes ranging from 512 to 108,000 points. Using a smaller spatial batch size results in a larger number of optimizer updates per epoch, and in what follows the number of epochs is scaled so that the number of optimizer updates is approximately equal to that used when the spatial batch size is 54,000. In fact, the number of required epochs is rounded up to ensure at least the same number of optimizer updates is performed when smaller batch sizes are considered. Additionally, both vehicle speeds are considered, the same piecewise learning rate schedule is used, and single-precision training is employed using the same model architectures as the main text.

Figure 16 shows the variation in training RMSE for each flow-quantity as the spatial batch size is varied. This figure shows that generally larger errors are seen using smaller spatial batch sizes for both DVH and DV-MLP, where the second-to-rightmost points correspond to the results in the main text. This point represents the best predictive performance for DVH, while DV-MLP shows the best results using a larger spatial batch size of 108,000 points. However, the trends in the plot are noisy as only a single replicant at each condition was trained. It is observed the DVH outperforms DV-MLP at every spatial batch size considered.

The models were also profiled using `Tensorboard` callbacks in an analogous fashion to Sections 3.1 and 4.1 of the main text. The optimizer step time versus spatial batch size is shown below in Figure 17. DVH has larger optimizer step times for all spatial batch sizes considered, but the variation is not as large as may be anticipated given the greater relative expense of evaluating the hypernetwork as the spatial batch size is decreased.

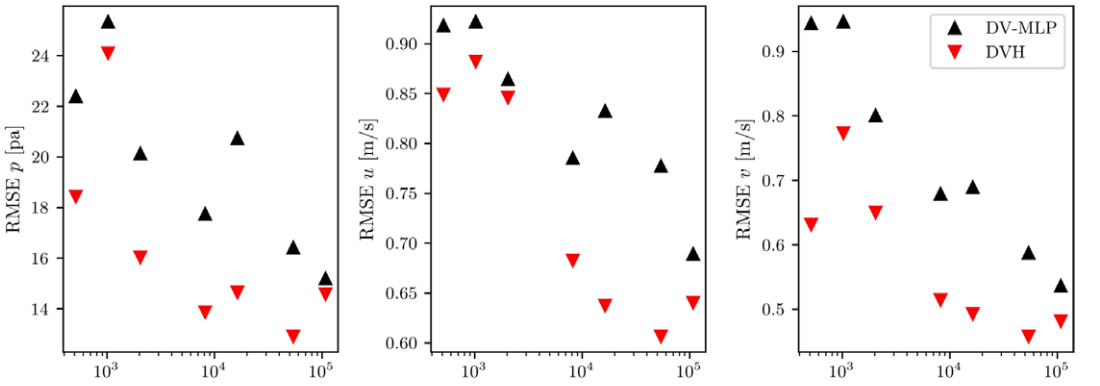


Figure 16. *The variation in training RMSE for each component of the predicted flow-field as the spatial batch size is varied. Non-Fourier DV-MLP and DVH models are considered.*

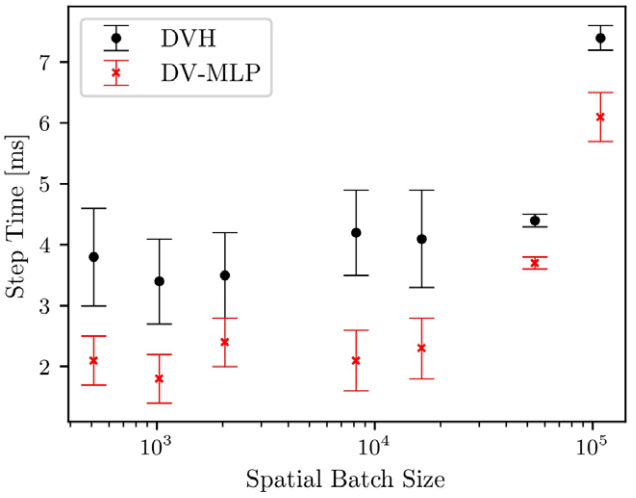


Figure 17. *The variation in optimizer step times as the spatial batch size is varied for non-Fourier DVH and DV-MLP models trained using single precision. The errors bars denote the standard deviation of the profiled step time.*

A.7. Baseline results: additional figures

A.7.1. Single vehicle speed

Full domain predictions for an unseen vehicle shape corresponding to Figure 7 are shown below.

