Machine Learning for Physics-Informed Generation of Dispersed Multiphase Flow Using Generative Adversarial Networks

B. Siddani[a], S. Balachandar[a,*], W. C. Moore[a], Y. Yang[a], R. Fang[b]

[a]*Center for Compressible Multiphase Turbulence, University of Florida, Gainesville, FL 32611, USA*
[b]*J. Crayton Pruitt Family Department of Biomedical Engineering, University of Florida, Gainesville, FL 32611, USA*

arXiv:2005.05363v1 [physics.flu-dyn] 11 May 2020

Abstract

Fluid flow around a random distribution of stationary spherical particles is a problem of substantial importance in the study of dispersed multiphase flows. In this paper we present a machine learning methodology using Generative Adversarial Network framework and Convolutional Neural Network architecture to recreate particle-resolved fluid flow around a random distribution of monodispersed particles. The model was applied to various Reynolds number and particle volume fraction combinations spanning over a range of [2.69, 172.96] and [0.11, 0.45] respectively. Test performance of the model for the studied cases is very promising.

Keywords: Pseudo-turbulence, Multiphase Flow prediction, Generative Adversarial Network (GAN), Convolutional Neural Network (CNN)

1. Introduction

Dispersed multiphase flows are flow systems that contain dispersed elements, such as particles, droplets or bubbles that are distributed in a continuous phase [1]. These flows are prevalent both in nature and industry: Sediment transport, fluidized bed and pneumatic transport of gas-particle mixture are some widespread applications. It is intuitive that the existence of a dispersed phase affects the continuous phase, but the extent of their interaction is highly dependent on spatial distribution, size, density, volume fraction and other dispersed phase properties.

Fluid flow around a random distribution of stationary spherical particles within a periodic box can be considered as a geometrically-simplified canonical problem of substantial importance in the study of dispersed multiphase flows. This problem has been studied extensively both experimentally and numerically with the focus on better understanding pseudo turbulence generated by the arrangement of particles and the back effect of pseudo turbulence on the hydrodynamic forces on the particles. This understanding enables model development of pseudo turbulent Reynolds stress and parameterization of particle forces, which can be leveraged in more complex scenarios like particle-laden flows and flow through packed-beds. Due to its fundamental significance, the problem of flow through a random distribution of stationary particles has been simulated over a range of Reynolds numbers and volume fractions using a variety of numerical methods: Finite Volume Methods [2], Immersed Boundary Methods, [3, 4, 5, 6] and Lattice Boltzmann Methods [7, 8, 9].

The rapid advancement of Machine Learning (ML) algorithms and computer technology in recent years has brought about great interest in its many applications. The impacted applications of interest are not only from the field of Computer Science but also from other streams of sciences like physics, chemistry, various branches of engineering, and data analytics. This interest is mainly due to the success of ML in solving the complex data-rich problems of different domains. In particular, in the field of fluid mechanics ML has found many emerging applications, which have been discussed in recent reviews [10]. For example, Duraisamy et al. [11] addressed the use of ML in turbulence modeling, ML applications in flow visualization using convolutional neural networks (CNN) was discussed in Guo et al. [12], and prediction of flow over an

*Corresponding author
Email address: bala1s@ufl.edu (S. Balachandar)

airfoil using CNN was presented in Bhatnagar et al. [13]. Raissi et al. [14] introduced *Physics Informed Neural Networks* (PINN) based on the idea that neural networks designed for physical systems should obey the governing equations of these systems. This approach was also successful in reproducing the underlying partial differential equations of systems [15]. Recently, PINNs were also used to approximate Euler equations in one-dimensional and two-dimensional domains [16]. In the domain of multiphase flows, Qi et al. [17] used ML for computing curvature for Volume of Fluids methods. Of particular relevance is the recent work of Farimani et al. [18] who used a conditional variant of generative adversarial network (GAN) [19, 20] to generate synthetic steady-state velocity and pressure fields in 2D incompressible cavity flows. Similarly, Xie et al. [21] proposed their tempoGAN ML architecture, which is a temporally coherent generative adversarial network model, for obtaining super-resolution of fluid flows from an input of coarse-grained flow fields.

Here we will present a ML methodology that will recreate particle-resolved fluid flow within a periodic box of random distribution of monodispersed particles. This work will make use of multiple CNNs in combination with the GAN algorithm to achieve its objective. In doing so the following three questions must first be addressed: (i) what information regarding the random distribution of particles and the external mechanism driving the flow is needed as *input* to the ML algorithm? (ii) what precise information regarding the flow needs to be predicted as *output* by the ML algorithm? and (iii) what prior information connecting the input to the output will be provided as training data in order to train the ML algorithm? These three aspects of ML methodology will be introduced below and will be elaborated in this paper.

The flow prediction question that we plan to address with the ML methodology is as follows: Given the location and motion of a *reference* particle within a random distribution of particles in a multiphase flow, along with the location and motion of its few nearest neighbors (say for example 10 nearest neighbors), we desire to accurately predict the flow around the reference particle. Important distinctions must be made between the above ML quest and the prediction goal of conventional computational fluid dynamics (CFD) approach. In a particle-resolved direct numerical simulation (PR-DNS) of flow over an array of randomly placed particles, a large periodic domain of size much larger than the correlation length is chosen containing $O(10^3)$ or more particles [7, 22, 9, 4, 5, 23, 24, 25]. The domain is then discretized with millions of grid points and the flow around all the particles over the entire computational domain is simultaneously solved. Such a *global approach*, where the flow around all the particles is solved simultaneously, is neither feasible with the current ML algorithms, nor necessary due to the manner in which ML predicts the flow. Therefore, in contrast to conventional CFD, a *local approach* will be pursued with ML, where the flow around each particle is predicted with a deterministic knowledge of the precise state of a few nearby particles, and a statistical knowledge of distribution of all other distant particles. Once the flow associated with each particle is determined, the overall flow can be obtained through appropriate superposition.

In the present study, following the PR-DNS simulations mentioned above, we will restrict attention to steady flow over a random distribution of stationary particles. In this limit, when each particle within the system is chosen as the reference particle, the *input* to ML is the list of relative position of its nearest neighbors that are located within a chosen neighborhood (i.e., location of neighbors within a sub-volume centered around the reference particle). In addition, the *input* will include the Reynolds number of the flow within the sub-volume calculated based on the average fluid velocity within the sub-volume and the particle diameter. The desired *output* of ML is the accurate prediction of the velocity and pressure fields within the sub-volume, which can be further refined to focus only on the region in the immediate vicinity of the reference particle, without extending into the neighboring particles.

We now address the last question pertaining to the training data. The results of PR-DNS simulations of flow over a random distribution of particles for a range of Reynolds number and volume fraction [26, 25] will be used for both training and testing purposes. The raw simulation results will be curated to obtain the input and output data for each particle of the random array for training the ML algorithm. This curation is an important step, since it allows the number of training samples to scale as the number of particles within a CFD computational box. For each case (i.e., for each Reynolds number and volume fraction combination) we consider multiple CFD realizations to ensure sufficient amount of training data for fully converged training. Most of the CFD realizations will be used for training, but a few that are not used for training will be reserved for testing.

A simpler version of this flow prediction problem has recently been achieved within the framework of pairwise interaction extended point-particle (PIEP) approach [27, 28]. The perturbation flow induced by each particle was defined as its superposable wake, which was taken to depend only on the average statistical

presence of all its neighbors, and thus was parameterized as a function of the local particle Reynolds number and volume fraction. The key aspect of the PIEP approach was that the flow around any particle can be calculated as a summation of its perturbation flow along with those of all its neighbors. As a result, in this approach, the *input* to prediction of flow around a reference particle is still the relative position of its neighbors. However, the important simplification comes from the assumption that the influence of each neighbor can be taken pairwise, i.e., one at a time. Due to this assumption, the predicted flow was observed to range in accuracy from 56% to 83%, for varying Reynolds number and volume fraction.

Here we pursue a more advanced ML algorithm, which allows us to account for the perturbation flow of all the neighbors taken together without the pairwise approximation. By addressing this multi-particle interaction problem directly without the pairwise interaction assumption, we attempt to predict the flow around a random distribution of particles far more accurately than the PIEP model.

The paper describes a successful implementation of the generative adversarial network (GAN) methodology and architecture that models the Navier–Stokes equation and generates accurate dispersed multiphase flow. Once developed, the GAN can generate velocity and pressure fields around a random distribution of particles at a computational cost that is orders of magnitude cheaper than conventional CFD. However, we must emphasize two important facts: (i) CFD results are needed in the first place to train GAN, and (ii) the GAN results, though quite accurate, will not perfectly reproduce the CFD results. Nevertheless, the ability to very quickly generate synthetic flows over large arrays of particles will be of great value, since this capability can provide us with new opportunities. For example, by considering very large multiphase flow systems or by considering many repeated realizations, more converged higher-order statistical information of pseudo turbulence can be obtained. Similarly, the large amount of synthetic data can be used to develop better particle force models that depend not only on average Reynolds number and volume fraction, but also on the relative location of nearby neighbors.

Summary of succeeding sections of this paper is as follows. In section 2, Direct Numerical Simulation (DNS) methodology and analysis of the data used in this work is presented. In section 3, the GAN methodology is explained. Section 4 deals with performance of the proposed GAN model and discussion of the results. The presented neural network and GAN methodology can also act as a well-defined starting point to design more compact networks for similar applications.

