Convolutional Neural Networks for Steady Flow Approximation

Xiaoxiao Guo
University of Michigan
guoxiao@umich.edu

Wei Li
Autodesk Research
Wei.Li@autodesk.com

Francesco Iorio
Autodesk Research
Francesco.Iorio@autodesk.com

ABSTRACT

In aerodynamics related design, analysis and optimization problems, flow fields are simulated using computational fluid dynamics (CFD) solvers. However, CFD simulation is usually a computationally expensive, memory demanding and time consuming iterative process. These drawbacks of CFD limit opportunities for design space exploration and forbid interactive design. We propose a general and flexible approximation model for real-time prediction of non-uniform steady laminar flow in a 2D or 3D domain based on convolutional neural networks (CNNs). We explored alternatives for the geometry representation and the network architecture of CNNs. We show that convolutional neural networks can estimate the velocity field two orders of magnitude faster than a GPU-accelerated CFD solver and four orders of magnitude faster than a CPU-based CFD solver at a cost of a low error rate. This approach can provide immediate feedback for real-time design iterations at the early stage of design. Compared with existing approximation models in the aerodynamics domain, CNNs enable an efficient estimation for the entire velocity field. Furthermore, designers and engineers can directly apply the CNN approximation model in their design space exploration algorithms without training extra lower-dimensional surrogate models.

Keywords

Convolutional Neural Networks; Surrogate Models; Computational Fluid Dynamics; Machine Learning

1. INTRODUCTION

Computational fluid dynamics (CFD) analysis performs the calculations required to simulate the physical interaction of liquids and gases with surfaces defined by prescribed boundary conditions. This analysis process typically involves the solution of partial differential equations, such as the Navier-Stokes equations of fluid flow. The focus of our work is the analysis of non-uniform steady laminar flow (Figure 1).

KDD '16, August 13-17, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2939672.2939738

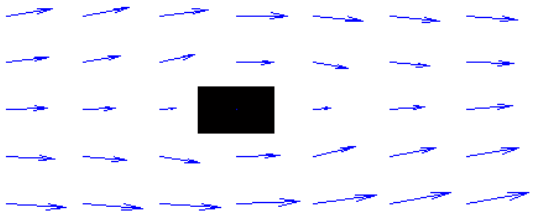

Figure 1: Non-uniform steady laminar flow for 2D geometry. The black box shows the object and the arrows show the velocity field in the laminar flow.

Laminar flow occurs when a fluid flows in parallel layers, with no disruption between the layers [2]. There are no cross-currents perpendicular to the direction of flow, nor eddies or swirls of fluids. Non-uniform steady flow occurs when the flow exhibits differences from point to point, but these differences do not vary over time. The non-uniform steady laminar flow is most often found at the front of a streamlined body and it is an important factor in flight and automobile design. [1]

While traditional CFD methods produce high-accuracy results, they are computationally expensive and the time required to obtain results is often measured in hours, or even days for complex prototypes. In many domains, CFD analysis becomes one of the most intensive and time consuming processes; therefore it is typically used only for final design validation.

During the early design stages, designers often need to quickly iterate over multiple design alternatives to make preliminary decisions and they usually do not require high-fidelity simulations. Significant efforts have been made to make conceptual design software environments interactive, so that designers can get immediate or real-time feedback for continuous design iterations. One of the alternatives to high accuracy CFD simulation is the use of fast approximation models, or surrogates.

In the design optimization practice, multiple design alternatives are explored, evaluated and optimized at the same time [1]. Guiding this process with fluid dynamics-based performance is difficult due to the slow feedback from conventional CFD solvers. The idea of speed-accuracy tradeoffs

[1]The geometry and velocity field representations and the approximation method would be much different if we attempted to model time-dependent unsteady flow and turbulent flow.

[6] also leads to practical design space exploration and design optimization in the aerodynamics domain by using fast surrogate models.

Data-driven surrogate models become more and more practical and important because many design, analysis and optimization processes generate high volume CFD physical or simulation data. Over the past years, deep learning approaches (see [5, 26] for survey) have shown great successes in learning representations from data, where features are learned end-to-end in a compositional hierarchy. This is in contrast to the traditional approach which requires the hand-crafting of features by domain experts. For data that have strong spatial and/or temporal dependencies, Convolutional Neural Networks (CNNs) [20] have been shown to learn invariant high-level features that are informative for supervised tasks.

In particular, we propose a general CNN-based approximation model for predicting the velocity field in 2D or 3D non-uniform steady laminar flow. We show that CNN based surrogate models can estimate the velocity field two orders of magnitude faster than a GPU-accelerated CFD solver or four orders of magnitude faster than a CPU-based CFD solver at a cost of a low error rate. Another benefit of using a CNN-based surrogate model is that we can collect training data from either physical or simulation results. The knowledge of fluid dynamics behavior can then be extracted from data sets generated by different CFD solvers, using different solution methods. Traditional high accuracy simulation is still an important step to validate the final design, but fast CFD approximation models have wide applications in the conceptual design and the design optimization process.

2. RELATED WORK

2.1 CFD Analysis

Computational methods have emerged as powerful techniques for exploring physical and chemical phenomena and for solving real engineering problems. In traditional CFD methods, Navier-Stokes equations solve mass, momentum and energy conservation equations on discrete nodes (the finite difference method, FDM) [11], elements (the finite element method, FEM) [29], or volumes (the finite volume method, FVM) [24]. In other words, the nonlinear partial differential equations are converted into a set of non-linear algebraic equations, which are solved iteratively. Traditional CFD simulation suffers from long response time, predominantly because of the complexity of the underlying physics and the historical focus on accuracy.

The Lattice Boltzmann Method (LBM) [22] was introduced to overcome the drawbacks of the lattice gas cellular automata. In LBM, the fluid is replaced by fractious particles. These particles stream along given directions (lattice links) and collide at the lattice sites. LBM can be considered as an explicit physical-based approximation method. The collision and streaming processes are local [23]. LBM emerged as an alternative powerful method for solving fluid dynamics problems, due to its ability to operate with complex geometry shapes and to its trivially parallel implementation nature [31, 21]. Since distinct particle motion can be decomposed over many processor cores or other hardware units, efficient implementations on parallel computers are relatively easy. LBM can also handle complex boundary conditions. However, LBM still requires numerous iterations and its convergence speed depends on the geometry boundaries. In this paper, we use LBM to generate all the training data for the CNNs, and we compare the speed of our CNN based surrogate models with both traditional CPU LBM solvers and GPU-accelerated LBM solvers.

