Proceedings of the ASME 2020
International Design Engineering Technical Conferences
and Computers and Information in Engineering Conference
IDETC/CIE 2020
August 16-19, 2020, St. Louis, MO, USA

IDETC2020-22714

LEARNING TO ABSTRACT AND COMPOSE MECHANICAL DEVICE FUNCTION AND BEHAVIOR

Jun Wang[*]
IDEAL Lab
Dept. of Mechanical Engineering
University of Maryland
College Park, MD 20742
Email: jwang38@umd.edu

Kevin Chiu
IDEAL Lab
Dept. of Mechanical Engineering
University of Maryland
College Park, MD 20742
Email: knchiu@terpmail.umd.edu

Mark Fuge
IDEAL Lab
Dept. of Mechanical Engineering
University of Maryland
College Park, MD 20742
Email: fuge@umd.edu

ABSTRACT

While current neural networks (NNs) are becoming good at deriving single types of abstractions for a small set of phenomena, for example, using a single NN to predict a flow velocity field, NNs are not good at composing large systems as compositions of small phenomena and reasoning about their interactions. We want to study how NNs build both the abstraction and composition of phenomena when a single NN model cannot suffice. Rather than a single NN that learns one physical or social phenomenon, we want a group of NNs that learn to abstract, compose, reason, and correct the behaviors of different parts in a system. In this paper, we investigate the joint use of Physics-Informed (Navier-Stokes equations) Deep Neural Networks (i.e., Deconvolutional Neural Networks) as well as Geometric Deep Learning (i.e., Graph Neural Networks) to learn and compose fluid component behavior. Our models successfully predict the fluid flows and their composition behaviors (i.e., velocity fields) with an accuracy of about 99%.

1 INTRODUCTION

Mathematical abstractions are a fundamental tool used by scientists and engineers to study and predict the world [1]. Current neural networks (NNs) are gaining great and increasing success in deriving single types of abstractions for a small set of phenomena, for example, using a single NN to predict a flow velocity field or structural stress field [2, 3]. As one data-driven approach, NNs are becoming a prominent alternative to the conventional analysis (e.g., finite-element analysis) by significantly cutting the computation cost with minimal loss to accuracy, especially on problems similar to their training data. However, NNs are not effective and flexible when predicting a large (but different) system as a composition of small phenomena and reasoning about their interactions. For example, to model a fluid pipe flow system composed of different shapes of pipe components, one might first abstract the fluid behavior of each pipe component, then make the composition of these pipe components by reasoning about their interactions and correcting the composed fluid behavior. Instead of training a single NN that models one physical or social phenomenon, we want to study how multiple NNs learn to abstract, compose, reason, and correct the behaviors of different parts in a system.

In the present work, we demonstrate the importance of independently modeling both abstraction and composition when learning and predicting system behavior by exploring and discovering the fluid component behavior. Particularly, we explore building deep neural networks (DNNs) that automatically abstract (predict) and compose two-dimensional (2D) velocity fields of pipe flows (Figure 1). Our proposed multi-NN

[*]Corresponding Author

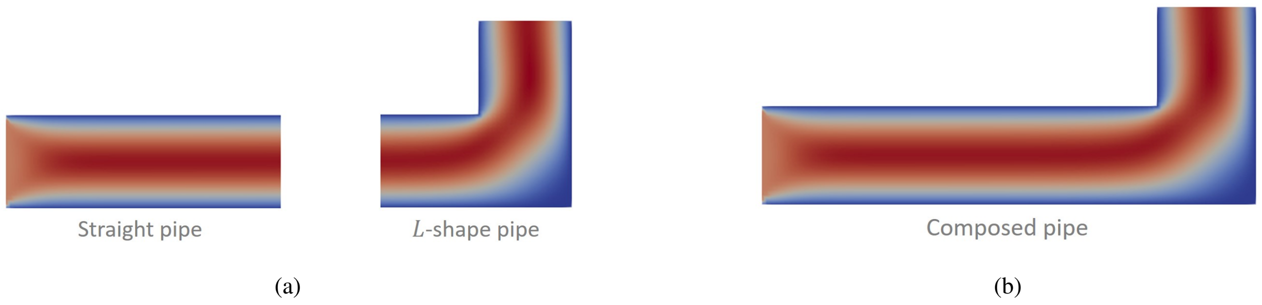


FIGURE 1: Two-dimensional pipe flows. In the simulated color maps, blue outlines pipe shapes and red represents fluid flows (deeper red indicates higher speed). (a) Two types of basis pipes are used in this work. One is a straight pipe and the other one is an *L*-shape pipe. (b) The long pipe flow composed by the straight pipe and *L*-shape pipe.

model consists of two DNNs, a Deconvolutional Neural Network (DeCNN) [4, 5] and a Graph Neural Network (GNN) [6, 7]. The DeCNN predicts the 2D velocity fields of short basis pipe flows (*learning-to-abstract*), while the GNN connects two consecutive basis pipe flows to make a smooth long pipe flow by correcting errors in the velocity fields (especially at the interface) of the two connected short pipe flows (*learn-to-compose*). We compare the prediction and composition accuracy for these two DNN models with ground-truth results obtained by a partial differential equation (PDE) solver (*FEniCS*[1]) for multi-physics models.

The two contributions of this paper are:

- We propose an approach to composing and reconstructing the mechanical behavior of multi-component systems. We do this by separating out the steps of composition and abstraction via the use of geometric deep learning (e.g., via GNNs) and physics-informed neural networks (e.g., DeCNNs), respectively. Figure 12 and Table 2 demonstrate that directly modeling composition using the GNN model reduces prediction error by an order of magnitude compared to just abstracting each part using the DeCNNs (Fig. 10). We demonstrate this on small laminar flow pipe networks.
- We study the effects of increasing the composition length (via adding longer chains of parts) and part complexity (via adding a new pipe element type) on the predictive accuracy of the above model. Increasing the system composition size beyond the size of the training set degrades predictive accuracy while adding an additional part type did not significantly reduce performance, at least for the two pipe segment types we used in this limited class of pipe network problems.

In the next section, we outline the related work on NN models for abstraction and composition. An overview of the whole framework is given in Section 3. In Section 4, the specific methods are described in detail. Numerical results are presented in Section 5. Conclusion and discussions are provided in Section 6.

[1]https://fenicsproject.org/

2 LITERATURE REVIEW

The techniques pertinent to the (1) FEA-NN-based prediction models, and (2) GNN models for physical system modeling and image analysis are briefly reviewed in this section.

2.1 FEA-NN-Based Prediction Models

Prediction models based on FEA results and NN have been applied in computational mechanics as a faster-to-compute alternative to time-consuming FE simulations.

2.1.1 Multi-layer perceptron (MLP): First-generation FEA-NN-based models built upon artificial neural networks (ANN) with simple MLP architectures. For example, Shahani et al. [8] used an MLP to predict the behaviors (e.g., temperature field, strain field, and forces) of the slab in the hot rolling process based on the back-propagation (BP) method. Kazan et al. [9] created a prediction model of springback in a wipe-bending process of sheet metal using a simple three-layer ANN. Umbrello et al. [10] proposed a hybrid FE-ANN approach for predicting residual stresses and the optimal cutting conditions during hard turning of a particular type of bearing steel. Recently, MLPs have also been used to accelerate prediction of the separation stresses in the bottom-up stereolithography (SLA) process of additive manufacturing to assist in an *in-situ* feedback system for alleviating the print failure [11].

However, due to its limited hidden layer depth and lack of built in invariance properties, MLPs are not as effective in learning more general and complex PDEs or phenonmena. This limitation arises when an input feature vector insufficiently conveys essential information (e.g., spatial position relationship).

2.1.2 Convolutional nueral networks (CNN): CNNs [12], aimed at learning data that comes in the form of arrays (e.g., images), provide the ability to capture the spatial and temporal dependency relationship within an array (e.g.,

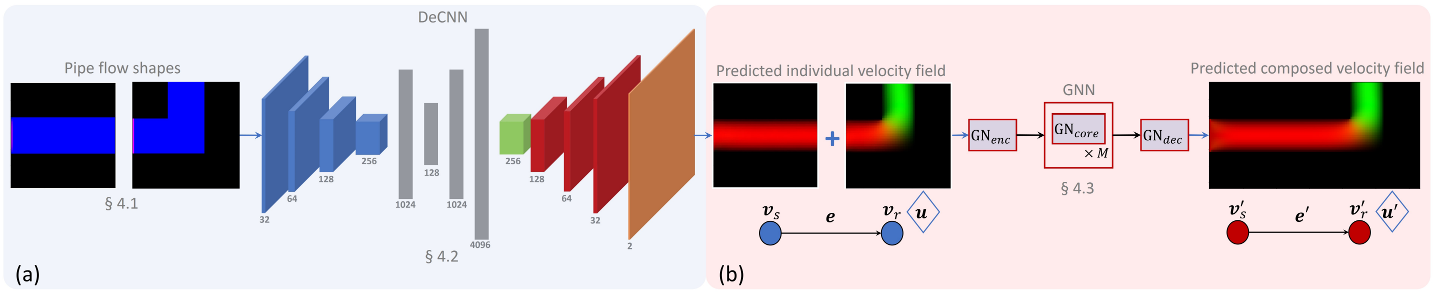


FIGURE 2: Overview of our DeCNN-GNN computational pipeline for abstraction and composition of fluid pipe flows.

translation invariance). For example, Khadilkar et al. [13] improved the prediction model of the bottom-up SLA process by enabling the multi-slice stress field prediction using CNN. Liang et al. [14] took advantage of convolutional and transposed convolutional layers to develop an encoder-decoder model for fast prediction of the aortic wall stress distribution. Sosnovik et al. [15] introduced a convolutional encoder-decoder architecture to speed up the topology optimization methods. Nie et al. [3] realized the prediction of the stress fields in 2D linear elastic cantilevered structures using CNN. A recent study on shaping microfluid flow (flow sculpting) using CNN has been proposed by Stoecklein et al. [16] to solve the inverse problem of designing a flow sculpting device for a desired fluid flow shape.