2. Simulation Data and Its Curation

Direct Numerical Simulation (DNS) results are required for the training of a neural network to produce synthetic flows, and the current work makes use of simulation results presented in [27, 6]. The basic geometric structure of these simulations is a random distribution of stationary, monodispersed spherical particles in a cubic domain. The non-dimensional diameter of the particles is unity. They are randomly distributed within the domain with uniform probability, and the size of the cubic box is 3π. The domain is subjected to periodic boundary conditions along x, y directions. Symmetric, no-stress boundary conditions are applied along the z direction. A mean macroscale pressure gradient is imposed along the x direction making it the streamwise direction. A grid resolution of $376 \times 376 \times 401$ points along x, y, and z, respectively, ensures that the fluid is fully resolved around each particle in every simulation. Incompressible Navier-Stokes equations in non-dimensional form are solved to steady state with no-penetration and no-slip boundary conditions on each particle's surface using immersed boundary method. The different cases simulated can be categorized using two macroscale parameters: particle Reynolds number (Re) and particle volume fraction (ϕ). A list of all the cases considered is presented in Table 1. Akiki et al. [25] considered only the inner 64% along z direction to avoid the effects of no-stress boundary condition. This inner portion will be referred to as the *inner flow region* in succeeding sections of this paper.

The variables have been non-dimensionalized using particle diameter d_* as length scale, the mean streamwise fluid velocity within the inner flow region U_* as velocity scale, and $\rho_* U_*^2$ as the pressure scale. Here, ρ_* denotes fluid density. Here and henceforth in this study, all dimensional quantities will be indicated with $*$ as subscript and all non-dimensional quantities will be represented without the subscript.

2.1. Sub-domain Selection

As discussed in the introduction, though the raw DNS results of each realization includes flow information over all the particles within the entire computational domain, each training data set will be centered around

Case	Re	ϕ	Realizations	σ_u	σ_v	σ_w	σ_p	N_{tr}	N_{te}
1	39.47	0.11	10	0.5026	0.1810	0.1844	0.2749	418	46
2	69.88	0.11	10	0.5031	0.1781	0.1806	0.2357	418	46
3	172.96	0.11	10	0.4968	0.1947	0.1960	0.1973	418	46
4	16.49	0.21	8	0.6131	0.2821	0.2877	0.7473	636	91
5	86.22	0.21	7	0.5960	0.2580	0.2609	0.3807	635	90
6	2.69	0.45	5	0.8204	0.4740	0.4732	17.3688	772	193
7	20.66	0.45	5	0.8227	0.4584	0.4631	2.9072	772	193
8	114.60	0.45	5	0.8006	0.4340	0.4421	1.0467	772	193

Table 1: The Reynolds number and volume fraction of the different cases considered, the corresponding RMS values of u, v, w, p and the average size of training and testing datasets

an individual *reference* particle and will cover only a portion of the computational domain centered around the particle. To decrease the effects of the top and bottom no-stress boundaries in the training date set, only the particles that are inside the inner 27% along z direction will be considered as reference particles. This inner 27% portion along the z direction will be referred to as *inner particle region*. Thus, each realization of each case listed in Table 1 will yield N_i training data sets, where N_i is the number of particles within the inner particle region of the i^{th} realization. The first step of this curation process is to determine the size of the sub-domain that will surround the reference particle, so that the flow information within this sub-domain will form the training data of the reference particle.

The size of the sub-domain will be chosen to satisfy the following properties: (i) It must be large enough to contain sufficient number of neighbors, since the relative location of these neighbors will be the key input to the ML algorithm. In this regard, we require the sub-domain to be larger along the streamwise direction than along the transverse directions, since the neighbor's influence on the reference particle is expected to be dominant along the flow direction. (ii) The sub-domain must not be too large, for otherwise the implementation of the three-dimensional (3D) ML algorithm will be computationally impossible (this issue will be elaborated later); (iii) The average properties of the flow within the sub-domain must be reasonably close to the ensemble average. This condition has been imposed so that variation in the average flow properties around the different particles when chosen as reference is mainly due to the distribution of its neighbors and not due to differences in the average macroscale flow within their sub-domain.

In order to identify the sub-domain size that satisfies the above requirements, we first calculate the mean and rms values of velocity components and pressure of all the cases. These statistics were first computed for the inner flow region of the DNS taking into account all the realizations. These global mean and rms values define the reference against which corresponding values computed over the sub-domains can be compared to determine the appropriate sub-domain size. Since the mean pressure gradient was applied only along the x direction, mean velocities along y and z are zero and the mean non-dimensional streamwise velocity is unity. Standard deviations of u, v, w, and p are also tabulated in Table 1.

For a given particle volume fraction, the rms values of velocities do not show a strong variation to changes in Re, however, they tend to increase with volume fraction. From Table 1, the rms values of streamwise and transverse velocities can be approximated as follows:

$$\sigma_u \approx 0.6444\ [1.4\phi + 0.63]\ , \quad \sigma_v \approx 0.3075\ [2.6\phi + 0.33]\ . \tag{1}$$

It was also observed by Moore et al. [27] that rms values of pressure for these cases show a very good correlation with $\phi\, C_D$, where C_D is non-dimensional volume fraction-dependent mean drag on the particles [4]. This correlation between p and $\phi\, C_D$ is given below and depicted in Figure 1.

$$\sigma_p \approx 1.2505\, \phi C_D + 0.034948\,, \tag{2}$$

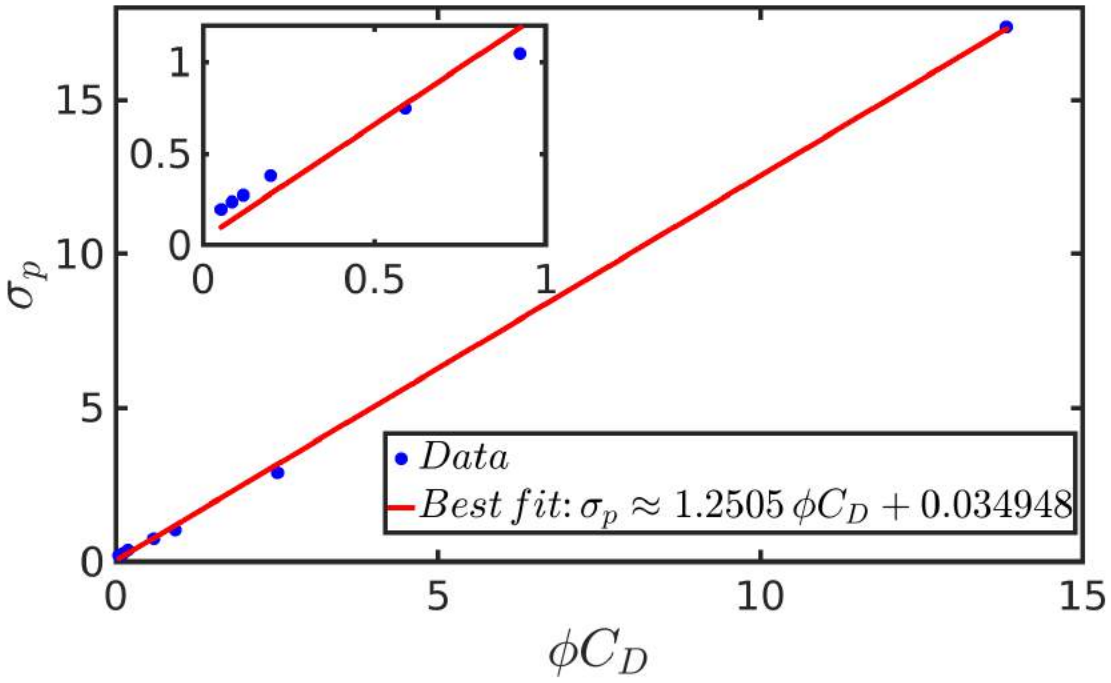


Figure 1: σ_p vs ϕC_D for all cases.

where

$$C_D = 3\pi \frac{(1-\phi)}{Re_s}\left[\frac{1+0.15Re_s^{0.687}}{(1-\phi)^3} + \frac{5.81\phi}{(1-\phi)^3} + \phi^3 Re_s\left(0.95 + \frac{0.61\phi^3}{(1-\phi)^2}\right)\right]$$
$$Re_s = \frac{Re}{(1-\phi)}\,.$$

The mean and rms values presented in Table 1 are for the entire inner flow region. We now proceed to calculate the mean and rms for the sub-domains in the following manner: (i) Choose the sub-domain size to be tested; (ii) In the i^{th} DNS realization of a case, define sub-domains centered around each of the N_i particles within the inner particle region. This will result in $N_c = \sum_{i=1}^{M_c} N_i$ sub-domain information for the training of the ML algorithm, where the sum is over all the M_c realizations of the case; (iii) A grid resolution comparable to that used in the DNS is used to discretize each sub-domain and the flow variables at these grid points are obtained using linear interpolation of the PR-DNS data.

The streamwise and transverse mean velocities ($\langle u\rangle$ & $\langle v\rangle$, $\langle w\rangle$) computed for the different domain sizes are shown in Figure 2. The results of the 8 different cases are shown. It can be seen that for all choices of sub-domain size in every case the transverse mean velocities are less than 1% of the streamwise mean velocity. Furthermore, domains of size $4\times4\times4$ and larger are within 1% of the global average value of unity. Hence, it can be considered that the mean Reynolds number of each sub-domain is primarily determined by the global mean streamwise velocity. A similar conclusion can be drawn for the estimation of mean particle volume fraction within each sub-domain. As a result, in each of the eight cases listed in Table 1, the appropriate Reynolds number and volume fraction of all the sub-domain data are the same as the macroscale value.