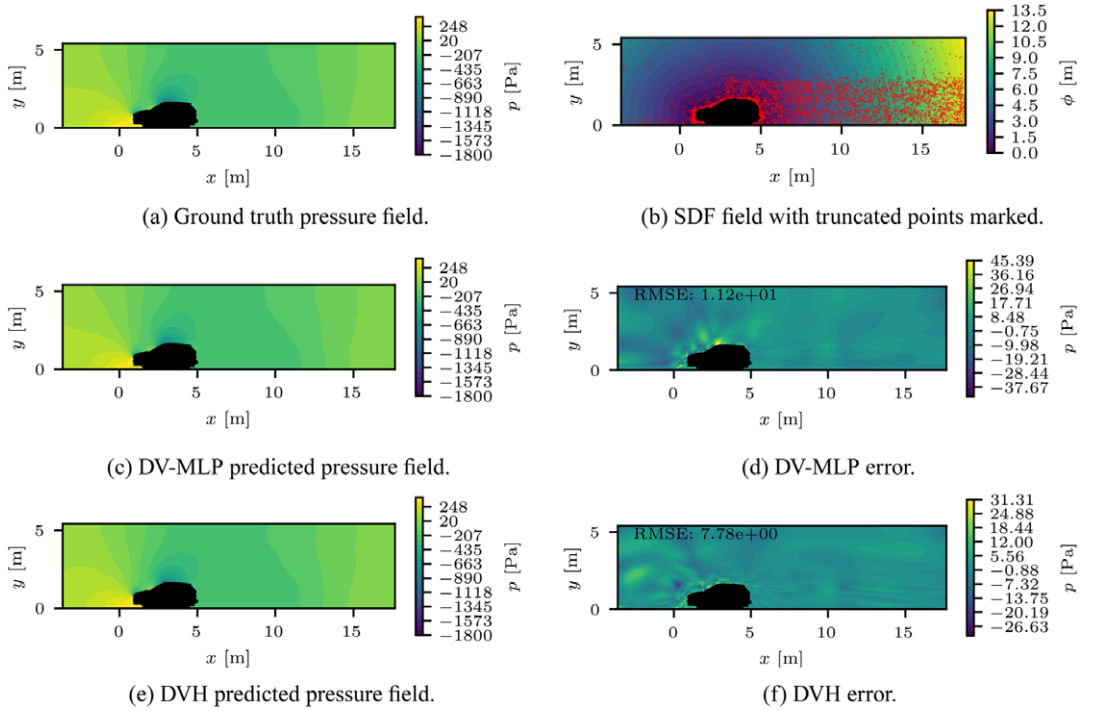
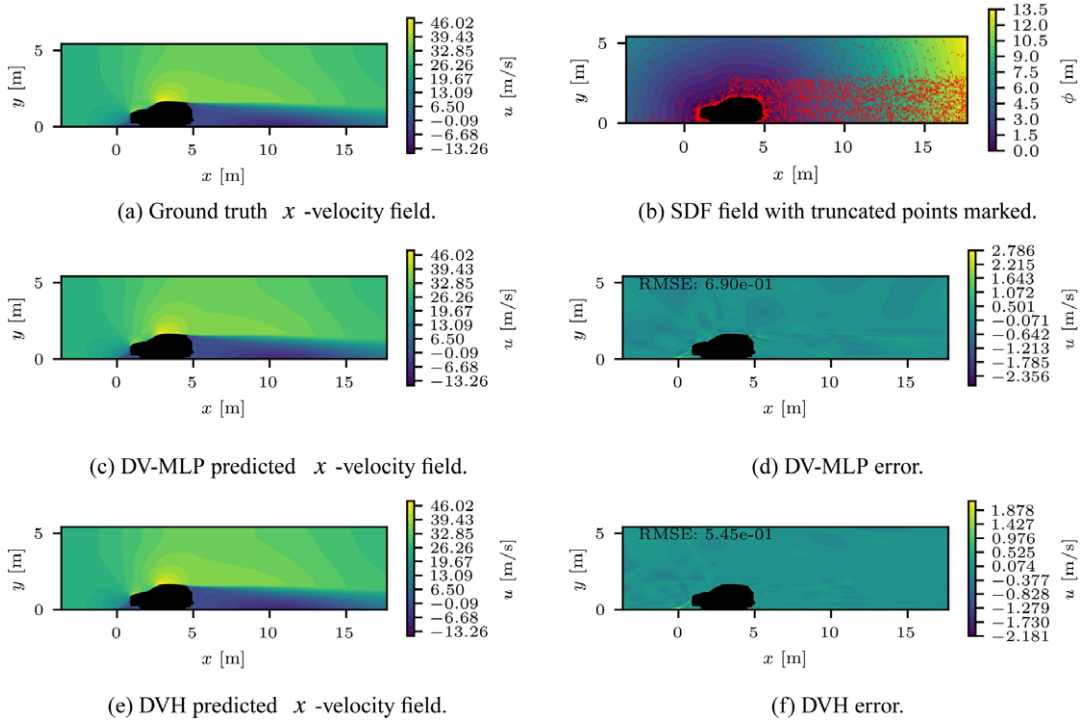