In this paper, we build upon those past efforts by constructing an encoder-decoder to predict the pipe flow velocity fields. This architecture choice—making full use of convolutional and deconvolutional (transposed convolutional) layers, as opposed to just a feedforward network on the boundary conditions—enables us to explore composition of multiple components in a way that is not straightforward to do with feedforward networks.

2.2 GNN Models for Physical System Modeling and Image Analysis

GNNs [17] are connectionist models that capture the dependence of graphs via message passing between the nodes of graphs. Zhou et al. [7] defined GNN applications in physical system modeling as a structural scenario where the data has explicit relational structure and the applications in image analysis as a non-structural scenario where the relational structure is not explicit. In this paper, we investigate the physical system composition of fluid pipe flows relying on non-structural image data, thus reviewing the relevant work below.

2.2.1 Physical system modeling: By representing objects as nodes and relations as edges, a GNN can reason about objects, relations, and physics in a simplified but effective way. Inspired by the GNN model [17], Battaglia et al. [18] introduced an interaction network (IN), which they propose as a general-purpose, learnable physics engine, and a framework for reasoning about object and relations in a wide variety of complex real-world domains. Sanchez-Gonzalez et al. [19] then extended and improved the GNN model such that it can support accurate prediction, inference, and control across eight distinct physical systems. Rather than INs that tackle fully observable systems, Li et al. [20] further developed propagation networks (PropNet), a differentiable, learnable dynamics model that handles partially observable situations and enables instantaneous propagation of signals beyond pairwise interactions. The DeepMind AI group has made available a *Graph Nets*[2] library that enables building GNNs in *TensorFlow*[3] and *Sonnet*[4] based on the work of Battaglia et al. [6]. Their library provides the flexibility to design GNNs for different applications, including the physical system prediction (e.g., mass-spring systems). Instead of using static and dynamic parameters of a physical system, Watters et al. [21] proposed a visual interaction network (VIN) that predicts future physical states from input image frames (pixels from video data).

In our study, we adopt the Graph Nets library to compose the fluid pipe flows to construct a GNN model depending on non-structural images of the pipe flow velocity field. To the best of our knowledge, GNN models have never been used to reason and compose physical systems using whole images.

2.2.2 Image analysis: Given that images belong to non-structural scenarios where the relational structure is not explicit, GNNs can be leveraged to incorporate, infer, or assume relational structures for improving the performance in images tasks. GNNs have been used in *image classification* [22, 23, 24, 25, 26] by collaborating the implicit knowledge representations (e.g., word embedding) with explicit relations (e.g., knowledge graph or similarity kernel), where nodes are the ex-

[2]https://github.com/deepmind/graph_nets
[3]https://www.tensorflow.org/
[4]https://sonnet.readthedocs.io/en/latest/

tracted representation features of images, and the edges represent relations between nodes. GNNs are a natural message passing tool between objects or humans in an image, and thus provide a reasonable basis for *visual reasoning* problems, like social relationship understanding [27], visual question answering [28, 29], and interaction detection [30].

To this end, a graph refers to an organization of the correlations between spatial (social) relationships and semantic objects (humans), with nodes having the extracted feature vectors (embedding) of objects (words), and edge features encoding their spatial relationships (syntactic dependencies). Another important application for GNNs are image *semantic segmentation*. In that context, GNNs resolve some shortcomings of traditional CNNs, which only capture limited local context. GNNs, in contrast, can explicitly model semantic part layouts and their interactions with respect to the whole image. GNNs do this by representing an image as pixels, patches, or points, and taking each pixel or point (e.g., superpixel or superpoint) as a semantically consistent node and their spatial (contextual) relations as edge features [31, 32, 33].

In these existing image applications, to construct the graphs, images were processed or encoded into representative feature vectors for node features. No work has ever taken the whole original images as node features. In the meantime, image data has never been used to construct a GNN model for any physical system modeling problem. In this work, the velocity fields of the pipe flows are represented as images, and we want to directly use these images to train a GNN model without losing important information.

3 OVERVIEW

Figure 2 outlines the computational pipeline for learning the abstraction and composition using a DeCNN model (*learn-to-abstract*) and a GNN model (*learn-to-compose*). The DeCNN is designed to create an image-to-image mapping as an encoder-decoder model (including deconvolutional layers[5]), which takes the images of pipe shapes (together with boundary conditions) as input and the images of 2D velocity fields as output (Figure 2(a)). The DeCNN model approximates the Navier-Stokes equations to predict the velocity fields based on the pipe shapes and boundary conditions (inlet velocity and outlet pressure)—this is what we refer to as "learning to abstract".

The predicted velocity fields of two consecutive pipe flows are then passed to a GNN model consisting of two nodes that are connected by a directed edge (Figure 2(b)). The velocity field images of the two consecutive pipe flows are flattened into feature vectors for the two nodes, and the edge feature is a binary number indicating how the two pipes are connected. The input of the GNN model is obtained from the DeCNN model's prediction, and the ground truth is extracted from the simulated velocity field of the connected long pipe flow. The GNN model provides the ability to correct the composition errors (especially at the interface) through its message passing mechanism. In the next section, we describe each component in detail.

4 METHODS

4.1 Data Generation

We used the *FEniCS* library [34] (a finite-element solver) to compute the velocity fields of the pipe flows. We simulate both separated short pipe flows and combined long pipe flows of various lengths. The shorter segments of pipe flow data are used to train the DeCNN model (learn-to-abstract), while the longer composed segments are used to train the GNN model (learn-to-compose).

Specifically, we compute the 2D fluid flow using a stationary Navier-Stokes simulation in a laminar flow, with the following equations:

$$\begin{aligned} -\nu\Delta u + (u\cdot\nabla)u + \nabla p &= f \\ div(\mathbf{u}) &= 0 \end{aligned} \tag{1}$$

where u is the velocity field, p is the pressure field, f is the external body force on the domain, and ν is the kinematic viscosity.

To create various pipe flows, we simplify the task by selecting two basis (short) pipes (straight and L-shape) that compose long pipes as shown in Figure 1(a). These two basis pipes can be rotated or flipped as needed to fit the pipe composition. There are five unique combinations of this two-pipe composition (Figure 3). We omit the "Z" formation pipe (Figure 3(e)) in this paper due to length restrictions. It is worth noting that in Figure 3, we generate the "all-square" (formed by two identical right triangle elements) mesh element (with side length of 0.1) on the pipes such that the simulated velocity fields can be extracted in a structured data format, which can then be easily zero-padded as images.

For every configuration in Figure 3, we simulate the combined two-pipe composition and each basis pipe separately. The simulations have three different boundary conditions. At the outlet for every pipe, we set the pressure to 0 to model an open outlet. On the side walls (i.e., not the inlet or outlet), we set velocity to 0 in every direction to model the no-slip condition. At the inlet, we can apply four possible polynomial velocity profiles (Figure 4) with each scaled by a random number R ($0 \leq R \leq 1$). These polynomials of degrees 0, 2, 4, or 6 are given in Equation 2:

$$v_x = R[1-(6(y-y_c))^d] \tag{2}$$

[5]A deconvolutional layer is more strictly called a transposed convolutional layer.

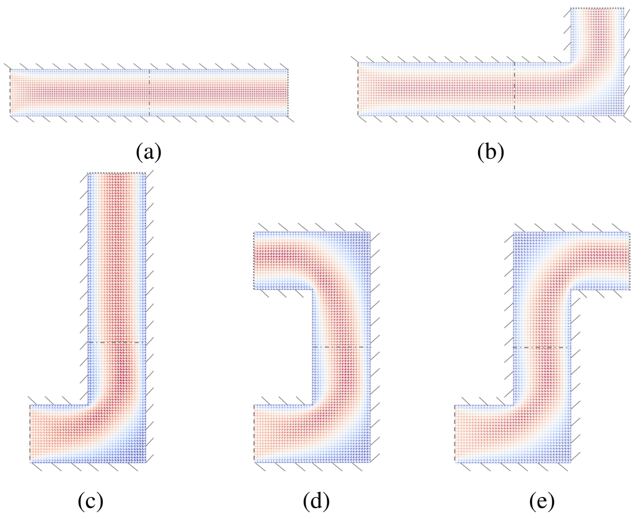


FIGURE 3: Five unique combinations of two-pipe composition using the straight and *L*-shape pipes.

where v_x is the *x*-direction velocity at a mesh grid point of the inlet, y is the y-coordinate of the point, and d is the degree of the polynomial. When $d = 0$, we set $v_x = R$ to make a constant inlet velocity. The constant y_c is the y-coordinate of the center point of the pipe inlet. In this work, y_c equals to 0.315 (the padded square image has a side length of 0.63).