The average value of rms velocity and pressure variation within the sub-domain is defined as follows:

$$\sigma_{u,sd} = \frac{\sum_{i=1}^{N_c} \sigma_{u,i}/N_c}{\sigma_u} \quad \text{and} \quad \sigma_{p,sd} = \frac{\sum_{i=1}^{N_c} \sigma_{p,i}/N_c}{\sigma_p}\,, \tag{3}$$

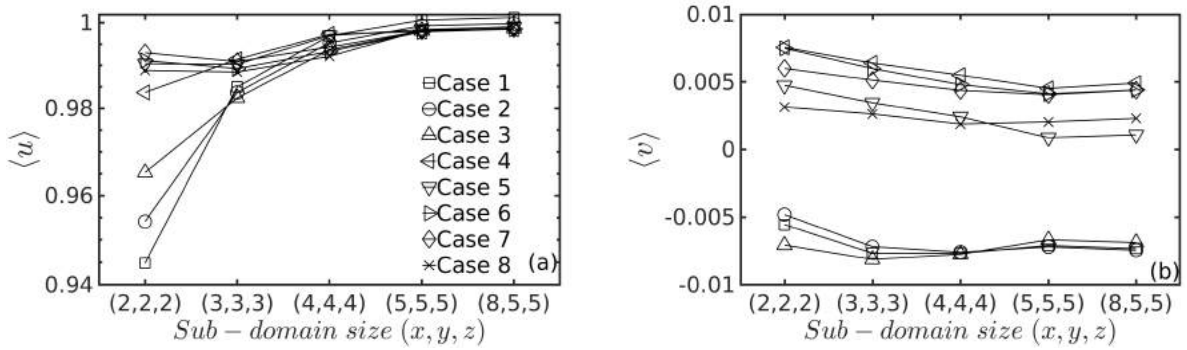


Figure 2: Variation of mean (a) streamwise and (b) transverse velocities with sub-domain size

where N_c is the total number of sub-domain samples of a case. Thus, $\sigma_{u,sd}$ defines the rms of streamwise velocity computed within each sub-domain, then averaged over all the sub-domains and normalized by the global rms value. Similar definitions apply for v and w velocity components as well. In the above equation σ_u, σ_v, σ_w and σ_p are the global rms velocity and pressure fluctuations computed for the entire flow data and are listed in Table 1 for all the cases. The effect of sub-domain size on normalized rms of u, v and p are shown in Figure 3. The general trend is that they increase with increasing sub-domain size. Based on these plots we choose a sub-domain of non-dimensional size $8 \times 5 \times 5$, along the x, y and z directions for further investigation, as it satisfies all the requirements discussed above.

As the final step of data curation we consider normalization of the velocity and pressure fields. Normalization of all inputs/outputs of a neural network to a similar scale is necessary to avoid biased preference to any particular input/output scales by weights in the network. This is achieved by normalizing velocity and pressure perturbations of the i^{th} sample as:

$$u' = \frac{u - \langle u \rangle}{3\sigma_{u,i}}, \quad v' = \frac{v - \langle v \rangle}{3\sigma_{v,i}}, \quad w' = \frac{w - \langle w \rangle}{3\sigma_{v,i}}, \quad p' = \frac{p - \langle p \rangle}{3\sigma_{p,i}} \tag{4}$$

here, $\sigma_{u,i}$, $\sigma_{v,i}$, $\sigma_{p,i}$ are rms of streamwise and transverse velocities, and pressure respectively of the i^{th} sample obtained using (1) & (2). Note that the mean values of streamwise velocity is unity while the mean transverse velocities are zero. By scaling with three sigma, u', v', w', p' are essentially normalized to be between -1 and 1.

3. GAN Methodology

Our philosophy for the model presented in this paper is as follows. At a high level of spatial resolution, applying the ML algorithm over the entire inner particle region with all the particles in it as one single data set is not computationally feasible. This motivates the use of sub-domain of size $8 \times 5 \times 5$ which was chosen based on the requirements described in section 2.1. Most importantly, with this choice of sub-domain size, the prediction of the flow around the reference particle will be well informed by all its immediate neighbors. This step also greatly increases the number of data sets to be used in the training and testing processes. Even with the reduced size of the sub-domain, the number of grid points required to obtain resolution similar to that used in PR-DNS is still significantly large. Therefore, the GAN architecture will be used to predict only a coarse-grained mesoscale flow within the sub-domain. The process of improving the accuracy of flow prediction in the immediate neighborhood of the reference particle to the desired spatial resolution is achieved with an attention mechanism. We choose the size of the innermost domain of high resolution to be of size $2 \times 2 \times 2$ centered around the reference particle and term this the *attention-domain* of the particle. This choice is motivated by the fact that such an attention-domain will yield flow information in an immediate region of one diameter around the particle. The above detailed approach of the model has been schematically presented in Figure 4.

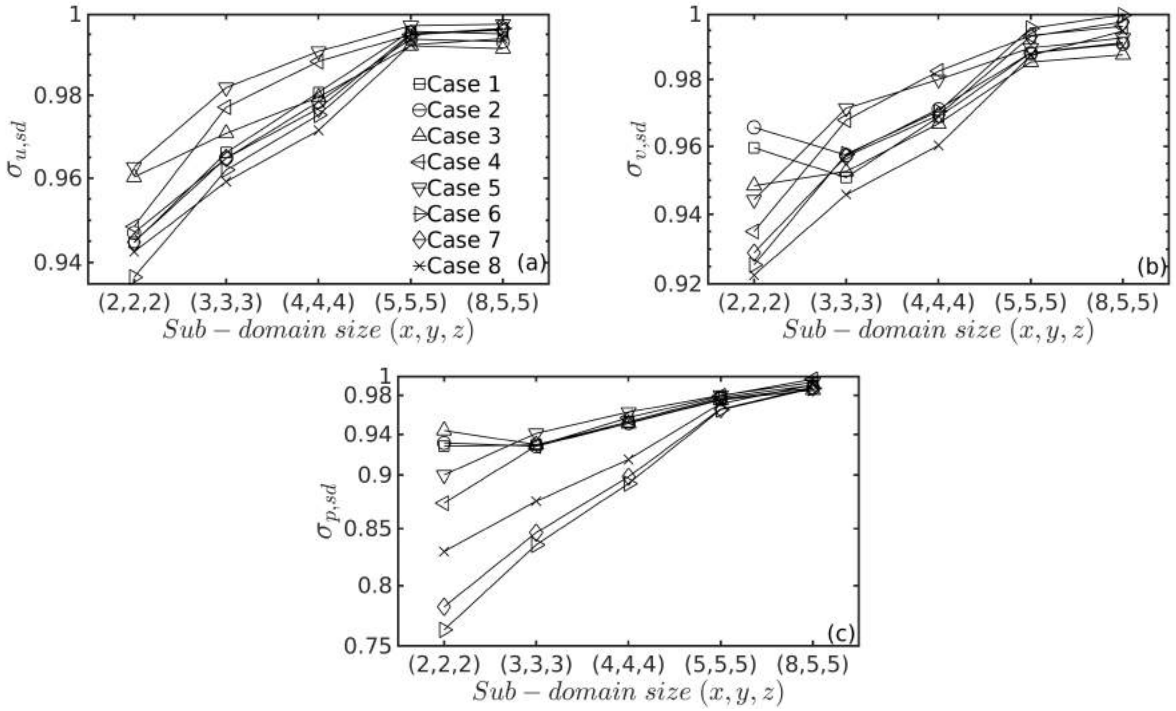


Figure 3: Effect of sub-domain size on: (a) RMS of u, (b) RMS of v, and (c) RMS of p

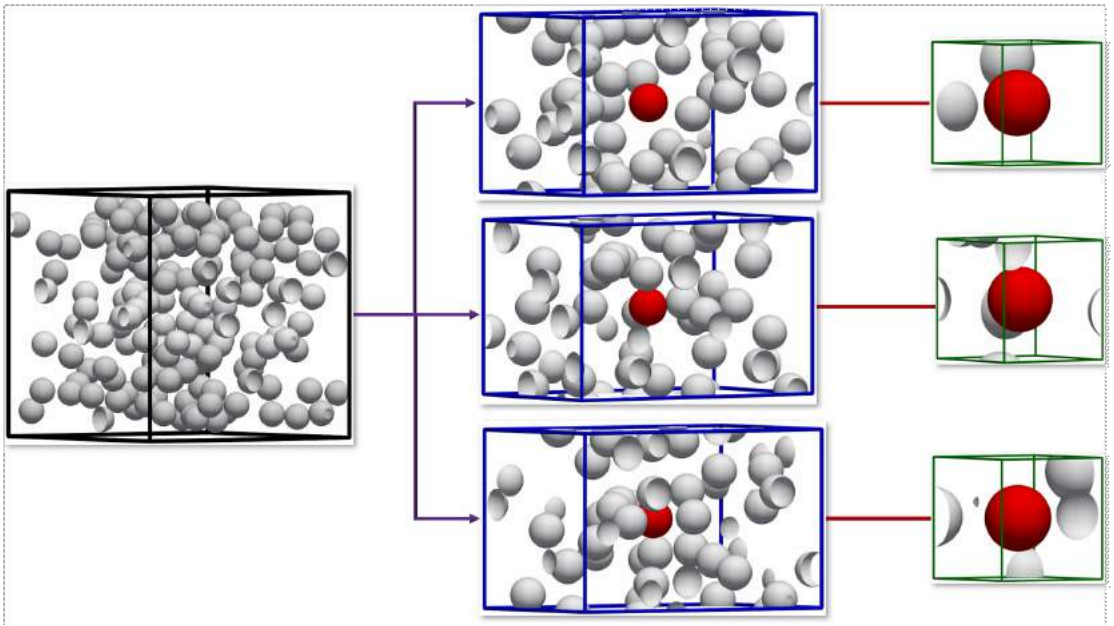


Figure 4: *Model Approach*: The 3π cubic domain bounded by the black outline on the left column corresponds to a CFD realization of 11% volume fraction. *sub-domains* of size $8 \times 5 \times 5$ are defined around every particle within the inner particle region of the cubic domain. Three such sub-domains created around three different particles as the reference particle is shown in the middle column. In each sub-domain (bounded by the blue cuboid) the reference particle is distinctly depicted by the red colored sphere. Coarse-grained mesoscale flow prediction within the sub-domain is obtained using the GAN architecture. This prediction is used to further obtain a highly resolved solution within $2 \times 2 \times 2$ *attention-domain* centered around the reference particle, which for the three sub-domains are shown on the right column bounded by green outline.