Compared with physical-based simulation and approximation methods, our approach falls into the category of data-driven surrogate models (or supervised learning in machine learning). The knowledge of fluid dynamics behavior can be extracted from large data sets of simulation data by learning the relationship between an input feature vector extracted from geometry and the ground truth data output from a full CFD simulation. Then, without the computationally expensive iterations that a CFD solver requires to allow the flow to converge to its steady state, we can directly predict the steady flow behavior in a fraction of the time. The number of iterations and runtime in a CFD solver is usually dependent on the complexity of geometries and boundary conditions, but they are irrelevant to those factors in the prediction process. Note that only the prediction runtime of surrogate models has a significant impact on the design processes. The training time of the surrogate models is irrelevant.

2.2 Surrogate Modeling in CFD-based Design Optimization

The role of design optimization has been rapidly increasing and evolving in aerodynamics related fields, such as aerospace and architecture design [1]. CFD solvers have been widely used in design optimization (finding the best design parameters that satisfy project requirements) and design space exploration (finding multiple designs that satisfy the requirements). However, the direct application of a CFD solver in design optimization is usually computationally expensive, memory demanding and time consuming. The basic idea of using surrogates (approximation models) is to replace a high-accuracy, expensive analysis process with a less expensive approximation. CFD surrogate models can be constructed from physical experiments [30]. However, high-fidelity numerical simulation proves to be reliable, flexible, and relatively cheap compared with physical experiments (see [1] and references therein).

Various surrogate models have been explored for design processes, such as polynomial regression [12, 3], multiple adaptive regression splines (MARS) [10], Kriging [12, 16], radial basis function network [19], and artificial neural networks [7, 4]. The existing approaches only predict a restricted subset of the whole CFD velocity field. For example, predict the pressure on a set of points on an object surface [32]. Most existing approaches also depend on domain-specific geometry representations and can only be applied to low-order nonlinear problems or small scale applications [10, 18]. The models are usually constructed by modeling the response of a simulator on a limited number of intelligently chosen data points. Instead of modeling each individual low-dimensional design problem, we model the general non-uniform steady laminar CFD analysis solution, so that our model can be reused in multiple different, lower-dimensional optimization processes, and predict the whole velocity field.

2.3 Convolutional Neural Networks

Convolutional neural networks have been proven successful in learning geometry representations [27, 13]. CNNs

also enable per-pixel predictions in images. For example, FlowNet [8] predicts the optical flow field from a pair of images. Eigen, Puhrsch and Fergus [9] estimate depth from a single image. Recently, CNNs are applied on voxel to voxel prediction problems, such as video coloring [28].

Our approach applies CNNs to model large-scale nonlinear general CFD analysis for a restricted class of flow conditions. Our method obtains a surrogate model of the general high-dimensional CFD analysis of arbitrary geometry immersed in a flow for a class of flow conditions, and not a model based on predetermined low-dimensional inputs. For this reason, our model subsumes a multitude of lower-dimensional models, as it only requires training once and can be interrogated by mapping lower-dimensional problems into its high-dimensional input space. The main contribution of our CNN based CFD prediction is to achieve up to two to four orders of magnitude speedup compared to traditional Lattice Boltzmann Method (LBM) for CFD analysis at a cost of low error rates.

3. CONVOLUTIONAL NEURAL NETWORK SURROGATE MODELS FOR CFD

We propose a computational fluid dynamics surrogate model based on deep convolutional neural networks (CNNs). CNNs have been proven successful in geometry representation learning and per-pixel prediction in images. The other motivation of adopting CNNs is its memory efficiency. Memory requirement is a bottleneck to build whole velocity field surrogate models for large geometry shapes. The sparse connectivity and weight-sharing property of CNNs reduce the GPU memory cost greatly.

Our surrogate models have three key components. First, we adopt signed distance functions as a flexible and general geometric representation for convolutional neural networks. Second, we use multiple convolutional encoding layers to extract abstract and high-level geometric representations. Finally, there are multiple convolutional decoding layers that map the abstract geometric representations into the computational fluid dynamics velocity field.

In the rest of this section, we describe the key components in turn.

3.1 Geometry Representation

Geometry can be represented in multiple ways, such as boundaries and geometric parameters. However, those representations are not effective for neural networks since the vectors' semantic meaning varies. In this paper, we use a Signed Distance Function (SDF) sampled on a Cartesian grid as the geometry representation. SDF provides a universal representation for different geometry shapes and works efficiently with neural networks.

Given a discrete representation of a 2D geometry on a Cartesian grid, in order to compute the signed distance function, we first create the *zero level set*, which is a set of points (i, j) that give the geometry boundary (surface) in a domain $\Omega \subset R^2$:

$$Z = \{(i, j) \in R^2 : f(i, j) = 0\} \tag{1}$$

where f is the level set function s.t. $f(i, j) = 0$ if and only if (i, j) is on the geometry boundary, $f(i, j) < 0$ if and only if (i, j) is inside the geometry and $f(i, j) > 0$ if and only if (i, j) is outside the geometry.

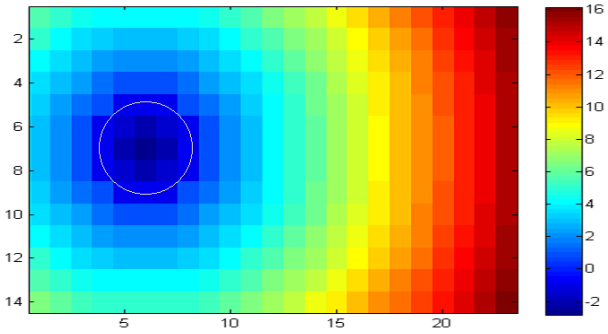

Figure 2: A discrete SDF representation of a circle shape (zero level set) in a 23x14 Cartesian grid. The circle is shown in white. The magnitude of the SDF values on the Cartesian grid equals the minimal distance to the circle.