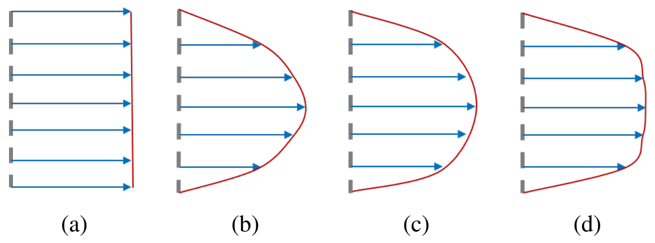
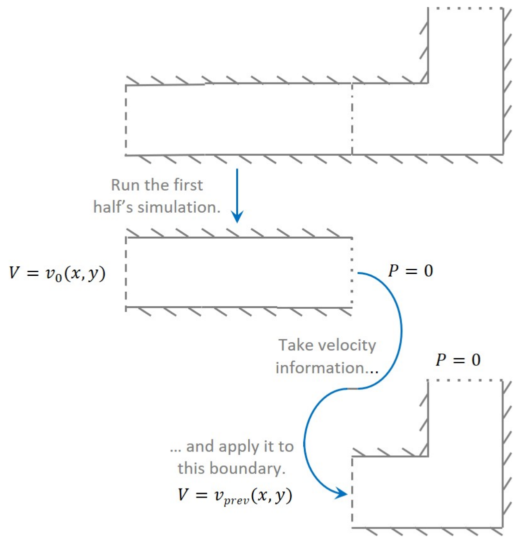


FIGURE 4: Four polynomial-shape velocity profiles. (a) $d = 0$. (b) $d = 2$. (c) $d = 4$. (d) $d = 6$.

These inlet velocity profiles (along with the zero-pressure and no-slip conditions on the outlet and walls) are first applied on the two-pipe composition to simulate the ground truth data for the GNN model. To perform the corresponding short pipe simulations, we split the two-pipe composition into its two basis pipes and apply the same inlet velocity profile on the first segment of the composition. We again apply the zero-pressure and no-slip conditions but on the interface (”intermediate outlet”) and walls, as shown in Figure 5. Running this simulation also provides the velocity profile for the next pipe. We take the velocity profile at the “intermediate outlet” of the first pipe and apply it to the “intermediate inlet” of the second pipe to run the second pipe simulation. These two short pipe simulations generate the ground truth data for the DeCNN model.


Run the first half's simulation.
$V = v_0(x, y)$
$P = 0$
Take velocity information...
$P = 0$
... and apply it to this boundary.
$V = v_{prev}(x, y)$
FIGURE 5: Schematic illustration of the database creation for DeCNN and GNN.

We repeat this entire process on each of the combinations (Figure 3(a)∼(d)) using all the four inlet velocity profiles with each one scaled by 2,000 random R values. Therefore, for each combination, we perform 8,000 simulations and totally collect 32,000 groups of data for GNN model ($32,000 \times 80\% = 25,600$ for training and $32,000 \times 20\% = 6,400$ for testing). Among the four combinations, Figure 3(a) and (b) both have the straight pipe as the first pipe and are thus assigned the same inlet velocities. The same strategy is applied to Figure 3(c) and (d). In this way, the database creation process is simplified such that the short split pipe has 8,000 simulation data for each of the first straight pipe and *L*-shape pipe, and 16,000 simulation data for the next straight pipe and *L*-shape pipe. As a result, we produce 48,000 data in total for DeCNN model ($48,000 \times 80\% = 38,400$ for training and $48,000 \times 20\% = 9,600$ for testing).

4.2 DeCNN

We use a DeCNN model to learn-to-abstract the flow field behavior (Figure 2(a)) as a projection into a lower-dimensional latent space. As illustrated in Figure 6, a two-channel is proposed to perform the prediction of the 2D velocity field of a pipe flow. For visualization purposes, the 2D velocity vectors (x, y) occupy the R- and G-channels, respectively, with B-channel padded with zeros. The input of the model is a three-channel RGB image that is stacked by the inlet velocity vector and the pipe shape. Among the three channels, the 2D inlet velocity vectors (x, y) are filled in the R- and G-channels, while the B-channel takes the binary image of the pipe shape. Figure 6 illustrates an example input

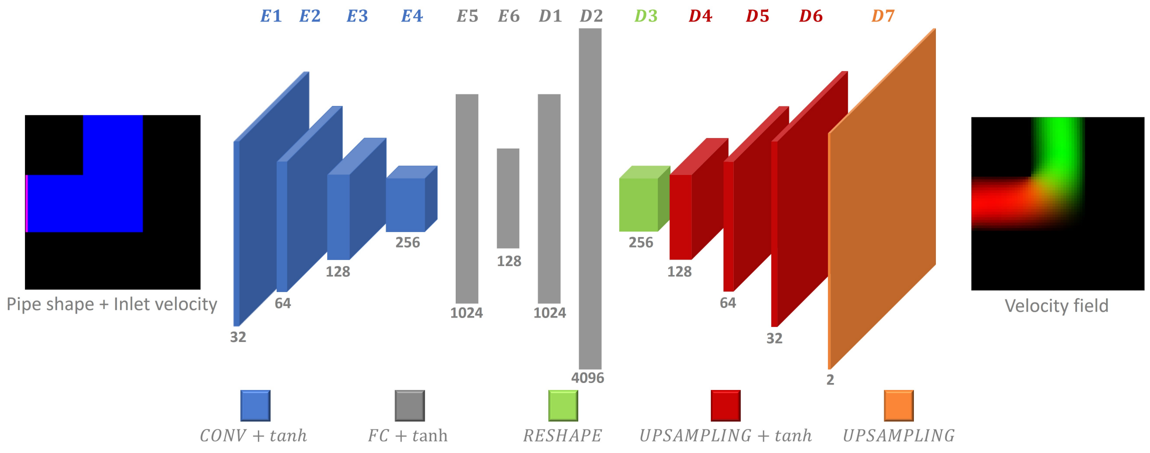


FIGURE 6: The architecture of DeCNN for learn-to-abstract. The stride is 2×2. All kernels are of size 4×4. The number of kernels is represented by the number at the bottom of the layer. Excluding the reshape output layer, every layer (11 in total) is followed by a batch normalization layer. The activation function used in DeCNN is *tanh*.

RGB image. The blue region describes the shape of pipes, while the red line indicates the inlet velocity vector—i.e., the vector is assigned to the very first column of pipe pixels at the inlet.

A DeCNN uses multiple convolutional and deconvolutional (transposed convolutional) layers in an encoder-decoder structure (convolutional autoencoder). This convolutional autoencoder learns to encode the input in a set of simple signals and reconstruct the input from them [35]. The encoder in our DeCNN model consists of four convolutional layers ($E1 \sim E4$). Each convolutional layer has a filter with a kernel size of 4×4 and a stride of 2×2. The padding scheme is a zero-padding so that the output in convolution does not differ in size as input. After two fully connected (FC) layers $E5$ and $E6$, the latent feature representation of the input image is obtained. The latent feature representation is then fed into the decoder, which is the reverse of the encoder. After two FC layers ($D1$ and $D2$) and a reshape layer ($D3$), four deconvolutional layers ($D4 \sim D7$) are used for upsampling the image to the same resolution as the input image. Excluding the reshape and the output layers, every layer (11 in total) is followed by a batch normalization layer. The activation function used in the DeCNN model is *tanh* (hyperbolic tangent).

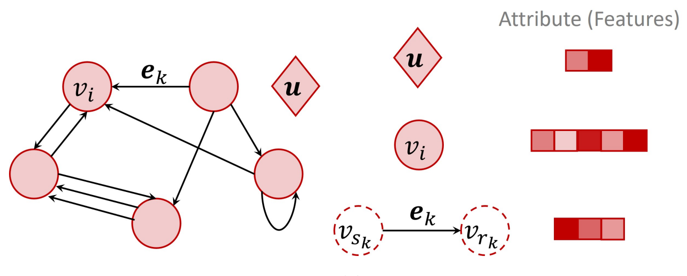
The height, width, and depth of the input image vary through the DeCNN model: $64 \times 64 \times 3$ (input image) $\rightarrow 32 \times 32 \times 32$ ($E1$) $\rightarrow 16 \times 16 \times 64$ ($E2$) $\rightarrow 8 \times 8 \times 128$ ($E3$) $\rightarrow 4 \times 4 \times 256$ ($E4$) $\rightarrow 1024 \times 1 \times 1$ ($E5$) $\rightarrow 128 \times 1 \times 1$ ($E6$) $\rightarrow 1024 \times 1 \times 1$ ($D1$) $\rightarrow 4096 \times 1 \times 1$ ($D2$) $\rightarrow 4 \times 4 \times 256$ ($D3$) $\rightarrow 8 \times 8 \times 128$ ($D4$) $\rightarrow 16 \times 16 \times 64$ ($D5$) $\rightarrow 32 \times 32 \times 32$ ($D6$) $\rightarrow 64 \times 64 \times 2$ ($D7$: output layer).