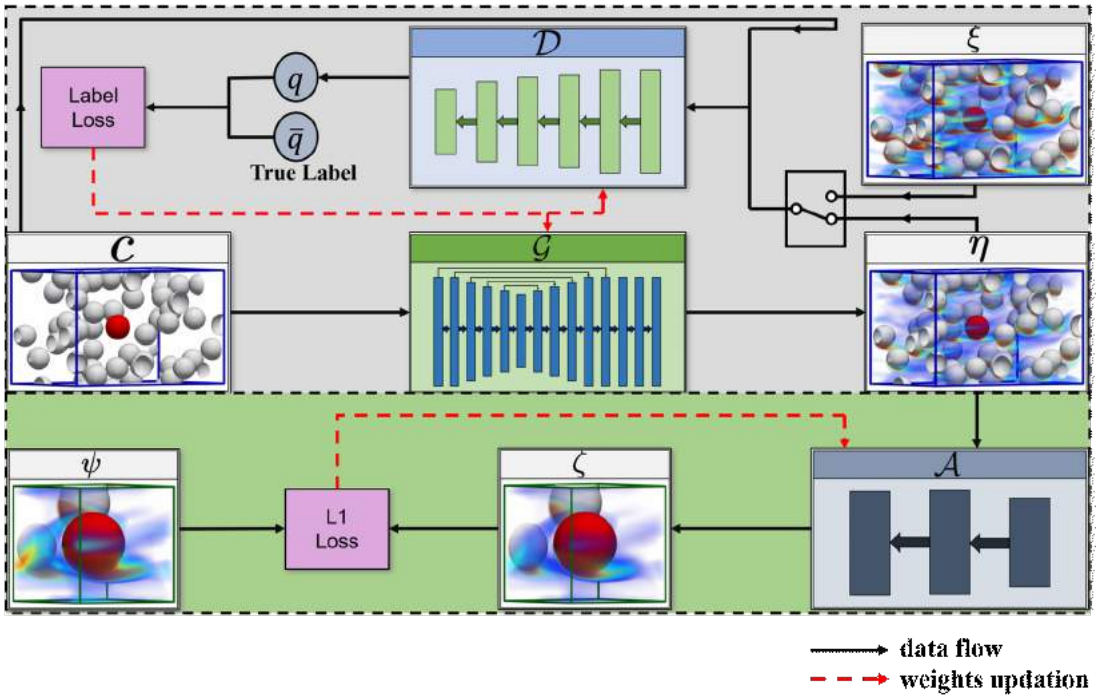


Figure 5: Model Framework.

Generative Adversarial Networks, abbreviated as GANs, are generative ML models that make use of adversarial learning process to produce artificial data that mimics true data. GANs primarily comprise of two neural networks, namely, *Generator* ($\mathcal{G}$) and *Discriminator* ($\mathcal{D}$). The purpose of a generator is to create synthetic data as close to the true data as possible so that the two cannot be differentiated by the discriminator, while the purpose of a discriminator is to correctly distinguish the true data from the synthetic data created by the generator. From the functioning of these two networks it can be observed that they compete against each other. This adversarial nature with the discriminator helps the generator in learning the target properties of the complex flow to be modeled.

In order to mathematically describe the optimization process of the competition between the generator and the discriminator, here we adopt the following notation [29]. Let $\boldsymbol{c}$ represent a sample input data. In the present problem, the input $\boldsymbol{c}$ corresponds to information on the relative position of all the neighbors within the sub-domain centered around the reference particle, along with the mean Reynolds number and volume fraction of the case. Now let $\boldsymbol{\xi}$ be the true DNS velocity and pressure fields obtained around the particles within the sub-domain. Let $\boldsymbol{\eta}$ be the synthetic velocity and pressure fields generated around the particles within the sub-domain by the generator $\mathcal{G}$. Thus, for the input condition $\boldsymbol{c}$, the output of the generator is $\boldsymbol{\eta}$, while the true target output is $\boldsymbol{\xi}$. We now define the following probability distributions:

P_c: is the distribution of all possible input condition, which in the present case corresponds to all possible distribution of neighboring particles.

P_ξ: is the distribution of DNS velocity and pressure fields corresponding to all possible values of input.

P_η: is the distribution of generated synthetic velocity and pressure fields corresponding to all possible values of input.

With this definition, each data set consists of a condition $\boldsymbol{c} \in P_c$ and the corresponding flow field $\boldsymbol{\xi} \in P_\xi$. The generator of the GAN will be trained such that given the condition $\boldsymbol{c}$ it should generate the synthetic flow $\boldsymbol{\eta} \in P_\eta$. The goal is then for $\boldsymbol{\eta}$ to as closely mimic $\boldsymbol{\xi}$ as possible. Note that for a given $\boldsymbol{c} \in P_c$, the corresponding true flow $\boldsymbol{\xi}$ and synthetic flow $\boldsymbol{\eta}$ will also depend on Reynolds number and volume fraction.

The working of GAN algorithm has been pictorially represented in Figure 5 [20, 30]. The output of the discriminator ($\mathcal{D}$) is a scalar value, q, between 0 and 1, with 0 being the expected output, $\bar{q}$, if the

discriminator identifies its input to be synthetic and 1 being the expected output if it identifies the input as real. Thus, as shown in the figure, $\mathcal{D}$ tries to optimize its weights to produce a value close to one if its input is $\boldsymbol{\xi} \in P_\xi$ and a value near zero for $\boldsymbol{\eta} \in P_\eta$ as input. On the contrary, $\mathcal{G}$ optimizes its coefficients in such a way that its output $\boldsymbol{\eta}$ will fool the discriminator to yield $\mathcal{D}(\boldsymbol{\eta})$ closer to one. This entire process can be represented using objective function ($V(\mathcal{D}, \mathcal{G})$) given by

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{\boldsymbol{\xi} \in P_\xi} \left[\log \mathcal{D}(\boldsymbol{\xi})\right] + \mathbb{E}_{\boldsymbol{c} \in P_c} \left[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{c})))\right] , \tag{5}$$

where $\mathbb{E}$ stands for expectation. The first term on the right is the contribution to the objective function for a true field $\boldsymbol{\xi} \in P_\xi$ as input, which the discriminator tries to maximize. The second term on the right is the contribution to the objective function for a synthetic field $\boldsymbol{\eta} \in P_\eta$ as input, which the discriminator tries to maximize, while the generator tries to minimize. The adversarial networks $\mathcal{D}$ and $\mathcal{G}$ are pitted against each other and upon iteration, they approach what is known as Nash equilibrium in game theory. The training process thus theoretically achieves convergence when discriminator cannot differentiate a true sample from a generator generated synthetic sample, i.e., $\mathcal{D}(\boldsymbol{\xi}) = \mathcal{D}(\boldsymbol{\eta}) = 0.5$. This also implies that the generator has efficiently learnt the characteristics of the true flow given the condition $\boldsymbol{c}$.

However, Mescheder et al. [31] show that a GAN training with objective function (5) does not always converge for data whose distribution is not absolutely continuous. They also show that instance noise and zero-centered gradient penalty methods stabilize GAN training even for discontinuous data distributions. Thanh-Tung et al. [29] proposed a zero-centered gradient penalty method that along with local convergence improves generalization and prevents gradient explosion in the discriminator. Gradient explosion in a discriminator leads to mode collapse in a generator. Mode collapse can be described as a process where the generator is not sufficiently sensitive to its input information [32, 33, 34]. Therefore, the GAN methodology presented in this paper was trained using the following objective function [29]:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{\boldsymbol{\xi} \in P_\xi} \left[\log \mathcal{D}(\boldsymbol{\xi})\right] + \mathbb{E}_{\boldsymbol{c} \in P_c} \left[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{c})))\right] - \lambda \mathbb{E}_{\widetilde{\boldsymbol{\xi}}} \left[\|(\nabla \mathcal{D})_{\widetilde{\boldsymbol{\xi}}}\|_2^2\right] , \tag{6}$$

where $\widetilde{\boldsymbol{\xi}} = \alpha \boldsymbol{\xi} + (1 - \alpha)\boldsymbol{\eta}$ and $\alpha \sim \mathcal{U}(0, 1)$, which represents that α is a random number from a uniform distribution on the interval $[0, 1)$. In equation (6) $\| \cdot \|_2$ stands for $L2$-norm, $(\nabla \mathcal{D})_{\widetilde{\boldsymbol{\xi}}}$ is the gradient of $\mathcal{D}(\widetilde{\boldsymbol{\xi}})$ with respect to $\widetilde{\boldsymbol{\xi}}$ and parameter λ is a measure of discriminative power of $\mathcal{D}$. A λ value of 0 makes the $\mathcal{D}$ only focus on maximizing its discriminative power. $\mathcal{D}$ has maximum generalization capability and no discriminative power if λ approaches infinity. It can be observed that the original GAN objective function (5) can be obtained for $\lambda = 0$. Details of the loss functions for generator and discriminator, which are based on (6), are elaborated in Appendix A.