A signed distance function $D(i, j)$ associated to a level set function $f(i, j)$ is defined by

$$D(i, j) = \min_{(i', j') \in Z} \left|(i, j) - (i', j')\right| \text{sign}(f(i, j)) \tag{2}$$

$D(i, j)$ is an oriented distance function and it measures the distance of a given point (i, j) from the nearest boundary of a closed geometrical shape Z, with the sign determined by whether *(i,j)* is inside or outside the shape. Also, every point outside of the boundary has a value equal to its distance from the interface (see Figure 2 for a 2D SDF example).

Similarly, given a discrete representation of a 3D geometry on a Cartesian grid, the signed distance function is

$$D(i, j, k) = \min_{(i', j', k') \in Z} \left|(i, j, k) - (i', j', k')\right| \text{sign}(f(i, j, k)) \tag{3}$$

A demonstration of 3D SDF is shown in Figure 3, where SDF equals to zero on the cube surface. In this paper, we use the Gudonov Method [14, 25] to compute the signed distance functions.

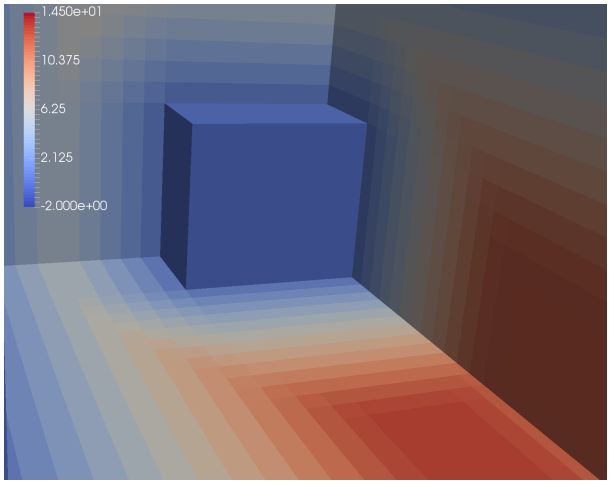

Figure 3: Signed distance function for a cube in 3D domain

The values of SDF on the sampled Cartesian grid not only provide local geometry details, but also contain additional

information of the global geometry structure. To validate the efforts of computing SDF values in our experiments, we compare the geometry representiveness of SDF and a simple binary representation, where a grid value is 1 if and only if it is within or on the boundary of geometry shapes. Such binary representations are easier to compute, but the values do not provide any global structure information. We empirically show that SDF is more effective in representing the geometry shapes for convolutional neural networks.

3.2 CFD Simulation

In this paper, we intend to leverage deep neural networks as real-time surrogate models to substitute traditional time-consuming Lattice Boltzmann Method (LBM) CFD simulation [22]. LBM represents velocity fields as regular Cartesian grids, or lattices, where each lattice cell represents the velocity field at that location. The LBM procedure updates the lattice iteratively, and usually the amount of required iterations for the flow field to stabilize from a cold start is large.

In LBM, geometry is represented by assigning an identifier to each lattice cell. This specific identifier defines whether a cell lies on the boundary or in the fluid domain. Figure 4 illustrates this using an example of a 2D cylinder in a channel. In general, the LBM lattice cell resolution does not necessarily equal to the SDF resolution. In this work, we use the same geometry resolution in the LBM simulation and in the SDF representation. Thus, for a geometry with $H \times W \times D$ SDF representation, its CFD velocity field is also $H \times W \times D$. A 2D velocity field consists of two components x and y to represent the velocity components in x-direction and y-direction respectively. The velocity field for 3D geometry has one additional z-component to represent the velocity component in the z-direction.

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	1	5	5	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	5	5	0	5	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	5	0	0	0	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	5	5	0	5	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	1	5	5	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Figure 4: 2D channel flow definition for LBM (1=fluid, 2=no-slip boundary, 3=velocity boundary, 4=constant pressure boundary, 5=object boundary, 0=internal)

3.3 Convolution Encoding

The encoding part takes the SDF, s, as an input, and stacked convolution layers extract geometry features directly from the SDF representation. For 2D geometry, s is a matrix that $s_{ij} = D(i,j)$ and the 2D convolution operations are used for encoding. For 3D geometry, s is a 3D tensor that $s_{ijk} = D(i,j,k)$, and the encoding uses 3D convolution. The

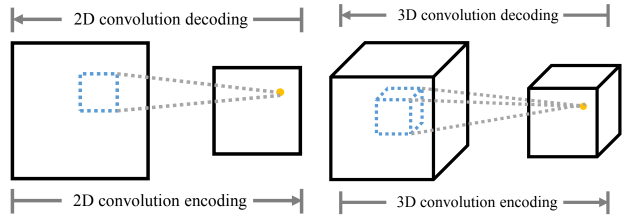

Figure 5: Convolution encoding and decoding.

encoded feature vector $h^{enc}(s)$ can be formulated as:

$$h^{enc}(s) = \text{ConvNN}(s) \tag{4}$$

where ConvNN(.) denotes the mapping from the SDF to a geometry representation vector using multiple convolution layers and a fully connected layer at the end, each of which is followed by a rectifier non-linearity (ReLU).

3.4 Convolutional Decoding

We use the 'inverse' operation of convolution, called deconvolution, to construct multiple stacked decoding layers. Deconvolution layers multiply each input value by a filter elementwise, and sum over the resulting output windows. In other words, a deconvolution layer is a convolution layer with the forward and backward passes reversed. 2D deconvolution maps 1×1 spatial region of the input to $w \times h$ convolution kernels. 3D deconvolution maps $1 \times 1 \times 1$ spatial region of the input to $w \times h \times d$ convolution kernels, as shown in Figure 5.

In our method, deconvolution operations are used to decode high-level features encoded and transformed by the encoding layers. The decoding procedure can be represented as:

$$\hat{t}^x(s) = \text{DeconvNN}(h^{enc}(s)) \tag{5}$$

where $\hat{t}^x(s)$ denotes the x-component prediction of the CFD velocity field. DeconvNN(.) is a mapping from the extracted geometry representation vector to CFD velocity field component using multiple deconvolution layers, each of which is followed by a rectifier non-linearity (ReLU). We decode the y-component (and z-component for 3D geometry) of the CFD velocity field in the same way.