4.3 GNN

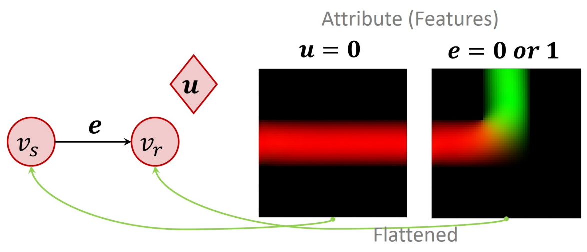
The GNN model's purpose is to learn-to-compose the abstraction produced by the DeCNN. Specifically, we use the graph networks (GN) framework [6], which unifies and extends various GNN [36], message-passing neural network (MPNN) [37], and non-local neural network (NLNN) [38] approaches. It also supports constructing complex GNN architectures from simple building blocks (GN blocks). This approach affords the following advantages: (1) flexible representations of the graph attributes and the graph structure, (2) configurable within-block structure and functions, and (3) composable multi-block architectures by simply linking GN blocks.

4.3.1 A Graph and GN Block In the GN framework, a *graph*, defined as a 3-tuple $G = (V, E, \mathrm{u})$, is referred to as a directed, attributed multi-graph with a global attribute (Figure 7(a)). The $V = \{v_i\}_{i=1...N_n}$ is the set of nodes, where each v_i is a node's attribute (a feature vector). The $E = \{e_k, s_k, r_k\}_{k=1...N_e}$ is the set of edges, where each e_k is an edge's attribute (a feature vector), and r_k and s_k are the indices of the sender and receiver nodes, respectively. The u is the global attribute (e.g., u can be the gravity constant). In this paper, to compose two short pipe flows into one longer pipe flow, we construct a two-node graph with the flattened velocity field images fed into the (receiver and sender) nodes as feature vectors (Figure 7(b)). To distinguish between horizontal and vertical connections of two pipes, we assign zero or one as the edge feature. There is no global attribute for the pipe flow composition. The global feature can be ignored or fixed as zero. This approach can model longer pipe segments or more complex pipe networks by expanding the size of the graph.

The main computation unit in the GN framework is a *GN block*, a graph-to-graph module that takes a graph as input, per-



(a)



(b)

FIGURE 7: Schematic illustration of a graph. (a) The standard graph defined within GN framework with global, node, and edge feature vectors. (b) A two-node graph built for the pipe flow composition of two pipes. The velocity field images ($64 \times 64 \times 2$) of two consecutive pipe flows are flattened into feature vectors for the (sender and receiver) nodes that are connected by an edge. The edge feature is assigned zero indicating horizontal connection and one for vertical connection. The global feature can be ignored or fixed as zero.

forms computations over the structure, and returns a graph with the same structure but updated edge, node, and global features as output. A GN block contains three *update* functions, ϕ, and three *aggregation* functions, ρ:

$$\begin{aligned} e'_k &= \phi^e(e_k, v_{s_k}, v_{r_k}, \mathrm{u}) & \bar{e}'_i &= \rho^{e\to v}(E'_i) \\ v'_i &= \phi^v(\bar{e}'_i, v_i, \mathrm{u}) & \bar{e}' &= \rho^{e\to u}(E') \\ \mathrm{u}' &= \phi^u(\bar{e}', \bar{v}', \mathrm{u}) & \bar{v}' &= \rho^{v\to u}(V') \end{aligned} \tag{3}$$

where $E'_i = \{e'_k, s_k, r_k\}_{r_k=i,\ k=1\dots N_e}$, $V' = \{v'_i\}_{i=1\dots N_n}$, and $E' = \bigcup_i E'_i = \{e'_k, s_k, r_k\}_{k=1\dots N_e}$. The ϕ^e is mapped across all edges to compute per-edge updates, the ϕ^v is mapped across all nodes to compute per-node updates, and the ϕ^u is applied once as the global update. For vector features, a MLP is often used for ϕ. Each ρ function takes a set as input and reduces it to a single element that represents the aggregated information. When a graph, G, is fed as input into a GN block, the computations proceed from the edge, to the node, and to the global level. Algorithm 1 describes the computation steps in a full GN block (Figure 8).

4.3.2 GN Architecture for Pipe Flow Composition

To compute the two-node graph for the pipe flow composition, we compose a three-block architecture following an *encode-process-decode* configuration [39], where a GN$_{enc}$ encodes an input graph into a latent representation, which is then processed M times by a shared core block GN$_{core}$. The output of the core is decoded by a GN$_{dec}$ into an output graph, whose node features would be the pipe flow composition results. Among the three GN blocks, the GN$_{enc}$ and GN$_{dec}$ have an *independent internal structure*, which independently encodes or decodes the edge, node, and global features, while the GN$_{core}$ is a *full GN block*, which performs M rounds of message-passing. Given all vector features in the two-node graph, we adopt an MLP model (with two layers and a latent layer size of 16) for all the edge (ϕ^e), node (ϕ^v), and global (ϕ^u) update functions in the three GN blocks. Figure 8 depicts the detail of the GN architecture used for the pipe flow composition.

Algorithm 1 Computation steps in a full GN block

```
Input: Graph, $G = (V, E, \mathrm{u})$
 1: for each edge $\{e_k, s_k, r_k\}$ do
 2:     Gather sender and receiver nodes, $v_{s_k}$, $v_{r_k}$
 3:     Compute updated edge features, $e'_k = \phi^e(e_k, v_{s_k}, v_{r_k}, \mathrm{u})$
 4: end for
 5: for each node $\{v_i\}$ do
 6:     let $E'_i = \{e'_k, s_k, r_k\}_{r_k=i,\ k=1\dots N_e}$
 7:     Aggregate edge features $e'_k$ per node, $\bar{e}'_i = \rho^{e\to v}(E'_i)$
 8:     Compute updated node features, $v'_i = \phi^v(\bar{e}'_i, v_i, \mathrm{u})$
 9: end for
10: let $V' = \{v'\}_{i=1\dots N_n}$
11: let $E' = \bigcup_i E'_i = \{e'_k, s_k, r_k\}_{k=1\dots N_e}$
12: Aggregate edge features, $\bar{e}' = \rho^{e\to u}(E')$
13: Aggregate node features, $\bar{v}' = \rho^{v\to u}(V')$
14: Compute global features, $\mathrm{u}' = \phi^u(\bar{e}', \bar{v}', \mathrm{u})$
Output: Graph, $G' = (V', E', \mathrm{u}')$
```

5 NUMERICAL ILLUSTRATION

Our code is written in TensorFlow and executed on an NVIDIA Tesla V100 GPU. The hyper-parameters used for training the DeCNN and GNN models are listed in Table 1. The training and testing results show that both DeCNN and GNN models are stable and converged reliably.

5.1 Loss Function and Metrics

Mean squared error (MSE) is selected as the loss function for the DeCNN and GNN training:

$$MSE = \frac{1}{MN} \sum_{j=1}^{M} \sum_{k=1}^{N} (\hat{\eta}_{j,k} - \eta_{j,k})^2 \tag{4}$$

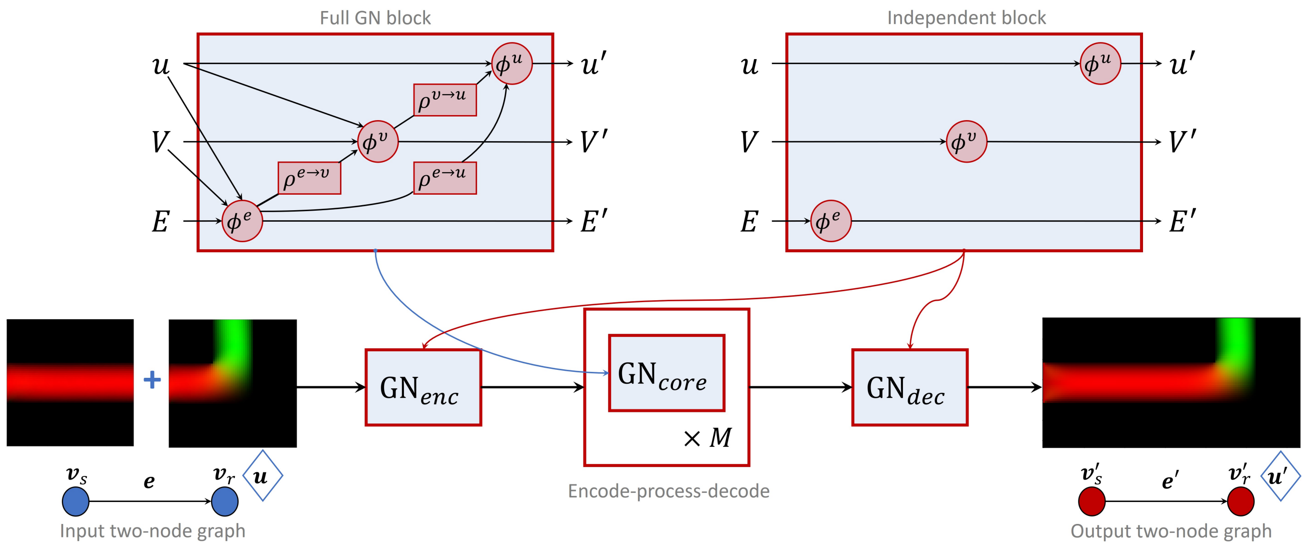


FIGURE 8: GN architecture used for the pipe flow composition.