3.1. GAN Architecture

The generator and the discriminator used in the current work are built using 3D convolutional layers, whose schematic is shown in Figure 5. As shown in the figure, the generator comprises of 14 convolutional layers and the discriminator contains 5 convolutional layers which are followed by a single linear layer. The architectural details of these neural networks have been detailed in Tables A.5 & A.6 respectively in Appendix A.

The input to the generator is the 3D *Indicator Function* (I_f)

$$I_f(x, y, z) = \begin{cases} 0, & if\ (x, y, z)\ is\ inside\ a\ particle. \\ 1, & if\ (x, y, z)\ is\ outside\ a\ particle. \end{cases} \tag{7}$$

which clearly demarcates the regions of the sub-domain occupied by the reference particle (which is located at the center of the sub-domain) and its neighbors. Note that with the above definition of indicator function, portions of neighboring particles that happen to lie within the sub-domain are taken into account, even when that neighbor's center happen to fall outside the sub-domain. In addition, the input information includes the Reynolds number of the flow. The local volume fraction information has already been provided in terms of the indicator function. The indicator function and the Reynolds number information can be combined together by redefining the indicator function to equal the Reynolds number at every point within the fluid

region. The redefined indicator function is then the condition $\boldsymbol{c}$ that serves as input to the generator. The output of the generator are the velocity and pressure fields within the sub-domain in regions occupied by the fluid.

The input to the discriminator is both the redefined indicator function $\boldsymbol{c}$ and also the three components of velocity and pressure within the sub-domain in the region outside the particles. As shown in Figure 5, this 3D velocity and pressure inputs to $\mathcal{D}$ can be either in the form of true data $\boldsymbol{\xi}$ or synthetic data $\boldsymbol{\eta}$ outputted by the generator.

The 3D ML inputs and output fields described above have to be discretized and prepared in terms of 3D voxels. The computer memory and computational time required for training and testing of the GAN scale as the number of voxels. A finely discretized input and output fields may be desirable for accurate representation of the flow, but available memory on a GPU, where the GAN is implemented, places important restriction on the number of voxels. In this work, we limit the number of equi-spaced grid points along each direction to be 64, and thus, $(64)^3$ voxels are used to represent the indicator function and the velocity and pressure fields within the sub-domain. Since the sub-domain is of size $8 \times 5 \times 5$, the resolution is slightly lower along the streamwise direction than along the transverse direction. Clearly the resolution of the sub-domain with $(64)^3$ voxles is somewhat lower than the resolution of the DNS. However, the purpose of predicting the flow within the sub-domain is to only obtain a coarse-grained solution around the reference particle taking into account a somewhat larger neighborhood of particles. As we will see below, once the coarse-grained solution is obtained within the sub-domain, a much finer solution will be obtained in the immediate neighborhood of the reference particle through attention mechanism.

3.2. Attention Mechanism CNN

The attention-domain of a sample corresponds to the innermost $16 \times 26 \times 26$ voxels of the generator's output. Increasing the resolution to $(64)^3$ within the attention-domain will provide high enough resolution that is comparable to DNS. This can be achieved by a simple interpolation from the coarse grid to a finer grid. However, such an interpolation will not improve upon the accuracy of the GAN solution that was obtained on the coarse grid.

Here a more refined solution in the attention-domain is obtained using a simple three layered convolutional neural network $\mathcal{A}$ (schematically shown in figure 5 and architectural details are given by A.7), whose goal is to improve the accuracy of the solution within the attention-domain. As shown in figure 5, the input to $\mathcal{A}$ is the generator's output $\boldsymbol{\eta}$ for an input condition $\boldsymbol{c}$. The objective of $\mathcal{A}$ is to make its output $\boldsymbol{\zeta}$ as close as possible to $\boldsymbol{\psi}$, which is the corresponding actual DNS solution on $(64)^3$ grid points in that attention-domain. The synthetic flow prediction $\boldsymbol{\eta}$ of GAN is first interpolated onto a $(16)^3$ voxel spanning an inner cube of side 2 centered around the reference particle. This information is passed as input to $\mathcal{A}$ and the expected output is u', v', w', p' fields in the $2 \times 2 \times 2$ attention-domain around the reference particle discretized by $(64)^3$ uniformly distributed voxels. The functioning of $\mathcal{A}$ can be perceived as super-resolution in the inner $2 \times 2 \times 2$ attention-domain.

Neural network training approach adopted for this model is that all the three networks, namely Generator, Discriminator and Attention Mechanism CNN, are trained together. This mode of training indicates that $\boldsymbol{\eta}$ which is passed as input to $\mathcal{A}$ is varying throughout the entire training process. It has to be mentioned that this is not the only approach that can be used in implementing the attention mechanism. For example, training of $\mathcal{A}$ can be started after the completion of GAN training. This will ensure that $\mathcal{A}$ is always trained using the optimum version of $\boldsymbol{\eta}$. But this may not guarantee optimal value of $\boldsymbol{\zeta}$, which is the final desired output from the attention mechanism. Here we train all three CNNs together in order to obtain an optimal $\boldsymbol{\zeta}$. Similar to the existence of different training approaches, there also exists a choice in network architecture for the attention mechanism. A simple three layer CNN was used in this work, however, a separate GAN architecture can also be implemented for this purpose [21, 35].

It should be pointed out that the above described two step process of first predicting the coarse grained flow in the sub-domain using a GAN followed by the second step of improving the accuracy with a fine-grained solution in the attention-domain with a CNN is essential. It is not possible to directly go to the fine-grained solution in the attention-domain, since the smaller domain will not contain information on many of the neighbors of the reference particle. As a summary, the input and output of the three networks $\mathcal{G}$, $\mathcal{D}$ and $\mathcal{A}$ have been concisely presented in Table 2.

Network	Input	Output
$\mathcal{G}$	$\boldsymbol{c}$	$\boldsymbol{\eta}$ = sub-domain (u', v', w', p')
$\mathcal{D}$	$\boldsymbol{c} \cup \{\boldsymbol{\xi}$ or $\boldsymbol{\eta}\}$	q = Prediction value $(\in [0, 1])$
$\mathcal{A}$	$\boldsymbol{\eta}$	$\boldsymbol{\zeta}$ = attention-domain (u', v', w', p')

Table 2: Inputs and outputs of neural networks used in this work

3.3. Data Augmentation With Symmetry

Since the mean flow of each sub-domain is along the x-direction, this is the only preferred direction of the problem. We desire the GAN networks $\mathcal{G}$, $\mathcal{D}$, and $\mathcal{A}$ to satisfy these symmetries. In other words, given a condition $\boldsymbol{c}$ as input if the generator $\mathcal{G}$ outputs the synthetic field $\boldsymbol{\eta}$, then if a rotated (or reflected) condition $\boldsymbol{c}'$ were to be provided then the generator $\mathcal{G}$ must output a synthetic field $\boldsymbol{\eta}'$ that is a correspondingly rotated (or reflected) copy of $\boldsymbol{\eta}$. An unbounded or a cylindrical domain presents axisymmetry and allows continuous rotation about the x-axis. However, the present cuboid domain allows only discrete rotations of 90°, 180°, 270° and reflections about the y and z axis.

The symmetry requirement of the GAN networks $\mathcal{G}$, $\mathcal{D}$, and $\mathcal{A}$ is weakly enforced by data augmentation. For every training data represented by the redefined indicator function $\boldsymbol{c}$ and the associated velocity/pressure fields which form the corresponding true data $\boldsymbol{\xi}$, eight equivalent conditions and associated flow fields can be generated by applying the discrete rotations and reflections. A schematic of this data augmentation is shown in figure 6. Only one $y - z$ plane of the sub-domain is shown where the intersection of the plane with the particles appears as circles of varying radius, the color contours show local value of streamwise velocity, and the arrows in the plane correspond to the in-plane velocity vector at that location. In the figure, frame (a) corresponds to the original data as obtained from PR-DNS and frames (b)-(d) are discrete rotations with an increment of 90° about x-axis. Frames (e)-(h) are reflections of (a)-(d) about the z coordinate. As expected a rotation/reflection leads to a corresponding rotation/reflection of independent scalar quantities like locations of particles, streamwise velocity, and pressure. On the other hand, rotation/reflection causes not only a change in the location of in-plane velocity vector but also a change in its direction. This change in quantities due to rotation/reflection can be observed in figure 6. Thus, the imposition of symmetry results in eight fold increase in the training and testing data.

4. Results and Discussion

The neural networks were trained separately for every case mentioned in Table 1. In each case, segregation of the entire dataset into *training data* and *testing data* was achieved by leaving one realization as testing data and all other realizations are used as training data. This type of data segregation ensures that there is no overlap between training and testing data, thereby, leading to a genuine network performance evaluation on test dataset. However, it is not clear a-priori which realization should serve as the testing data with others serving as training data.