3.5 Network Architecture and End-To-End Training

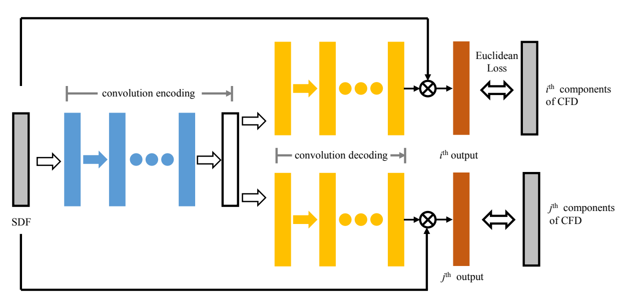

Figure 6: CNN based CFD surrogate model architecture

We use a shared-encoding and separated-decoding architecture for our CFD surrogate model in Figure 6. The

shared-encoding saves computations compared to separated-encoding alternatives. Unlike encoding layers, decoding layers are interfered by different velocity field components when a shared-decoding structure is used. In our shared-encoding and separated-decoding architecture, multiple convolutional layers are used to extract an abstract geometry representation, and the abstract geometry representation is used by multiple decoding parts to generate the various components of the CFD velocity field. There are two decoding parts for 2D geometry shapes and three for 3D geometry shapes. An alternative encoding structure is to use separated encoding layers for each decoding part. In the experiment section, we show that the prediction accuracy of the separated encoding structure is similar to the shared encoding structure.

Further, we condition our CFD prediction based on the SDF to reduce errors. By definition, the velocity inside and on the boundary of geometry shapes is zero. Thus, the velocity field values change dramatically on the boundary of geometry shapes. SDF itself fully represents the boundary information and our neural networks could condition the prediction by masking out the pixel or voxel inside or on the boundary of the geometry. Such conditioned prediction eases the training and improve the prediction accuracy. The final prediction of our model is $\hat{t^x} \otimes \mathbb{1}\{s > 0\}$, where $\mathbb{1}\{s > 0\}$ is a binary mask and has the same size as s. Its element is 1 if and only if the corresponding signed distance function value is greater than 0. $\otimes$ denotes element-wise product.

We train our model to minimize the mean squared error:

$$\begin{aligned}\mathbf{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} &|\hat{t^x}(s_n) \otimes \mathbb{1}\{s_n > 0\} - t^x(s_n)|^2 \\ &+|\hat{t^y}(s_n) \otimes \mathbb{1}\{s_n > 0\} - t^y(s_n)|^2\end{aligned} \tag{6}$$

A similar loss function is defined for 3D geometry shapes:

$$\begin{aligned}\mathbf{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} &|\hat{t^x}(s_n) \otimes \mathbb{1}\{s_n > 0\} - t^x(s_n)|^2 \\ &+|\hat{t^y}(s_n) \otimes \mathbb{1}\{s_n > 0\} - t^y(s_n)|^2 \\ &+|\hat{t^z}(s_n) \otimes \mathbb{1}\{s_n > 0\} - t^z(s_n)|^2\end{aligned} \tag{7}$$

4. EXPERIMENT SETUP

In the experiments, we evaluate our proposed method in both 2D and 3D geometry domains. In the 2D domain, we train the convolutional neural networks using a collection of 2D geometric primitives and evaluate the prediction over a validation dataset of 2D primitives and a test dataset of 2D car shapes (see Figure 7 for examples). We then train a separate network using a collection of 3D geometry shapes inside a channel (see Figure 8 for examples) and evaluate the prediction over a validation dataset of 3D geometry shapes.

We first empirically compare the representiveness of SDF with the simple binary representations and show that SDF outperforms binary representations in CFD prediction. Second, our CNN based surrogate model outperforms a patch-based linear regression baseline in predicting CFD for both 2D and 3D geometry shapes. Finally, we compare the time cost of our proposed method with traditional LBM solvers to highlight the speedup of our proposed method.

We begin by describing the details of the datasets, LBM data generation, convolutional neural network architectures, and training parameters.

4.1 Data and Preprocessing

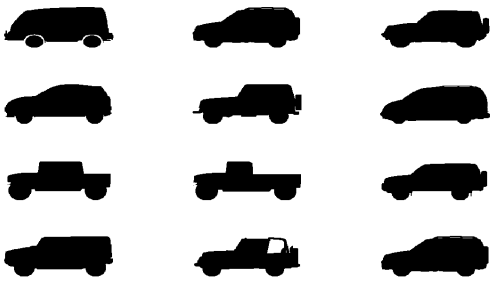

Figure 7: 2D car dataset visualization.

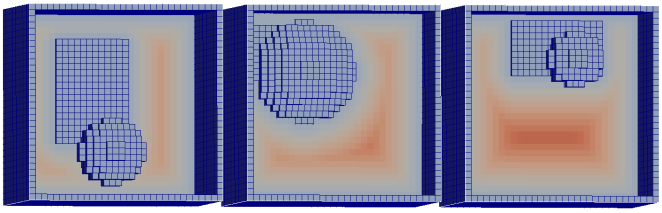

Figure 8: 3D channel dataset: two 3D primitives (a rectangular box and a sphere) in a 32-by-32-by-32 channel and the visualization of their SDF on the Y-Z plane

2D data set. The 2D training and validation datasets consist of 5 types of simple parametric geometric 2D primitives: triangles, quadrilaterals, pentagons, hexagons and dodecagons. Each set of primitives contains samples that are different in size, shape, orientation and location. The training dataset contains 100,000 samples (20,000 random samples for each kind of primitives), and the validation dataset contains 10,000 samples (2,000 random samples for each kind of primitives). The 2D primitives are projected into a 256-by-128 Cartesian grid and the LBM simulation computes a 256-by-128 velocity field as labels. We construct two kinds of 2D primitive datasets for different boundary conditions. In **Type I**, the primitives are always connected to the lower boundary. In **Type II**, the primitives are always above the lower boundary, and the gap ranges from 15 to 25 pixels.

2D car test data set. This data set is dedicated to test the generalization ability of our trained models. It contains various kinds of car prototypes, such as jeeps, vans and sport cars. The car samples are not visible in the training. Similar to 2D primitives, the car samples are projected into a 256-by-128 Cartesian grid and the LBM simulation computes a 256-by-128 velocity field. We also create two boundary conditions for the car samples.