Figure 18. *Validation group pressure field predictions and errors.*



(a) Ground truth x -velocity field.

(b) SDF field with truncated points marked.

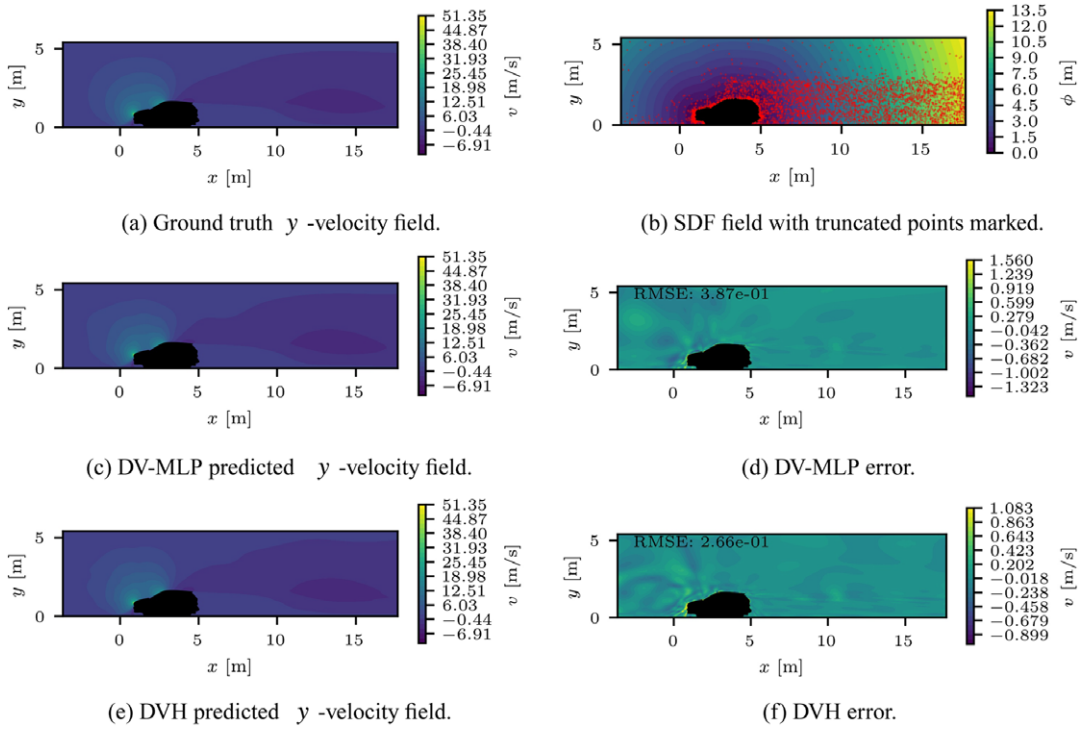
(c) DV-MLP predicted x -velocity field.