It should be also mentioned that in a given case (specified by Reynolds number and particle volume fraction) there does not exist a small and practical set of parameters that can be used to uniquely classify the arrangement of neighboring particles around the reference particle. This suggests that the level of similarity between testing and training datasets cannot be determined. Hence, to properly establish the uncertainty in the performance of the GAN networks we must quantify error by selecting every single realization as *testing realization* one at a time. I.e., in the example of case 1, where there are 10 realizations (see Table 1), training, testing and error analysis will be repeated 10 times, and in each repetition a different realization will be the testing data, while all others form the training data. The overall error to be reported is an average of these. In each of the Reynolds number, volume fraction cases considered, the number of training data N_{tr} and the number of testing data N_{te} are listed in Table 1.

Furthermore, it is very well known that generalization capabilities of a neural network increase with the amount of new training data given to it. However, the number of training samples required to achieve

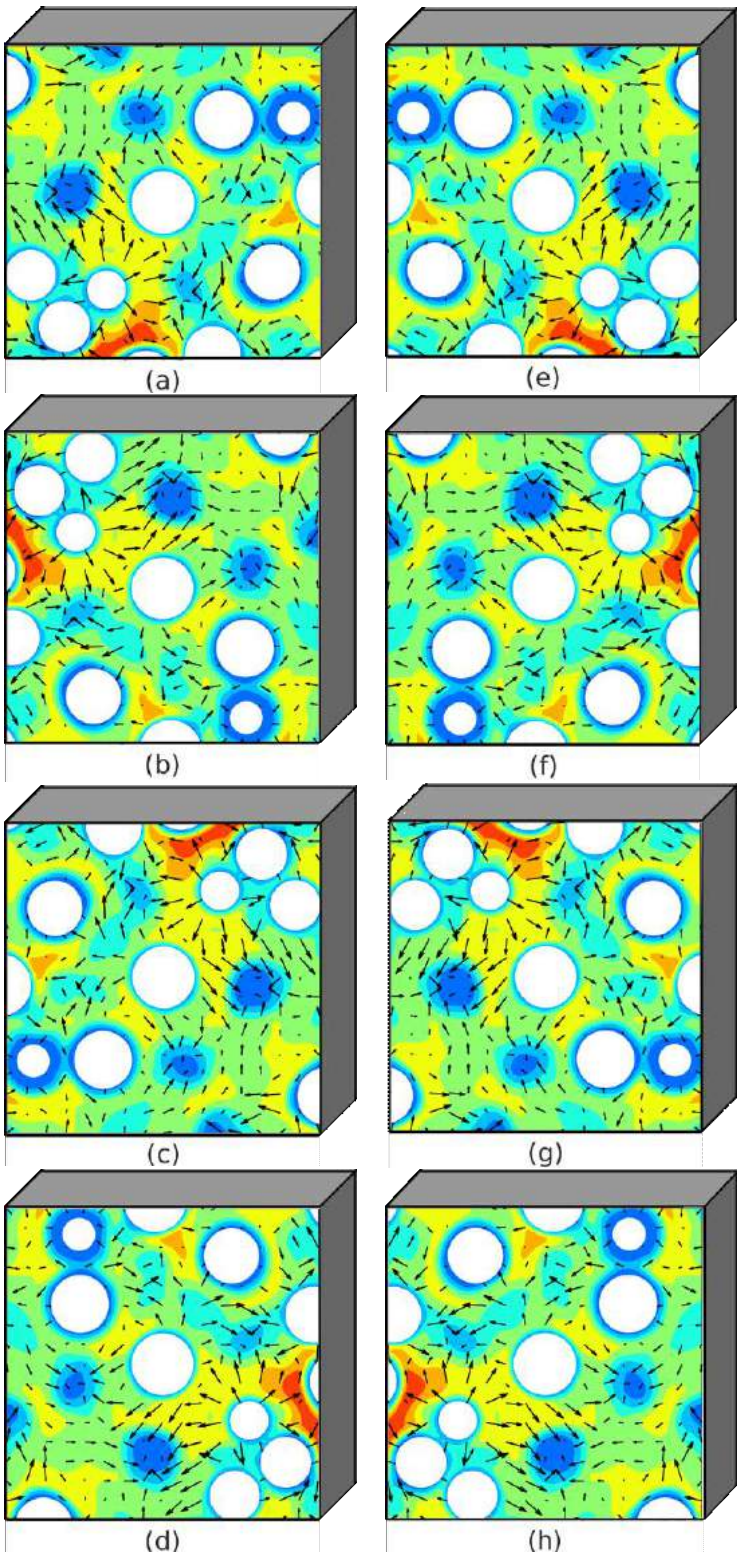


Figure 6: Data Augmentation using rotations and corresponding reflections shown for a y-z plane of a sample from case $Re = 172.96$, $\phi = 0.11$.

desired performance on test data is problem-specific. To evaluate the rate of increase of test performance with increase in training samples the networks are trained on different-sized subsets of the training dataset. This will help in getting an estimate of the required number of training samples to achieve a desired level of test performance. Therefore, the number of training data used for training the GAN and attention-CNN was varied from $N_{sub,tr} = N_{tr}/6$ to N_{tr}. These subsets are chosen randomly from the total training dataset. Entire test dataset is used in evaluating test performance of these networks that are trained on different sized subsets to ensure a valid and consistent measure of increase in generalization with increase in training data.

4.1. Sample Display

To provide a visual image of flow field generated by the GAN model, sample results from case 3 and case 8 are presented here in Figures 7, 8 and 9. These samples have been randomly chosen from the model's testing dataset when the model is trained on the entire training dataset, i.e., $N_{sub,tr} = N_{tr}$. GAN architecture output $\boldsymbol{\eta}$ is of particular interest as it produces flow field information in the sub-domain by taking location of particles and volume-averaged Reynolds number as input. For appropriate visualization and compact presentation of the output, velocity and pressure have been transformed back to their original scales, i.e., u', v', w', and p' are transformed to u, v, w and p. Central y-z plane of the sub-domain for both samples have been shown in the figures. In Figure 7 contours of true PR-DNS streamwise velocity and synthetic streamwise velocity from GAN are shown (streamwise velocity is the normal velocity on this plane) along with contours of the difference as a measure of error. Figure 8 shows plots of in-plane velocity vector from PR-DNS and GAN along with the error and Figure 9 shows the corresponding pressure contours. Videos of several y-z planes spanning over the streamwise length of sub-domain for these two samples can be seen in supplementary data. It can be seen from the figures and the corresponding videos that the GAN is able to capture the velocity and pressure fields reasonably accurately. In particular, it can be noted that the errors are quite low in the region immediately surrounding the central reference particle. The higher errors mostly occur closer to the boundaries of the sub-domain. This is to be expected, since for the central reference particle, its entire neighborhood is contained within the sub-domain so that the local flow is well predicted by GAN. In contrast, as we approach the boundaries of the sub-domain, the arrangement of particles that are outside the sub-domain are unknown as they are not contained in the condition $\boldsymbol{c}$. So as a result GAN prediction goes down in accuracy as we move away from the reference particle towards the sub-domain boundaries. Fortunately, in the first place, the idea of sub-domain was to get accurate flow field around the reference particle, which is being achieved to a very good extent. This will be quantified below in succeeding results.

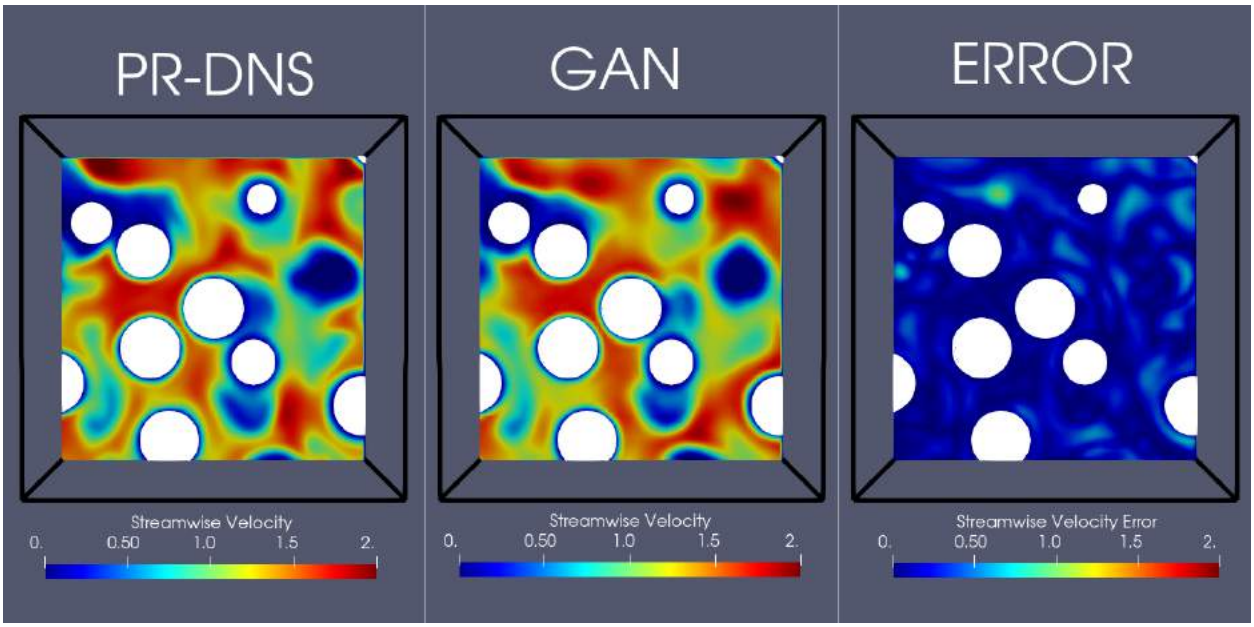
Flow information (u, v, w, p) for all grid points that fall inside of particles have been zeroed out in every PR-DNS sample. The generator which is trained using the loss function given by (A.1) takes only points outside of particles into consideration through $L1$-norm. Thus, flow variable data generated by $\mathcal{G}$ for grid points inside of particles is not optimized through $L1$-norm but may be learned from adversarial learning against $\mathcal{D}$. From Figures 7, 8, 9 and corresponding videos, it can be seen that the model fairly predicts velocities and pressure outside of particles for both cases.