TABLE 1: Hyper-parameters used for training DeCNN and GNN

Hyper-parameter	**DeCNN**	**GNN**
Optimizer	Adam	Adam
Batch size	32	32
Learning rate	0.001	0.001
Batch normalization (momentum)	0.9	-
Message-passing steps (M)	-	1
Training steps	200,000	1,000,000

where N is the number of pixels and M is the batch size. In each step, the pixel-wise square error between prediction ($\hat{\eta}$) and ground truth (η) is computed and averaged over all pixels ($N = 64\times64\times2$) and all samples of a batch ($M = 32$).

Given that the predicted velocity field is a vector field, to measure the prediction performance more intuitively, we calculate L^2 (Euclidean) norm of the velocity vectors ($\ell = \sqrt{x^2+y^2}$) to make the velocity field into a scalar field that leads to more straightforward quantification of the performance measurement. We also introduce the pixel-wise relative error (PRE) to compare the velocity norm between the prediction and ground truth:

$$PRE = \frac{|\hat{\ell}_i - \ell_i|}{\ell_i} \times 100\%, \quad i = 0, \ldots, N_n \tag{5}$$

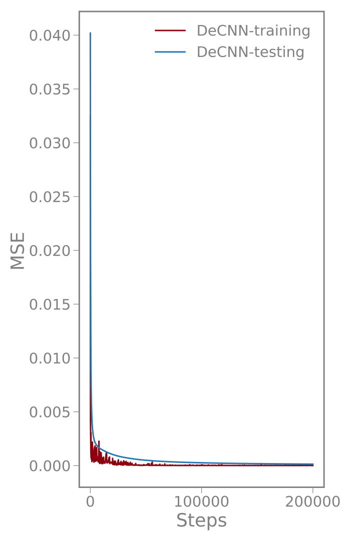
where N_n is the number of pixels ($N_n = 64\times64\times1$) in the velocity norm field. At the i-th pixel, $\hat{\ell}_i$ is the norm of the predicted velocity, and ℓ_i is the norm of the ground truth.

5.2 Performance of DeCNN

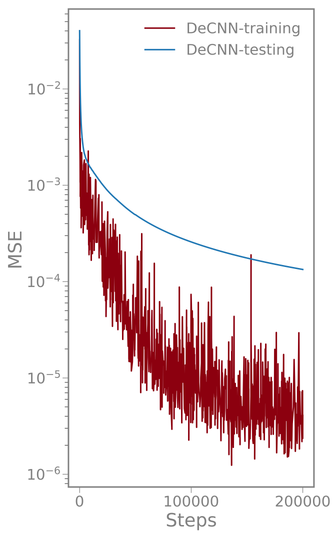
The MSE loss of the DeCNN model as a function of training steps (200,000 steps) is given in Figure 9, where the training loss is computed for a batch size (32) of training data, while the testing loss is for the entire testing data (9,600). The visualization of the testing results are given in Figure 10:

- *Input:* The input RGB image ($64\times64\times3$) for testing.
- *Prediction:* The RGB image ($64\times64\times2$ + zero padding of B-channel) of the predicted velocity field[6].
- *Ground truth:* The RGB image ($64\times64\times2$ + zero padding of B-channel) of the simulated velocity field.
- *Absolute discrepancy:* The absolute discrepancy of velocity fields between the prediction and ground truth at each pixel.
- *Scaled discrepancy:* The absolute discrepancy is scaled up to the range $[0,1]$ for better visualization.
- *Ground truth norm:* L^2 Norm of the simulated velocity vectors. Color map is given to visualize the magnitude.
- *Discrepancy norm:* L^2 Norm of the absolute velocity discrepancy between the prediction and ground truth.
- *Relative norm:* PRE of the velocity norm between the prediction and ground truth.

[6]For visualization purposes, the velocities are ensured in the positive range $[0,1]$ for valid RGB values in this paper.



(a)



(b)

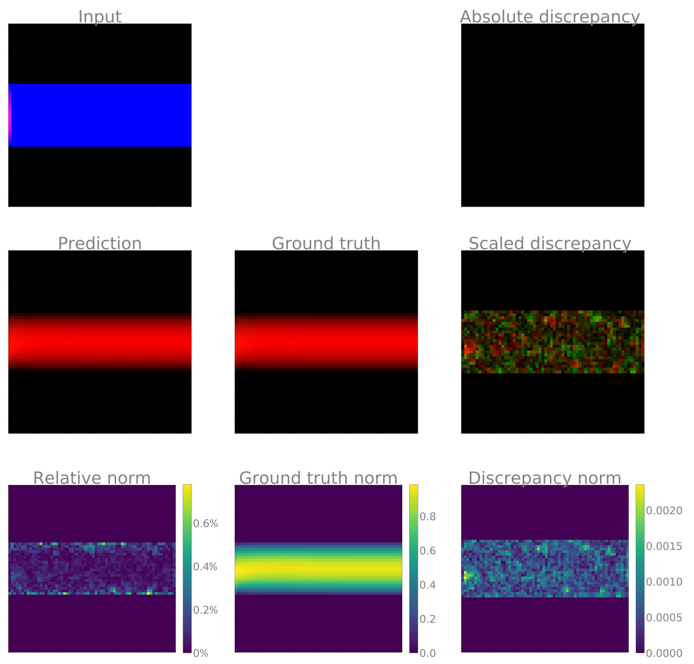
FIGURE 9: Training and testing loss of DeCNN model during the training history. MSE curves in (a) arithmetic coordinates and (b) logarithmic coordinates.

Key result for the DeCNN: As shown in Figure 10, the predictions are close to the ground truth. For example, the percentage error between the predicted and ground truth velocity norms (relative norm) is no higher than 3%. If we average the relative error across all valid sites ($\frac{1}{N_{valid}}\sum_{i=1}^{N_{valid}}\frac{|\hat{\ell}_i-\ell_i|}{\ell_i}$, where N_{valid} is the number of pixels in pipe area) on the testing samples, the trained DeCNN model can achieve an average prediction accuracy of about 98.6%. As we can see, large errors always occur at the pipe boundaries.

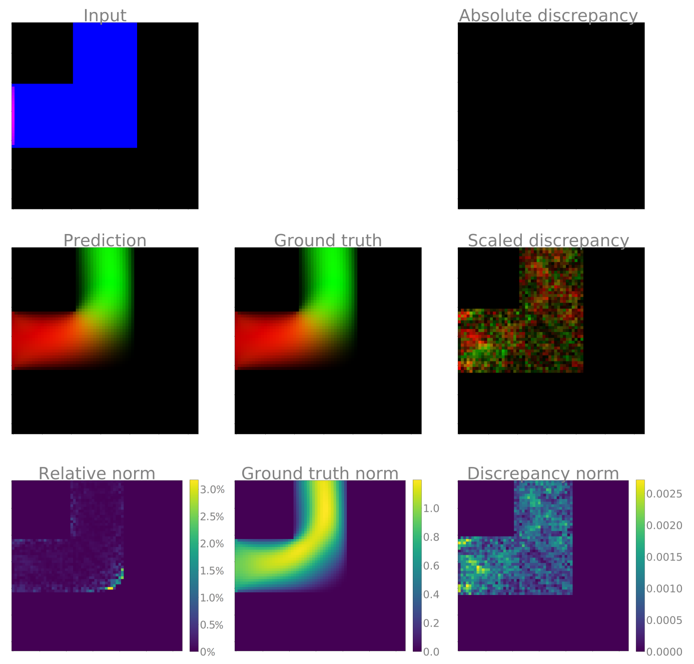
5.3 Performance of GNN

The MSE loss of the GNN model as a function of training steps (1,000,000 steps) is given in Figure 11, where the training loss is computed for a batch size (32) of training data, while the testing loss is for the entire testing data (6,400). The visualization of a testing result is given in Figure 13:

- *Input composition:* Two input RGB images for the two-node graph. The two input images represent the velocity fields of two consecutive pipe flows that are naively composed together. The two consecutive pipe flows are predicted using the DeCNN model.
- *Ground-truth composition:* A ground-truth image obtained directly from the long pipe flow simulation.
- *Naive composition discrepancy:* The absolute discrepancy of velocity fields between the input composition and ground-truth composition. The discrepancy has been scaled up for visualization purposes.
- *Naive discrepancy norm:* L^2 Norm of the naive composition discrepancy between the input composition and ground-truth composition.