(d) DV-MLP error.

(e) DVH predicted x -velocity field.

(f) DVH error.

Figure 19. *Validation group x-velocity predictions and errors.*



(a) Ground truth y -velocity field.

(b) SDF field with truncated points marked.

(c) DV-MLP predicted y -velocity field.

(d) DV-MLP error.

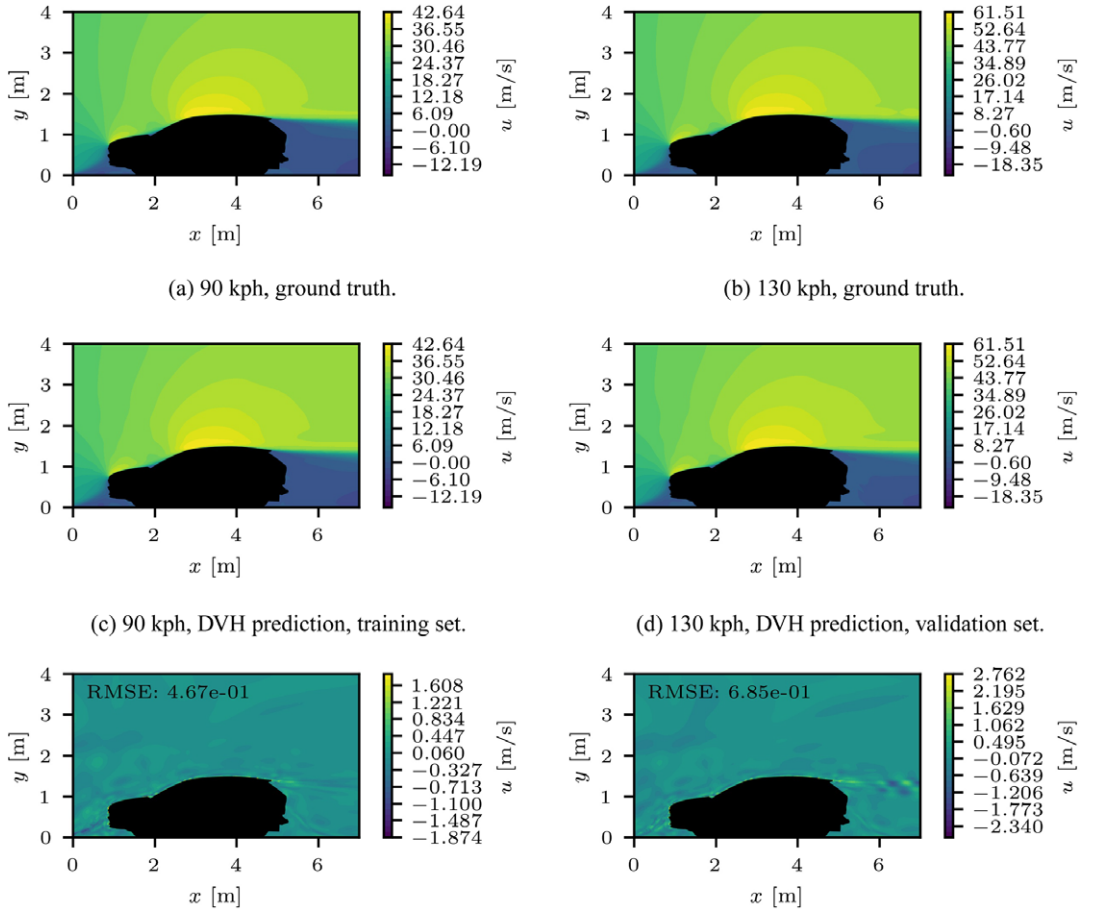
(e) DVH predicted y -velocity field.