4.2. Performance Evaluation Metric

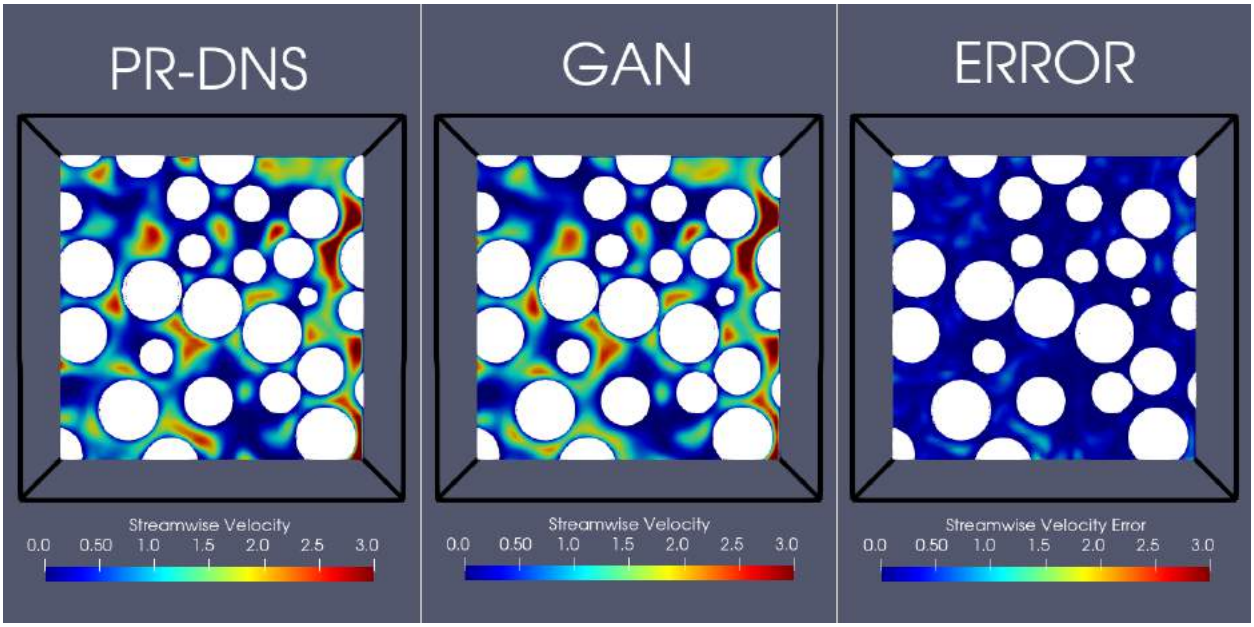
Coefficient of determination, R^2, is used to evaluate test performance of neural networks $\mathcal{G}$ and $\mathcal{A}$. R^2 is defined as

$$R^2_{u',\,box} = 1 - \frac{\sum_{i=1}^{N_{te}} \sum_{box} (u'_{DNS} - u'_{network})^2 I_f}{\sum_{i=1}^{N_{te}} \sum_{box} (u'_{DNS})^2 I_f}, \tag{8}$$

where, N_{te} is the number of test samples. Here the pair (box, $network$) corresponds to either ($sub-domain$, $\mathcal{G}$) or ($attention-domain$, $\mathcal{A}$). Therefore, if the box corresponds to sub-domain, then u'_{DNS} represents normalized perturbation streamwise velocity (u') obtained from PR-DNS within the $8 \times 5 \times 5$ sub-domain discretized on $(64)^3$ voxels. $u'_{network}$ then corresponds to the synthetic solution of the network $\mathcal{G}$. Similarly, if box is attention-domain, then u'_{DNS} is DNS solution within the $2 \times 2 \times 2$ region discretized on $(64)^3$ voxels and $u'_{network}$ is the predicted solution of the attention network $\mathcal{A}$. An $R^2_{u',\,box}$ value of one

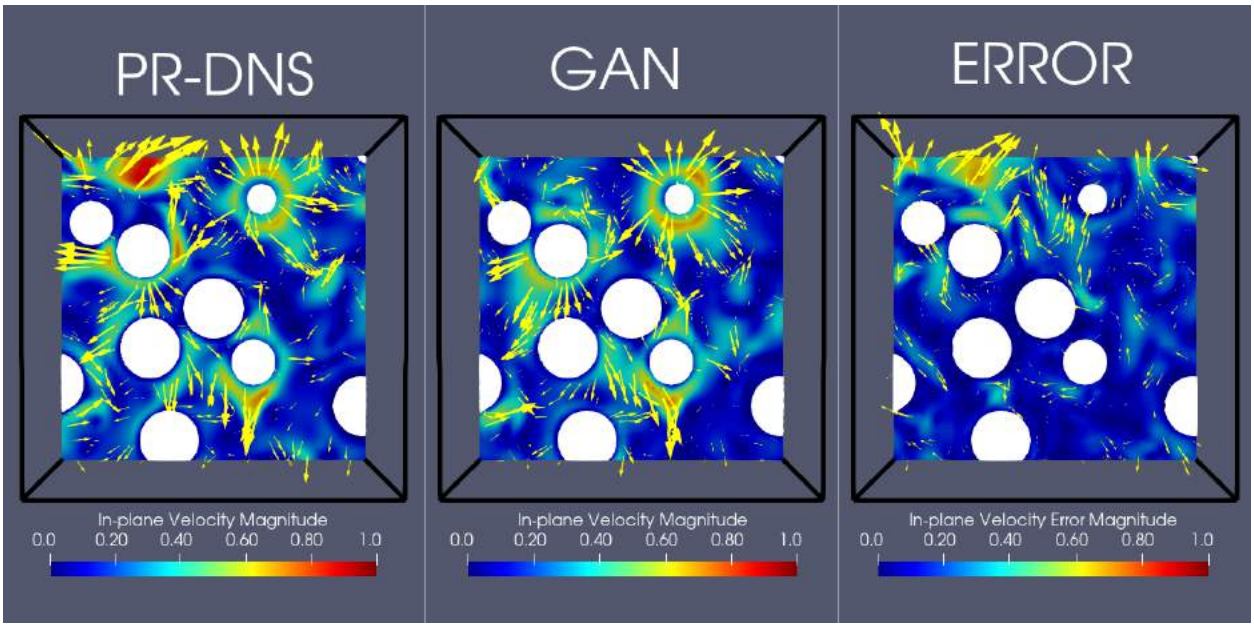


(a)

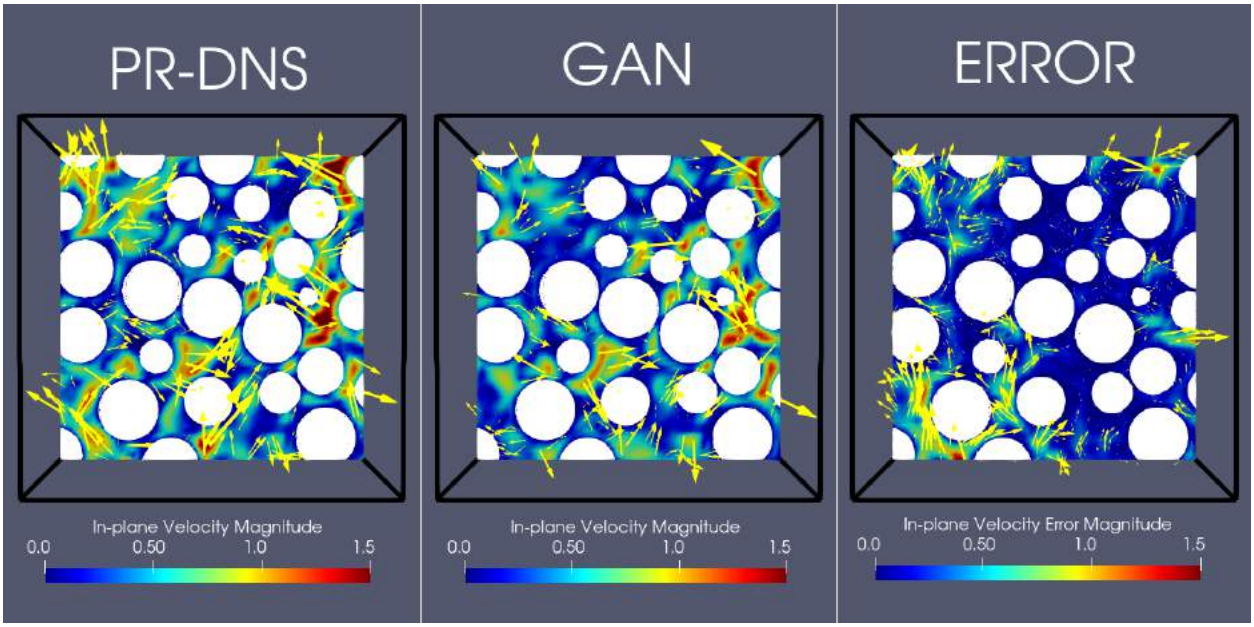


(b)

Figure 7: Comparison of GAN model output with PR-DNS using streamwise velocity for a sample from (a) Case 3 and (b) Case 8.