3D data set. In order to understand whether interactions of flow with sets of multiple boundaries can be learned effectively, the 3D dataset contains 0.4 million sphere and rectangular prism pairs. Each sphere-prism pair is positioned in a 32×32×32 channel. Both the location and the parameters of each 3D object are randomly generated. Then a LBM simulation is performed on the 3D channel, where the fluid enters the channel along the *x axile*.

SDF preprocessing. We performed pixel/voxel-wise mean removal for SDF and scaled the resulting representation by 0.01 as inputs to the CNNs.

4.2 CFD Solver Configuration

Laminar flow occurs at low Reynolds numbers[2], where viscous forces are dominant, and is characterized by a fluid flowing in parallel layers, with no disruption between the layers. In all the experiments, we used a Reynolds number of 20. For our 2D experiments, we used OpenLB [15], a C++ Lattice Boltzmann library, to generate CFD velocity field. Because of the lack of GPU support in OpenLB, we ran each simulation on a single core on a Intel Xeon E5-2640V2 processor. For our 3D experiments, we used a proprietary GPU-accelerated LBM solver.

4.3 Network Architectures and Training

The architectures and hyper-parameters of our CNN surrogate models for 2D and 3D CFD are shown in Figure 9. In the CNN architectures, the encoding part is **shared** by decoding parts for different CFD dimensions. Alternative architectures could use a **separated** encoding part for each decoding part where each separated encoding part has the same architecture and hyper-parameters as the shared encoding part. We compare these two encoding paradigms in our experiments.

We implemented the CNNs using Caffe [17]. The parameters of the networks were initialized using Xavier method in Caffe. We used RMSProp with a batch size of 64 to optimize the parameters of the convolutional neural networks. The RMS decay parameter was set to 0.05. We set the initial learning rate to be 10^{-4} and multiplied the learning rate by 0.8 for every 2500 iterations.

5. EXPERIMENT RESULTS

We use the average relative error [1] in the CFD literature to evaluate our prediction. We first compute the difference between the predicted velocity field and the LBM generated velocity field. For a pixel/voxel, the relative error is defined as the ratio of the magnitude of the difference to the LBM generated velocity field. For 2D geometry shapes, the relative error at pixel (i, j) in n-th test case is represented as:

$$\mathrm{err}^n(i,j) = \frac{\sqrt{(t^{\hat{x}}_{ij}(s_n) - t^x_{ij}(s_n))^2 + (t^{\hat{y}}_{ij}(s_n) - t^y_{ij}(s_n))^2}}{\sqrt{t^x_{ij}(s_n)^2 + t^y_{ij}(s_n)^2}} \tag{8}$$

The average relative error of n-th data case is the mean value of all the pixels outside the geometry shape:

$$\mathrm{err}^n = \frac{\sum_i \sum_j \mathrm{err}^n(i,j) \mathbb{1}_{ij}(s_n > 0)}{\sum_i \sum_j \mathbb{1}_{ij}(s_n > 0)} \tag{9}$$

where $\mathbb{1}_{ij}(s_n > 0)$ is 1 if and only if (i, j) is outside of the geometry boundary of s_n. err^n can be extended to 3D geometry shapes straightforwardly by considering the z-component.

[2]In CFD modeling, the Reynolds number is defined as a non-dimensional ratio of the inertial forces to viscous forces and quantifies their relevance for the prescribed flow condition [2].

The average relative error over the test data set is the mean of all the test cases:

$$\mathrm{err} = \frac{1}{N} \sum_{n=1}^{N} \mathrm{err}^{\mathrm{n}} \tag{10}$$

5.1 Signed Distance Function vs. Binary Representation

One contribution of this paper is proposing using SDF as geometric representations for the CNN models. We evaluate the effectiveness of SDF geometric representations by comparing with the binary representations in the same CNN architectures. In the experiments, the binary representations use a 256 × 128 binary matrix to represent the geometry. Its element is 1 if and only if the corresponding position is on the geometry boundary or inside the geometry.

The results of prediction on 2D primitives are summarized in the Table 1.

DataSet	Encoding Type	SDF	Binary
2D Type I	shared	1.98%	54.60%
	separated	1.76%	55.25%
2D Type II	shared	2.86%	74.52%
	separated	3.08%	80.71%

Table 1: 2D CFD prediction results of SDF and binary representations.

The results show that the errors of using SDF are significantly smaller than the errors of using binary representations. Given that the deep neural networks are the same, the results imply that the SDF representations are more effective than the binary representations. Each value in the SDF representations carries a certain level global information while the values in binary representations only carry restricted local information. Due to such difference, binary representations may require more complex architectures to capture whole geometric information properly. We also compare the SDF representations to the first order gradients of the SDF representations, the prediction accuracy improvement is not significant, so we focus our contributions on using SDF because of its computation simplicity. The results also show that the prediction accuracy of shared encoding is similar to that of separated encoding.

5.2 Patch-wise Linear Regression Baseline

A patch-wise linear regression model is applied as a baseline to evaluate our proposed models. For 2D geometry shapes, the 256 × 128 SDF inputs are divided into 32 × 32 patches without overlap. The patch-wise linear regression model takes 32 × 32 patches as inputs and predicts 32 × 32 patches to construct the CFD velocity field for each component. The predicted 32 × 32 patches are concatenated without overlap to form the x-component or y-component of the CFD velocity field. For 3D geometry shapes, the 32 × 32 × 32 SDF inputs are divided into 8 × 8 × 8 patches without overlap, and the predicted 8 × 8 × 8 patches are then concatenated to form the velocity field.