(f) DVH error.

Figure 20. *Validation group y-velocity predictions and errors.*

A.7.2. Multiple vehicle speeds

Additional plots of ground truth and predicted x-velocity and y-velocity fields for a single vehicle shape at both speeds of 90 and 130 kph, corresponding to the same vehicle shape as Figure 10. As with the pressure field, the velocity component predictions closely match the ground truth at both speeds, with the largest errors seen near the vehicle surface and in the free-shear layer of the wake.



(a) 90 kph, ground truth.

(b) 130 kph, ground truth.

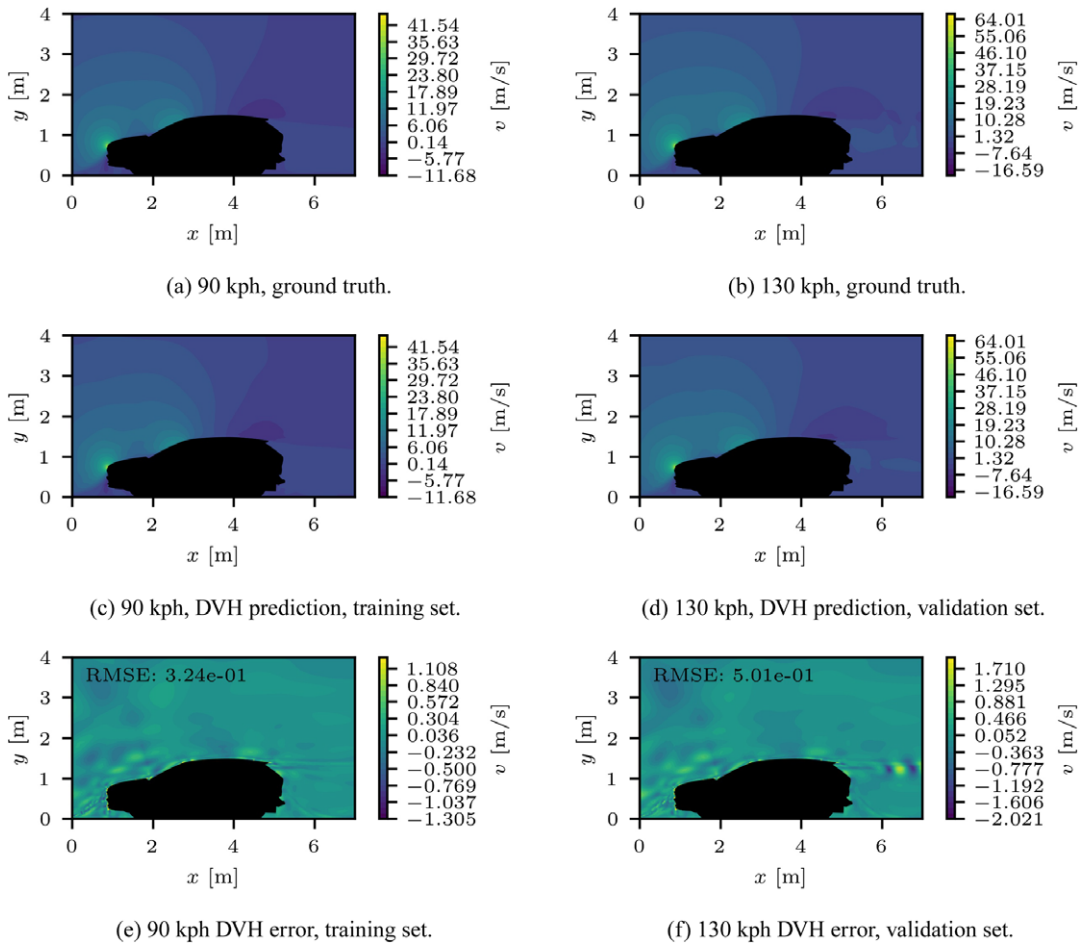
(c) 90 kph, DVH prediction, training set.