(a)



(b)

FIGURE 10: Visualization of learn-to-abstract using DeCNN. (a) Straight pipe. (b) *L*-shape pipe.

- *Predicted composition:* Two output images corrected by the GNN model. The velocity fields of two naively composed consecutive pipe flows are corrected (especially at the interface) to make a smooth long pipe flow via the learning of the GNN model.
- *Predicted composition discrepancy:* The absolute discrepancy of velocity fields between predicted composition and ground-truth composition.
- *Predicted discrepancy norm:* L^2 Norm of the predicted composition discrepancy between the predicted composition and ground-truth composition.
- *Ground truth norm:* L^2 Norm of the simulated velocity field of the long pipe flow.
- *Input relative norm:* PRE of the velocity norm between the input composition and ground-truth composition.
- *Predicted relative norm:* PRE of the velocity norm between the predicted composition and ground-truth composition.

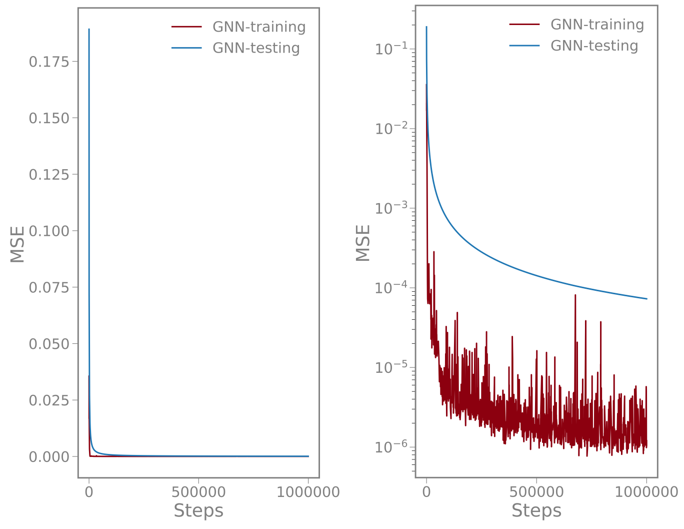


FIGURE 11: Training and testing loss of GNN model during the training history. MSE curves in (a) arithmetic coordinates and (b) logarithmic coordinates.

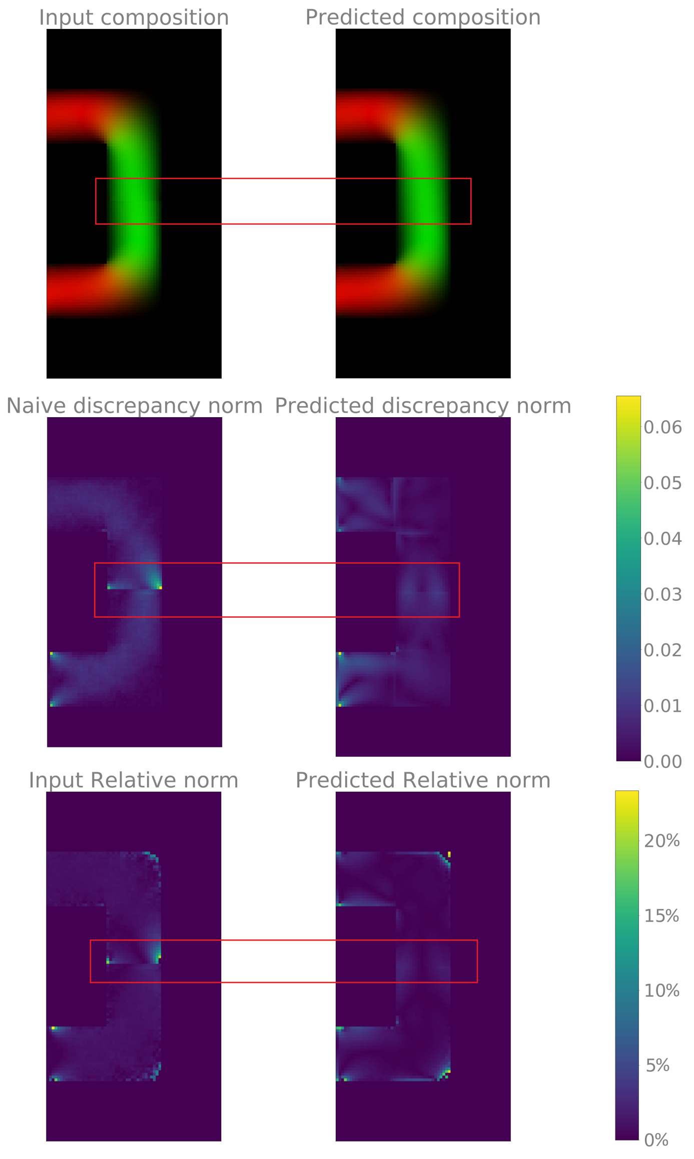


FIGURE 12: Composition comparison before and after using GNN. *Left:* Naive composition by DeCNN. *Right:* Modified composition after using GNN. The high errors at the connection interface (highlighted in the red box) are effectively corrected, and the pipe flow is thus smoothened by the GNN model.

Key result for the GNN: Figure 12 compares the composition of two *L*-shape pipes before and after using the GNN model to demonstrate the efficacy of our learn-to-compose model. As seen in Figure 12, the naive pipe flow composition predicted by the DeCNN model has been close to the ground truth but with discontinuity (high error) at the connection interface of two pipe segments. Our GNN model targets solving the discontinuity issue by correcting the high error at the connection interface. For the particular case in Figure 12, our GNN model can effectively reduce the relative (percentage) error at the interface from 20% to below 5%. By averaging the relative error across all valid sites, our GNN model can achieve an average prediction accuracy of about 98.2% (compared to 97.7% of the naive composition) on this example. Large errors also occur at the pipe boundaries. Table 2 summarizes the percentage errors of all the four types of compositions before and after using the GNN model. More detailed visualization of the four types of compositions is given in

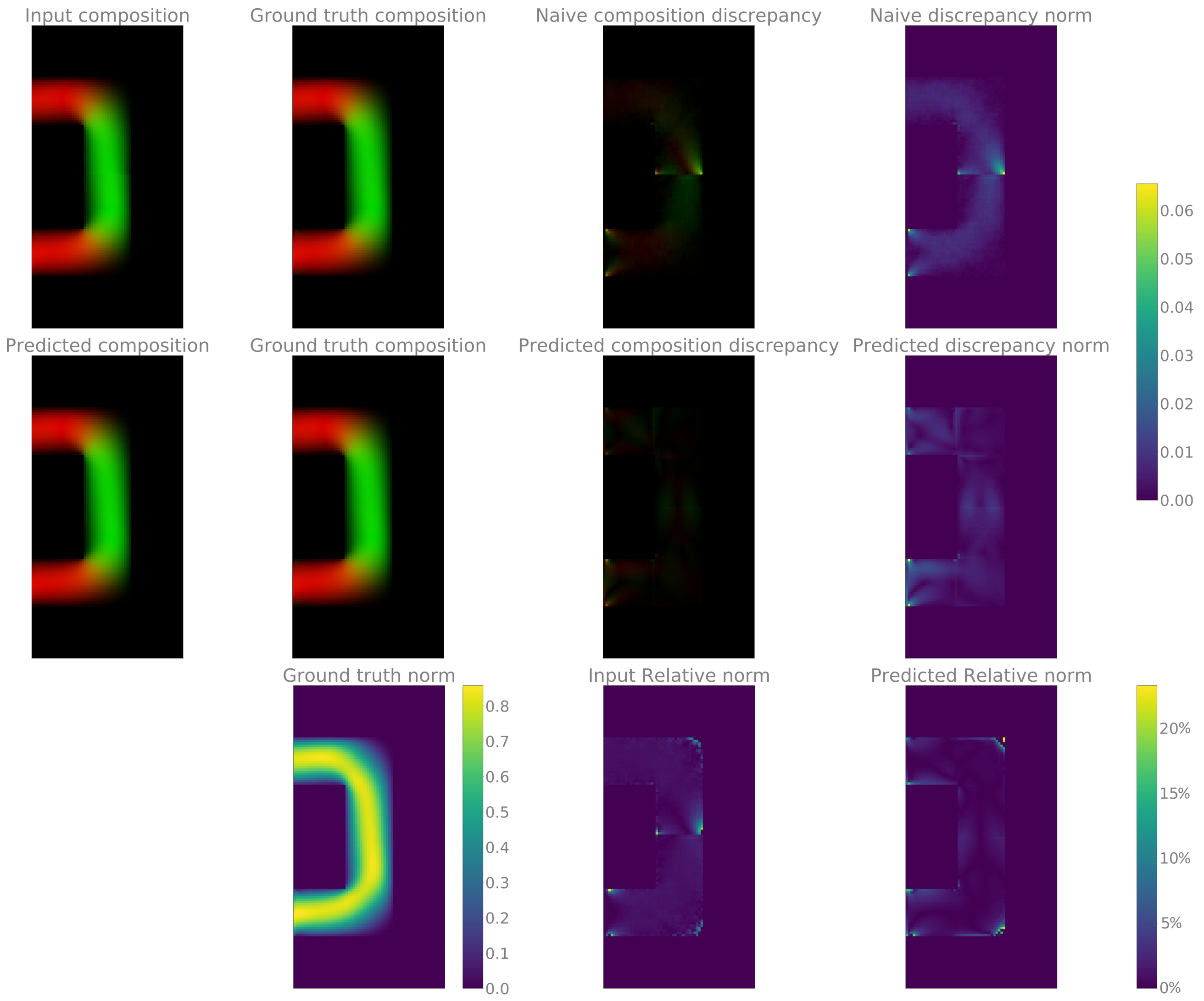


FIGURE 13: Visualization of learn-to-compose using GNN. This two *L*-shape pipe composition is shown as a representative of the other examples. Figure 15 in Appendix A displays all the four types of compositions.