The prediction results of CNNs and baseline are summarized in Table 2. The results show that our proposed CNN models outperform the patch-wise linear regression models. The CNN's prediction errors are significantly smaller than

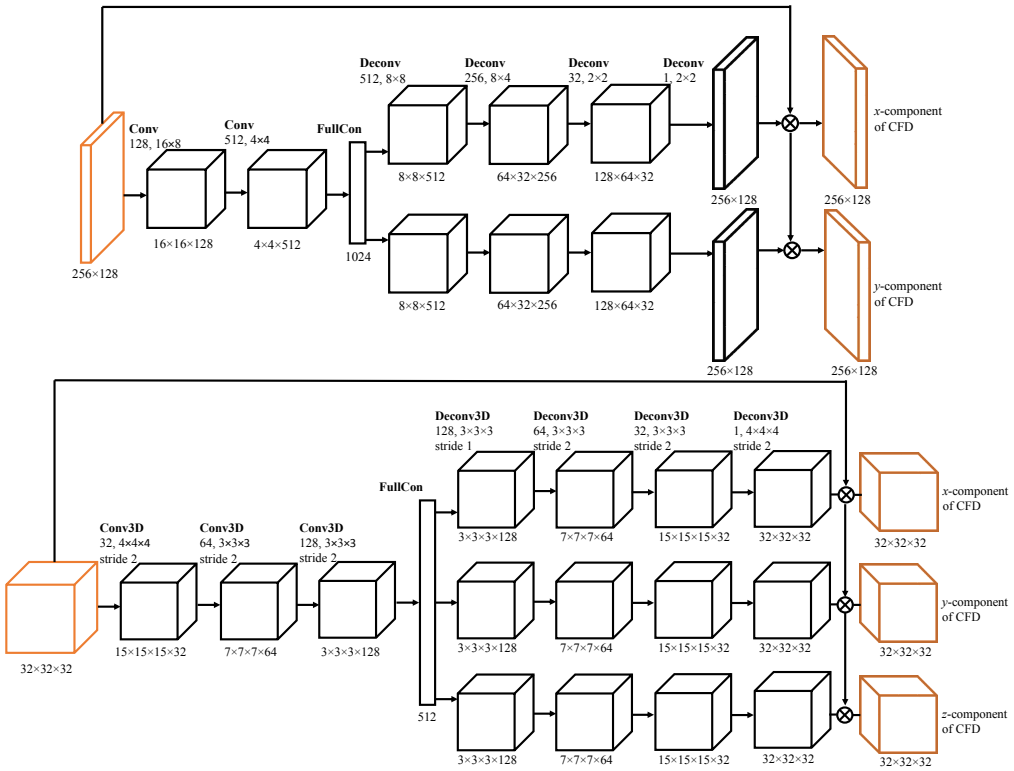

Figure 9: (Top) Network architecture for 2D geometry, (Bottom) Network architecture for 3D geometry. Black cubes/rectangles represent the feature maps. The dimensionalities of feature maps are indicated below. Brown cubes are for SDF and CFD components. Conv, Conv3D, Deconv and Deconv3D represent 2D/3D convolutions and 2D/3D deconvolutions respectively. The number of filters and the kernel size are shown below the operations. The strides for 2D convolutional and deconvolutional layers are the same as kernel sizes. All layers are followed by a rectifier non-linearity except output layers. Arrows indicate the forward operation directions.

the ones reported by the patch-wise linear regression. The prediction accuracy advantage comes from the fact that our CNN models learn a high-level representation of the entire geometry, while the patch-wise linear regression model only takes a local region of SDF as input, thus the input information may not be sufficient to provide accurate prediction for the CFD components. The loss of global geometry information degrades the prediction accuracy much more for 3D case because there are two geometry shapes and the global structure information is even more critical for CFD prediction. Even though a non-patch-wise linear regression model could also take the entire SDF as input, the number of parameters would be too large to handle and thus it is not applicable in practice.

We visualize the CNN's output in Figure 10 and Figure 11 for understanding of the errors from our model. Figure 10 visualizes the CNN prediction on 2D geometry. We visualize the LBM ground truth, the CNN prediction and prediction errors in columns. The shared encoding model's predictions are used in generating the CNN's results. The geometry shapes are visualized in dark blue. The results show that the errors are centered on the geometry boundaries, and the errors are much less than the CFD ground truth. Following the same column order, in our 3D visualization (Figure 11), we show three slices of the velocity fields on the X-Y, X-Z and Y-Z planes.

Data Set	Separated	Shared	PatchLR
2D Type I	1.76%	1.98%	22.86%
Car Type I	11.09%	9.04%	29.03%
2D Type II	3.08%	2.86%	27.66%
Car Type II	15.34%	16.53%	35.96%
3D	-	2.69%	334.19%*

Table 2: CFD prediction results of CNNs and patch-wise linear regression baseline. (*Based on Equation 10, if the error is larger than the true value, then error is larger than 1.)

5.3 Performance Analysis

The main motivation for our surrogate models is that CNN prediction of non-uniform steady laminar flow is considerably faster than traditional LBM solvers. Furthermore CNNs utilize GPU to evaluate the CFD results and the computation overhead per instance could therefore be reduced by allowing multiple predictions to be executed in parallel.

LBM are well suited to massively parallel architectures, as each cell in the lattice can be updated independently at every time step. A widely accepted performance metric for LBM based solvers is Million Lattice Updates per Second (MLUPS). For example, if a 2D LBM solver achieves a per-