(d) 130 kph, DVH prediction, validation set.

(e) 90 kph DVH error, training set.

(f) 130 kph DVH error, validation set.

Figure 21. *x-velocity field ground truth, DVH prediction, and errors at 90 and 130 kph for the same vehicle shape.*



(a) 90 kph, ground truth.

(b) 130 kph, ground truth.

(c) 90 kph, DVH prediction, training set.

(d) 130 kph, DVH prediction, validation set.

(e) 90 kph DVH error, training set.

(f) 130 kph DVH error, validation set.

Figure 22. *y-velocity field ground truth, DVH prediction, and errors at 90 and 130 kph for the same vehicle shape.*

A.8. Fourier features for DV-MLP

Naively applying Fourier features to all DV-MLP inputs $\mathbf{x}'$ and μ results in poor convergence and generalization, as shown in Figure 23a. When applied to only the spatial inputs $\mathbf{x}'$, the models converge readily as shown in 23b. The network architecture and dataset are identical between the two models other than the difference in which inputs are processed by Fourier features.

A.9. Fourier features: Additional figures

Additional figures showing DVH pressure field and y-velocity field predictions at speeds of 90 and 130 kph, where neither instance was included in the training set. These correspond to the same case as shown in Figure 13.

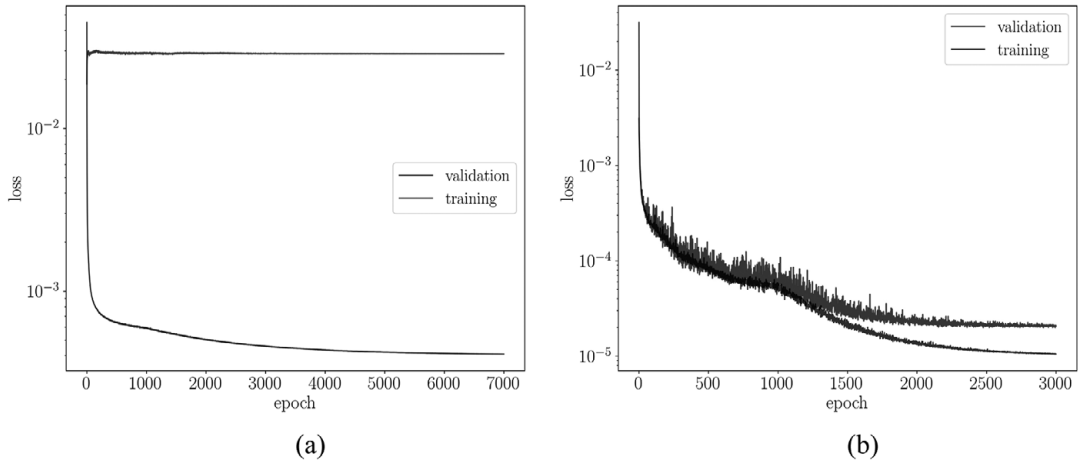
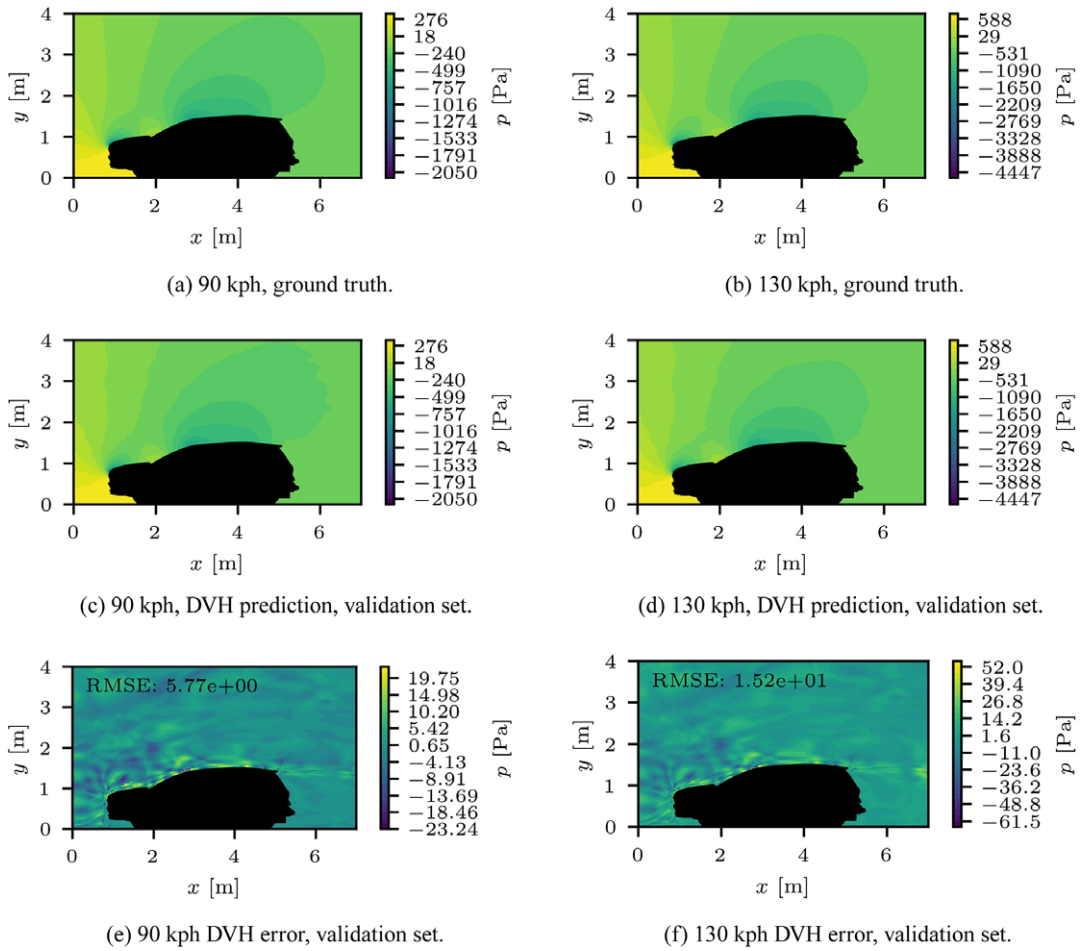


Figure 23. *Training curves for DV-MLP models, where the Fourier features are (a) applied to all inputs* $\mathbf{x}'$ *and* $\boldsymbol{\mu}$ *and (b) applied to only spatial inputs* $\mathbf{x}'$.



(a) 90 kph, ground truth.

(b) 130 kph, ground truth.