Figure 13 and Figure 15 in Appendix A.

5.4 Discussion of Multi-Pipe Composition

We expect to generalize our GNN model for composition with any number of pipes, that is, the prediction performance should be unaffected by the composition length. We thus use the current GNN model to predict longer pipe compositions to see the effect of composition length on the prediction performance. For simplicity, We select all straight pipe components to compose longer pipe flows with increasing lengths (from two-pipe to nine-pipe composition). To enable a fair comparison, each pipe composition is applied with the same boundary conditions. The *mean predicted discrepancy norm* ($MPDN = \frac{1}{N_{valid}} \sum_{i=1}^{N_{valid}} |\hat{\ell}_i - \ell_i|$) for each adjacent two pipe segments is computed across multiple samples to measure the prediction performance. Figure 14 plots the MPDN values that depict the effect of the composition length on the prediction performance.

As seen in Figure 14, while our GNN model decently predicts the short-length compositions with lower errors (MPDN), it does not perform well when the composition length becomes longer (e.g., when the pipe number is greater than five). The prediction error keeps increasing with the composition length. We reason that the error increase is due to (1) the limited types of

TABLE 2: Percentage errors of compositions before and after using GNN

Combinations	Interface error		Average accuracy	
	Before	After	Before	After
Two straight	$> 8\%$	$< 2\%$	98.3%	99.8%
Straight+L-shape	$> 16\%$	$< 3\%$	98.0%	99.7%
L-shape+Straight	$> 25\%$	$< 5\%$	98.1%	99.1%
Two L-shape	$> 20\%$	$< 5\%$	97.7%	98.2%

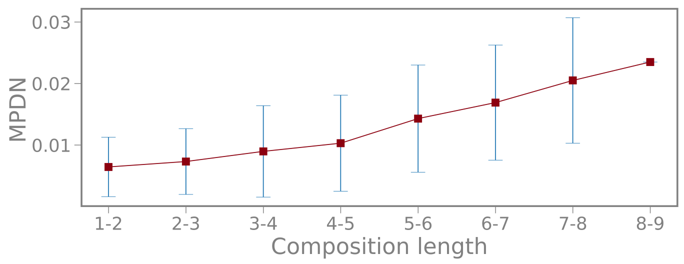


FIGURE 14: Effect of composition length on the prediction performance of GNN model. The x-axis represents each adjacent two pipes along multi-pipe composition (maximum 9 pipes). The red squares and blue bars indicate the MPDN and its standard deviation across multiple samples with different composition lengths.

inlet velocities for the first pipes in the GNN database, where the first node of the two-node graph is always fed with designed inlet velocity (using the polynomial Equation 2), and (2) the accumulative errors of the DeCNN model when using the predicted results of the previous pipe to predict the next pipe again. However, the proposed two models still show the potential for predicting more general cases. We believe these limitations could be mitigated by creating a more diverse database.

6 CONCLUSION

We proposed a computational pipeline for prediction of the fluid pipe flows and their composition behaviors, which can be used to demonstrate how NNs learn the abstraction and composition of phenomena when a single NN model cannot suffice in building different parts of a system. The functional aspects of the problems focused on the 2D velocity field prediction and correction for two types of basis pipe flows and their combinations. When pipe shapes (straight or L-shape) and boundary conditions (inlet velocity and outlet pressure) were fed into the computational pipeline, the first NN model predicted the velocity field of each basis pipe flow and the second NN model learned to compose a long pipe flow system by combining these basis pipes. This is achieved by developing a DeCNN model (convolutional autoencoder) to reconstruct the 2D velocity field and a GNN model (two-node graph network) to correct two naively connected velocity fields.

Our approach offered a solution for constructing multiple NNs that learn one physical phenomenon and also afford composition and automatic self-tuning of different components in a system. Our method demonstrated decent accuracy (around 99%) in the abstraction and composition of the pipe flow system.

Limitations and future work: Our current work demonstrated the DeCNN-GNN pipeline by using limited (two) types of pipes, and thus the possible combinations in pipe flow composition were limited as well. Besides the two-pipe composition illustrated in this work, we want the composition of more pipe flows to be validated. As future work, including more diverse pipe shapes and boundary conditions (i.e., more diverse inlet velocities) in the database, could be an approach to alleviating these limitations.

A natural extension for this work is to automate the pipe flow system's design by setting input parameters or rules to achieve output designs that meet the desired behaviors. Building upon AI algorithms, this could provide a set of solutions for generative design.

ACKNOWLEDGMENT

We acknowledge the funding and support provided by DARPA through their Fundamentals of Design program (#HR0011-18-9-0009). The views, opinions, and/or findings contained in this article are those of the author and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

REFERENCES

[1] Russell, B., 2009. *Principles of mathematics*. Routledge.
[2] Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., and Solenthaler, B., 2019. "Deep fluids: A generative network for parameterized fluid simulations". In Computer Graphics Forum, Vol. 38, Wiley Online Library, pp. 59–70.
[3] Nie, Z., Jiang, H., and Kara, L. B., 2020. "Stress field prediction in cantilevered structures using convolutional neural networks". *Journal of Computing and Information Science in Engineering,* ***20***(1).

[4] Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R., 2010. "Deconvolutional networks". In 2010 IEEE Computer Society Conference on computer vision and pattern recognition, IEEE, pp. 2528–2535.

[5] Xu, L., Ren, J. S., Liu, C., and Jia, J., 2014. "Deep convolutional neural network for image deconvolution". In Advances in neural information processing systems, pp. 1790–1798.

[6] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al., 2018. "Relational inductive biases, deep learning, and graph networks". *arXiv preprint arXiv:1806.01261*.

[7] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M., 2018. "Graph neural networks: A review of methods and applications". *arXiv preprint arXiv:1812.08434*.

[8] Shahani, A., Setayeshi, S., Nodamaie, S., Asadi, M., and Rezaie, S., 2009. "Prediction of influence parameters on the hot rolling process using finite element method and neural network". *Journal of materials processing technology,* ***209***(4), pp. 1920–1935.

[9] Kazan, R., Fırat, M., and Tiryaki, A. E., 2009. "Prediction of springback in wipe-bending process of sheet metal using neural network". *Materials & design,* ***30***(2), pp. 418–423.

[10] Umbrello, D., Ambrogio, G., Filice, L., and Shivpuri, R., 2008. "A hybrid finite element method–artificial neural network approach for predicting residual stresses and the optimal cutting conditions during hard turning of aisi 52100 bearing steel". *Materials & Design,* ***29***(4), pp. 873–883.

[11] Wang, J., Das, S., Rai, R., and Zhou, C., 2018. "Data-driven simulation for fast prediction of pull-up process in bottom-up stereo-lithography". *Computer-Aided Design,* ***99***, pp. 29–42.

[12] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P., 1998. "Gradient-based learning applied to document recognition". *Proceedings of the IEEE,* ***86***(11), pp. 2278–2324.

[13] Khadilkar, A., Wang, J., and Rai, R., 2019. "Deep learning–based stress prediction for bottom-up sla 3d printing process". *The International Journal of Advanced Manufacturing Technology,* ***102***(5-8), pp. 2555–2569.

[14] Liang, L., Liu, M., Martin, C., and Sun, W., 2018. "A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis". *Journal of The Royal Society Interface,* ***15***(138), p. 20170844.

[15] Sosnovik, I., and Oseledets, I., 2019. "Neural networks for topology optimization". *Russian Journal of Numerical Analysis and Mathematical Modelling,* ***34***(4), pp. 215–223.

[16] Stoecklein, D., Lore, K. G., Davies, M., Sarkar, S., and Ganapathysubramanian, B., 2017. "Deep learning for flow sculpting: Insights into efficient learning using scientific simulation data". *Scientific reports,* ***7***, p. 46368.

[17] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G., 2008. "The graph neural network model". *IEEE Transactions on Neural Networks,* ***20***(1), pp. 61–80.

[18] Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., et al., 2016. "Interaction networks for learning about objects, relations and physics". In Advances in neural information processing systems, pp. 4502–4510.