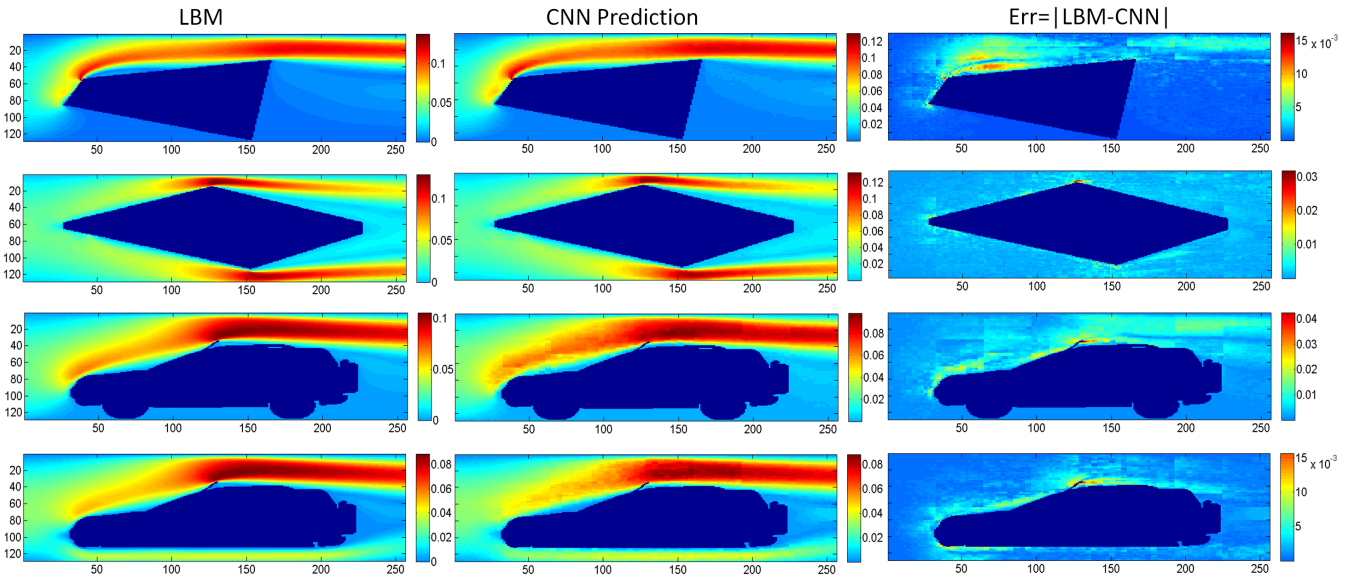

Figure 10: 2D prediction result visualization. The first column shows the magnitude of the LBM ground truth. The second column shows the magnitude of the CNN prediction. The third column shows the magnitude of the difference between the CNN prediction and LBM results.

formance of up to 20 MLUPS, it is the equivalent of performing 1000 time steps per second at a resolution of 200 × 100 lattice points. Modern LBM solvers that are algorithmically optimized for GPU hardware can achieve 820 MLUPS [31, 21]. Using MLUPS as the performance metric, we can estimate the run time of each individual LBM experiment we performed if they had been running at the speed of the state-of-the-art GPU optimized LBM solvers, which is approximately 2 seconds. The average time cost per instance results for LBM solvers are summarized in Table 3.

Methods	LBM CPU	LBM GPU
Time cost	82.64s	2.02s
MLUPS	20.11	820

Table 3: Time of LBM solver on CPU and GPU.

The time results of our CNNs are in Table 4. The time cost measures the average time to generate the CFD given the geometry shape's SDF input. Since CNN based surrogate models could amortize computational overhead per instance by predicting multiple instance in parallel. We measure the average time cost for different batch sizes. Moreover, we compare the time cost of the shared encoding and separated encoding[3]. First, the results show that the average time cost decreases significantly as the batch size becomes larger. Second, the separated encoding takes more time than the shared encoding on different batch sizes. The prediction accuracy of shared and separated encoding architectures is close, but the shared encoding outperforms the separated encoding in terms of time cost.

We use the shared encoding CNNs to compute the speedup,

[3]The time cost of separated encoding measures the total time of sequential prediction of different CFD components.

CNN batch size	1	10	100
shared encoding	0.0145s	0.0077s	0.0069s
separated encoding	0.0182s	0.0085s	0.0072s

Table 4: Time of CNN models on GPU.

compared to LBM on CPU and GPU. The speedup results are summarized in Table 5.

Batch Size	Speedup (CPU)	Speedup (GPU)
1	5699	139
10	10732	262
100	11977	292

Table 5: Speedup results of our CNN surrogate models compared to LBM for different batch sizes.

The speedup results show that (1) GPU accelerated CNN model achieves up to 12K speedup compared to traditional LBM solvers running on a single CPU core, (2) the CNN model achieves up to 292 speedup compared to GPU-accelerated LBM solver, and (3) the speedup increases as batch size increases because the overtime in using GPU is amortized.

6. FUTURE WORK AND CONCLUSION

Even though for many domains, such as architectural design, low Reynolds number flows [2] are usually sufficient, we intend to explore higher Reynolds number flows in the future, to extend the approach to other areas of design optimization.

It would also be worthwhile investigating whether we could use the results from our approximation models as an initial setup to warm start high-accuracy CFD simulations. Since the predictions are fairly close representations of the final,

fully converged results, the number of iterations required to converge to steady state could be greatly reduced, and therefore high-accuracy traditional CFD methods could be made to converge much more quickly.

In this paper, we apply deep Convolutional Neural Networks to Computation Fluid Dynamics modeling. Our main motivation is to provide lightweight (fast and less-accurate) interior and exterior flow performance feedback to improve interactivity of early design stages, design exploration and optimization. Using this approach, designers can easily generate immense amounts of design alternatives without facing the time-consuming task of evaluation and selection. Our results show that our CNN based CFD predictions achieve 2-4 orders of magnitude speedup compared to traditional CFD methods to model steady state laminar flow at a cost of low error rates.

7. REFERENCES

[1] M. Ahmed and N. Qin. *Surrogate-Based Aerodynamic Design Optimization: Use of Surrogates in Aerodynamic Design Optimization*. Aerospace Sciences and Aviation Technology, ASAT-13, 2009.

[2] J. Anderson. *Computational Fluid Dynamics*. 1995.

[3] V. Balabanov, A. Giunta, O. Golovidov, B. Grossman, H. Mason, T. Watson, and T. Haftkalf. Reasonable design space approach to response surface approximation. *Journal of Aircraft*, 1999.

[4] M. Batill, A. Stelmack, and S. Sellar. Framework for multidisciplinary design based on response-surface approximations. *Journal of Aircraft*, 1999.

[5] Y. Bengio. *Learning deep architectures for AI, Foundations and trends in Machine Learning*. 2009.

[6] L. Chittka, P. Skorupski, and E. Raine. Speed-accuracy tradeoffs in animal decision making. *Trends in Ecology and Evolution*, 2009.

[7] D. Daberkow and D. N. *New Approaches to Conceptual and Preliminary Aircraft Design: A Comparative Assessment of a Neural Network Formulation and a Response Surface Methodology*. World Aviation Conference, 1998.

[8] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirba, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[9] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, 2014.

[10] H. Fang, M. Rais-Rohani, Z. Liu, and M. Horstemeyer. A comparative study of metamodeling methods for multiobjective crashworthiness optimization. *Computers & Structures*, 2005.

[11] G. Forsythe and W. Wasow. *Finite-Difference Methods for Partial Differential Equations*. 1960.

[12] A. Giunta. *Aircraft Multidisciplinary Design Optimization Using Design of Experiments Theory and Response Surface Modeling Methods*. 1997.

[13] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. In *Computer Vision–ECCV*. 2014.

[14] D. Hartmann, M. Meinke, and W. Schröder. Differential equation based constrained reinitialization for level set methods. *Journal of Computational Physics*, 2008.