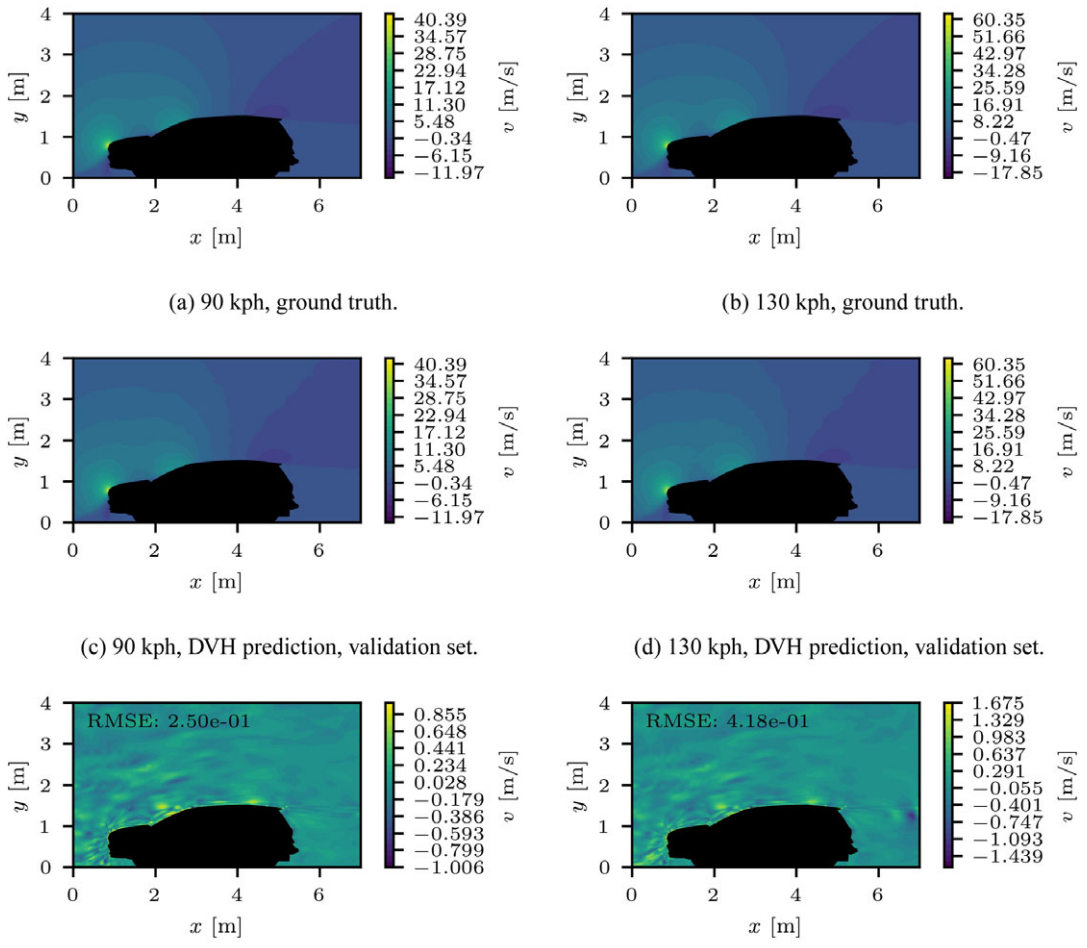
(c) 90 kph, DVH prediction, validation set.

(d) 130 kph, DVH prediction, validation set.

(e) 90 kph DVH error, validation set.

(f) 130 kph DVH error, validation set.

Figure 24. *Pressure field ground truth, DVH prediction, and errors at 90 and 130 kph for the same vehicle shape, where neither instance was included in the training set.*



(a) 90 kph, ground truth.

(b) 130 kph, ground truth.

(c) 90 kph, DVH prediction, validation set.

(d) 130 kph, DVH prediction, validation set.

(e) 90 kph DVH error, validation set.

(f) 130 kph DVH error, validation set.

Figure 25. *y-velocity field ground truth, DVH prediction, and errors at 90 and 130 kph for the same vehicle shape, where neither instance was included in the training set.*

Cite this article: Duvall J and Duraisamy K (2025). Discretization-independent surrogate modeling of physical fields around variable geometries using coordinate-based networks. *Data-Centric Engineering*, 6, e5. doi:10.1017/dce.2024.21