[19] Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P., 2018. "Graph networks as learnable physics engines for inference and control". *arXiv preprint arXiv:1806.01242*.

[20] Li, Y., Wu, J., Zhu, J.-Y., Tenenbaum, J. B., Torralba, A., and Tedrake, R., 2019. "Propagation networks for model-based control under partial observation". In 2019 International Conference on Robotics and Automation (ICRA), IEEE, pp. 1205–1211.

[21] Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A., 2017. "Visual interaction networks: Learning a physics simulator from video". In Advances in neural information processing systems, pp. 4539–4547.

[22] Kampffmeyer, M., Chen, Y., Liang, X., Wang, H., Zhang, Y., and Xing, E. P., 2019. "Rethinking knowledge graph propagation for zero-shot learning". In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 11487–11496.

[23] Wang, X., Ye, Y., and Gupta, A., 2018. "Zero-shot recognition via semantic embeddings and knowledge graphs". In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 6857–6866.

[24] Garcia, V., and Bruna, J., 2017. "Few-shot learning with graph neural networks". *arXiv preprint arXiv:1711.04043*.

[25] Lee, C.-W., Fang, W., Yeh, C.-K., and Frank Wang, Y.-C., 2018. "Multi-label zero-shot learning with structured knowledge graphs". In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1576–1585.

[26] Marino, K., Salakhutdinov, R., and Gupta, A., 2016. "The more you know: Using knowledge graphs for image classification". *arXiv preprint arXiv:1612.04844*.

[27] Wang, Z., Chen, T., Ren, J., Yu, W., Cheng, H., and Lin, L., 2018. "Deep reasoning with knowledge graph for social relationship understanding". *arXiv preprint arXiv:1807.00504*.

[28] Teney, D., Liu, L., and van Den Hengel, A., 2017. "Graph-structured representations for visual question answering". In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9.

[29] Narasimhan, M., Lazebnik, S., and Schwing, A., 2018. "Out of the box: Reasoning with graph convolution nets for factual visual question answering". In Advances in neural information processing systems, pp. 2654–2665.

[30] Qi, S., Wang, W., Jia, B., Shen, J., and Zhu, S.-C., 2018.

"Learning human-object interactions by graph parsing neural networks". In Proceedings of the European Conference on Computer Vision (ECCV), pp. 401–417.

[31] Liang, X., Shen, X., Feng, J., Lin, L., and Yan, S., 2016. "Semantic object parsing with graph lstm". In European Conference on Computer Vision, Springer, pp. 125–143.

[32] Landrieu, L., and Simonovsky, M., 2018. "Large-scale point cloud semantic segmentation with superpoint graphs". In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4558–4567.

[33] Qi, X., Liao, R., Jia, J., Fidler, S., and Urtasun, R., 2017. "3d graph neural networks for rgbd semantic segmentation". In Proceedings of the IEEE International Conference on Computer Vision, pp. 5199–5208.

[34] Alnæs, M. S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., and Wells, G. N., 2015. "The fenics project version 1.5". *Archive of Numerical Software,* ***3***(100).

[35] Masci, J., Meier, U., Cireşan, D., and Schmidhuber, J., 2011. "Stacked convolutional auto-encoders for hierarchical feature extraction". In International conference on artificial neural networks, Springer, pp. 52–59.

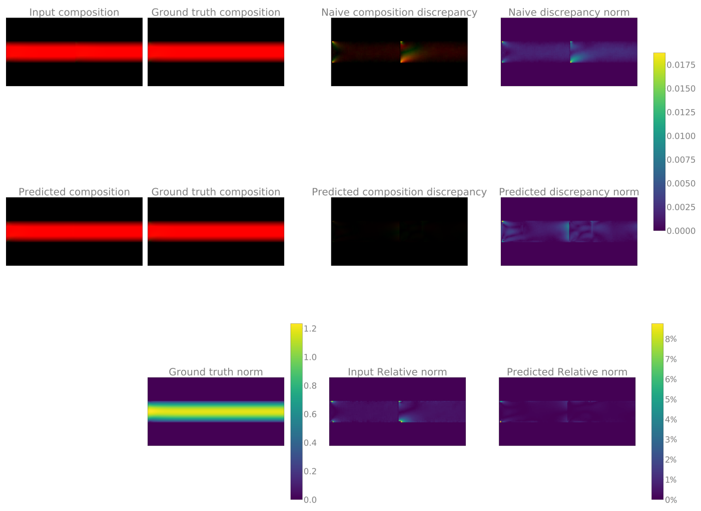
[36] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G., 2008. "Computational capabilities of graph neural networks". *IEEE Transactions on Neural Networks,* ***20***(1), pp. 81–102.

[37] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E., 2017. "Neural message passing for quantum chemistry". In Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, pp. 1263–1272.

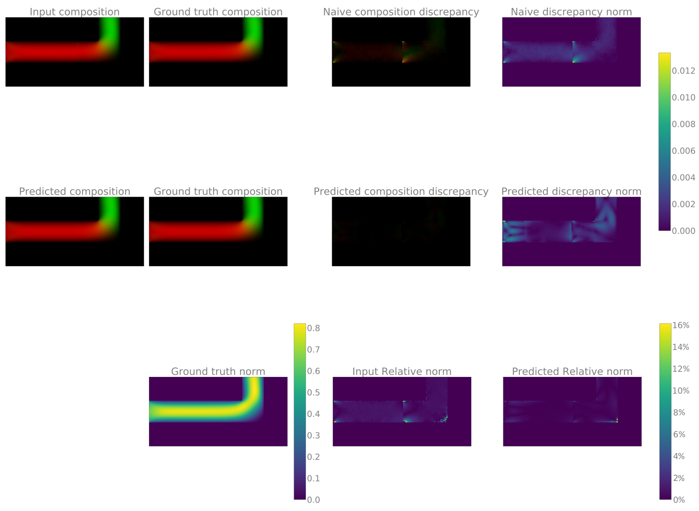
[38] Wang, X., Girshick, R., Gupta, A., and He, K., 2018. "Non-local neural networks". In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 7794–7803.

[39] Hamrick, J. B., Allen, K. R., Bapst, V., Zhu, T., McKee, K. R., Tenenbaum, J. B., and Battaglia, P. W., 2018. "Relational inductive bias for physical construction in humans and machines". *arXiv preprint arXiv:1806.01203*.

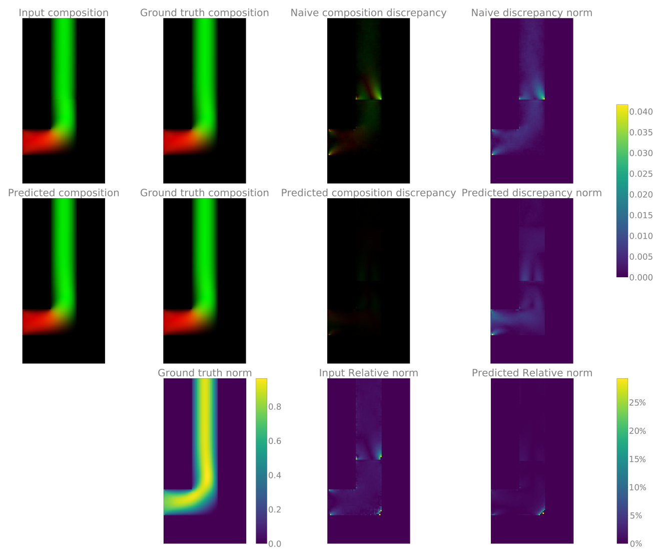
Appendix A: Visualization of Learn-to-Compose



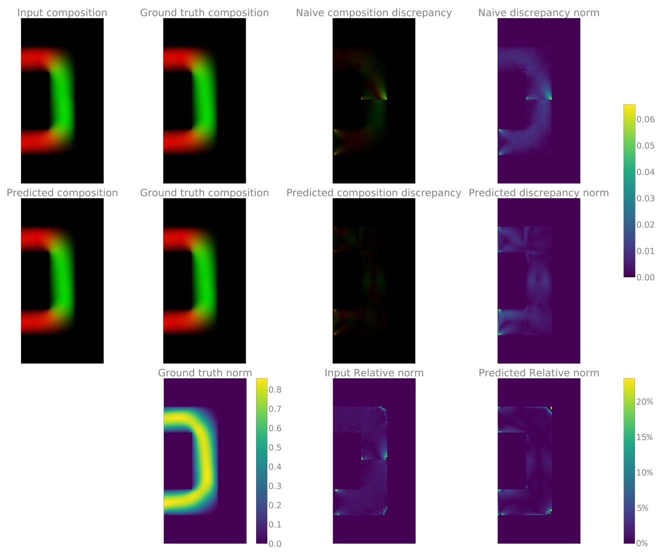
(a)



(b)



(c)



(d)

FIGURE 15: Visualization of learn-to-compose using GNN. (a) Two straight pipes. (b) Straight pipe + *L*-shape pipe. (c) *L*-shape pipe + straight pipe. (d) Two *L*-shape pipes.