[15] V. Heuveline and J. Latt. *The OpenLB project: an open source and object oriented implementation of lattice Boltzmann methods*. International Journal of Modern Physics, 2007.

[16] S. Jeong, M. Murayama, and K. Yamamoto. Efficient optimization design method using kriging model. *Journal of Aircraft*, 2005.

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*. 2014.

[18] A. Keane and P. Nair. *Computational approaches for aerospace design*. 2005.

[19] S. J. Leary, A. Bhaskar, and A. J. Keane. Global approximation and optimization using adjoint computational fluid dynamics codes. *AIAA journal*, 2004.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[21] M. Mawson, G. Leaver, and A. Revell. Real-time flow computations using an image based depth sensor and GPU acceleration. 2013.

[22] G. R. McNamara and G. Zanetti. Use of the boltzmann equation to simulate lattice-gas automata. *Physical Review Letters*, 1988.

[23] A. Mohamad. *Lattice Boltzmann Method*. 2011.

[24] S. Patankar. *Numerical heat transfer and fluid flow*. CRC Press, 1980.

[25] G. Russo and P. Smereka. A remark on computing distance functions. *Journal of Computational Physics*, 2000.

[26] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 2015.

[27] R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng. Convolutional-recursive deep learning for 3d object classification. In *Advances in Neural Information Processing Systems*, 2012.

[28] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Deep end2end voxel2voxel prediction. *arXiv preprint arXiv:1511.06681*, 2015.

[29] J. Turner, W. Clough, C. Martin, and J. Topp. Stiffness and deflection analysis of complex structures. *Journal of the Aeronautical Sciences*, 1956.

[30] N. Umetani, Y. Koyama, R. Schmidt, and T. Igarashi. Pteromys: interactive design and optimization of free-formed free-flight model airplanes. *ACM Trans. Graph.*, 2014.

[31] K. Valderhaug. *The Lattice Boltzmann Simulation on Multi-GPU Systems*. 2011.

[32] S. Wilkinson, G. Bradbury, and H. S. *Reduced-Order Urban Wind Interference, Simulation: Transactions of the Society for Modeling and Simulation International*. 2015.

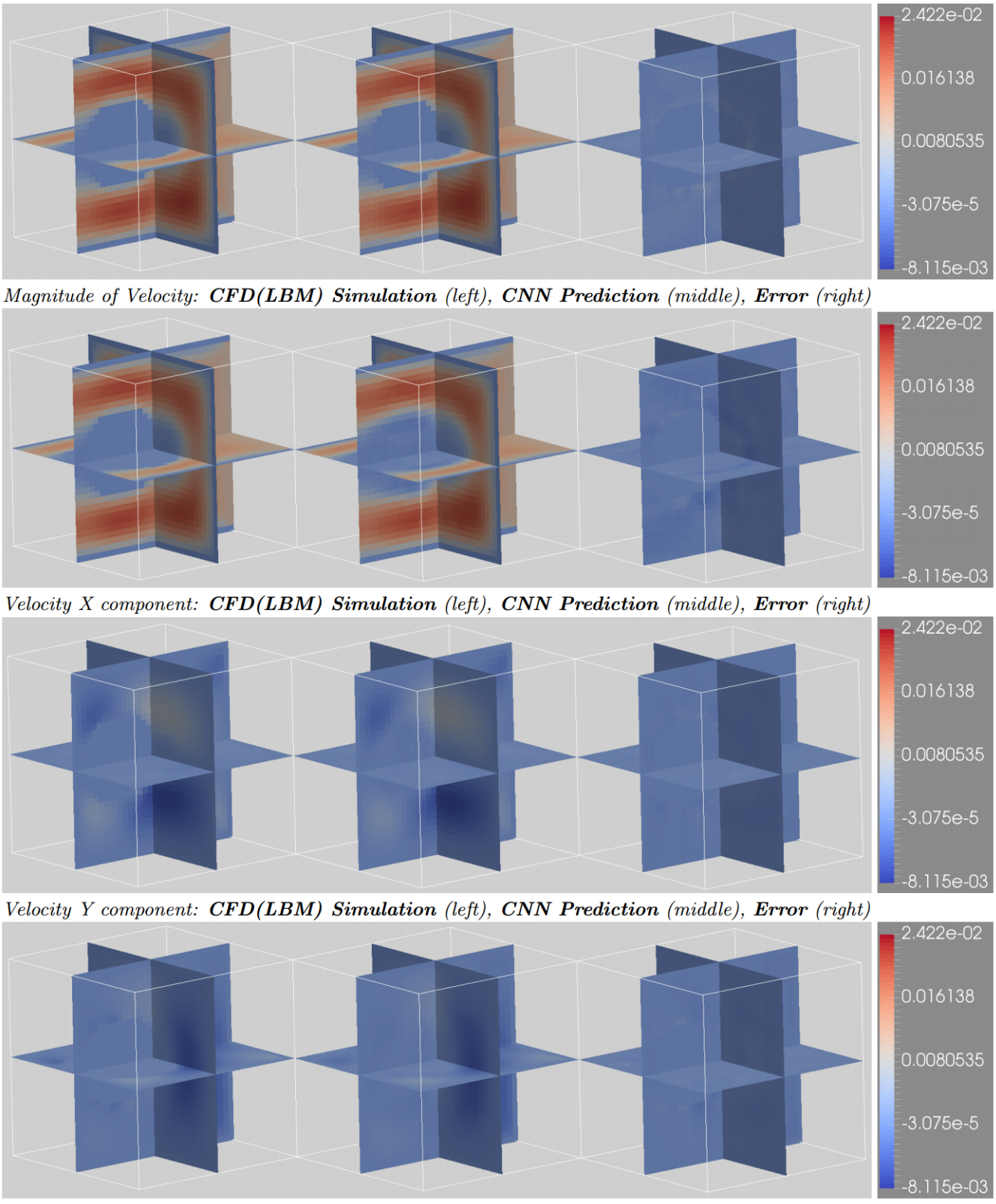

Figure 11: 3D prediction visualization. The first column shows the magnitude of the LBM ground truth. The second column shows the magnitude of the CNN prediction. The third column shows the magnitude of the difference between the CNN prediction and LBM results.