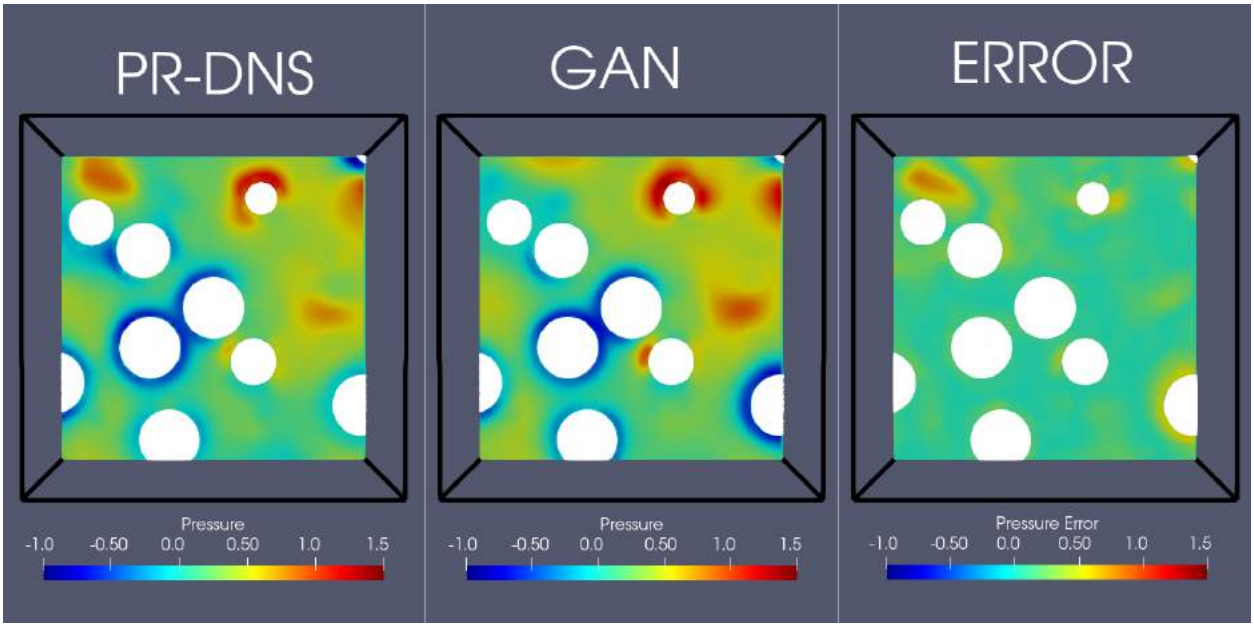


(a)

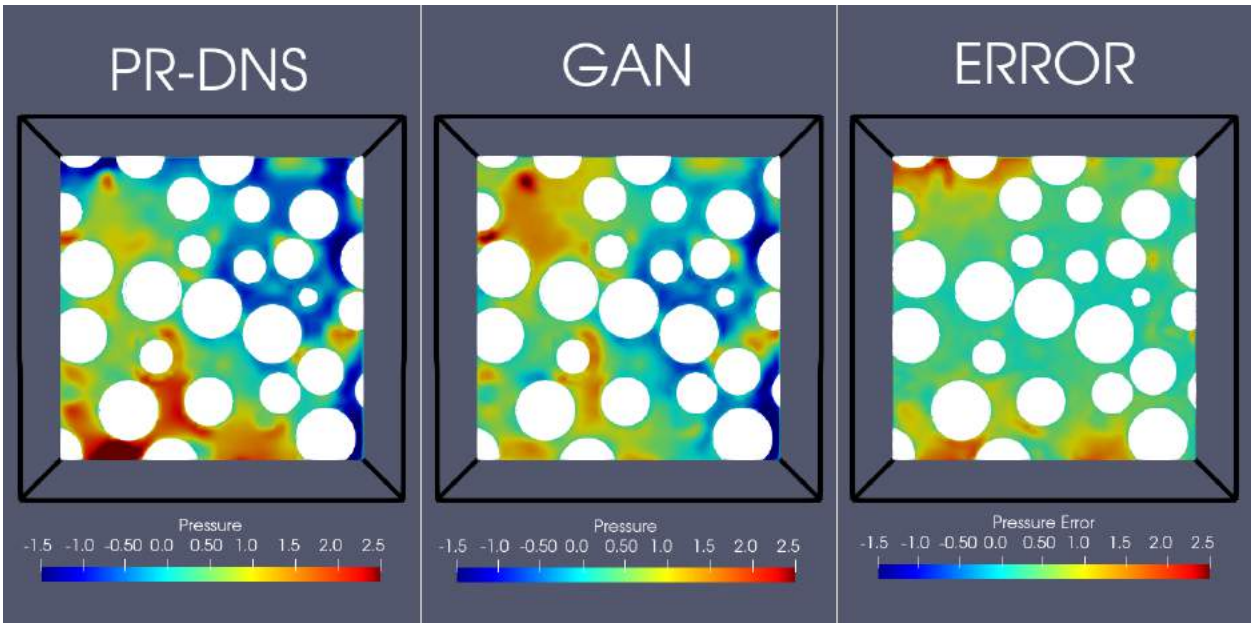


(b)

Figure 8: Comparison of GAN model output with PR-DNS using in-plane velocity for a sample from (a) Case 3 and (b) Case 8.



(a)



(b)

Figure 9: Comparison of GAN model output with PR-DNS using pressure for a sample from (a) Case 3 and (b) Case 8.

is equivalent to a perfect model that exactly predicts the DNS solution. Hence, the performance of GAN and the Attention-CNN can be measured by how close their $R^2_{u',box}$ value is to unity. Similar performance metrics are defined for network's transverse velocities and pressure components, which are given by $R^2_{v',box}$ and $R^2_{p',box}$ respectively.

4.3. Generator Performance

As explained earlier in this section, performance within a sub-domain is essentially evaluation of generator's capability to predict normalized perturbations inside the sub-domain. Figure 10 depicts the average performance of generator when trained on different subset sizes. For a given particle volume fraction the performance of velocity prediction with the GAN model decreases with increasing Reynolds number. The performance of transverse velocity components is lower than corresponding streamwise component in all of the cases. This can be explained using the reason that axisymmetric nature of the problem about streamwise direction has only been weakly built into the model through inclusion of discrete rotations and reflections. As previously discussed in Figure 6, rotation (or reflection) leads to a scalar rotation (or reflection) for streamwise velocity, however, it leads to a vector rotation (or reflection) for transverse velocity suggesting that the latter task is more involved compared to the former. Furthermore, the highest and least performances of streamwise component are associated with the smallest and largest Reynolds number cases respectively. There is a larger difference in the performance of velocity components and pressure for $\phi = 0.45$ cases compared to lower volume fraction cases.

It can also be observed from Figure 10 that the training set is sufficient in all cases to obtain reasonably converged GAN model. The rate of increase in performance with increase in $N_{sub,tr}$ is different for every case. A noticeable increase in a model's performance with increase of training dataset size occurs only in cases that have low performance value when trained on smaller sized datasets. It can also be seen that the performance of a model when $N_{sub,tr} = N_{tr}/6$ in some cases is higher than that when a model is trained with $N_{sub,tr} = N_{tr}$ in other cases. This indicates that the generalization capabilities of the presented architecture vary with the considered case. Hence, it requires minor optimization of the GAN model in terms of architecture, training dataset size, loss terms and training approach in order to achieve a desired level of test performance for a particular case.

4.4. Attention-CNN Performance

Unlike the task of a generator which has to produce flow data from a condition, $\mathcal{A}$ has to perform super-resolution of flow data variables. This task can be also perceived as performing an abstract non-linear interpolation from the coarse-grain output of $\mathcal{G}$ to the finely-resolved spatial points. Thus, the performance of $\mathcal{A}$ is compared with that of a linear interpolation performed on generator's output. The performance of interpolation is equivalent to the generator's performance in the attention-domain. Average performance of $\mathcal{A}$ and its comparison with linear interpolation have been shown in figure 11. First and foremost, the performance of attention-cnn in all cases is higher compared to their respective generator's performance, with the improvement being significant in the higher volume fraction cases and modest in the other cases.

It can be seen from the figure that performance of a simple three layered $\mathcal{A}$ for flow fields is similar or slightly lower to that of linear interpolation. This suggests that $\mathcal{A}$ is not necessarily improving upon the generator's output in the attention-domain. However, these results do not rule out the option of using an attention mechanism but only indicate that this architecture and methodology needs improvements like passing in an indicator function also as an input along with coarse-grain output. Furthermore, it should also be taken into consideration that performing linear interpolation using only the fluid grid points is not computationally straightforward.

4.5. Networks Performance Uncertainty

Here we discuss uncertainty in the performance of $\mathcal{G}$ and $\mathcal{A}$. We evaluate uncertainty in terms of how much the performance varies when each realization of a case is considered as the test realization. For example, in case 1 there are ten realizations and R^2 value can be computed as given in (8) using only the i^{th} realization as the test realization and all other nine realizations to be used in the training process. As the

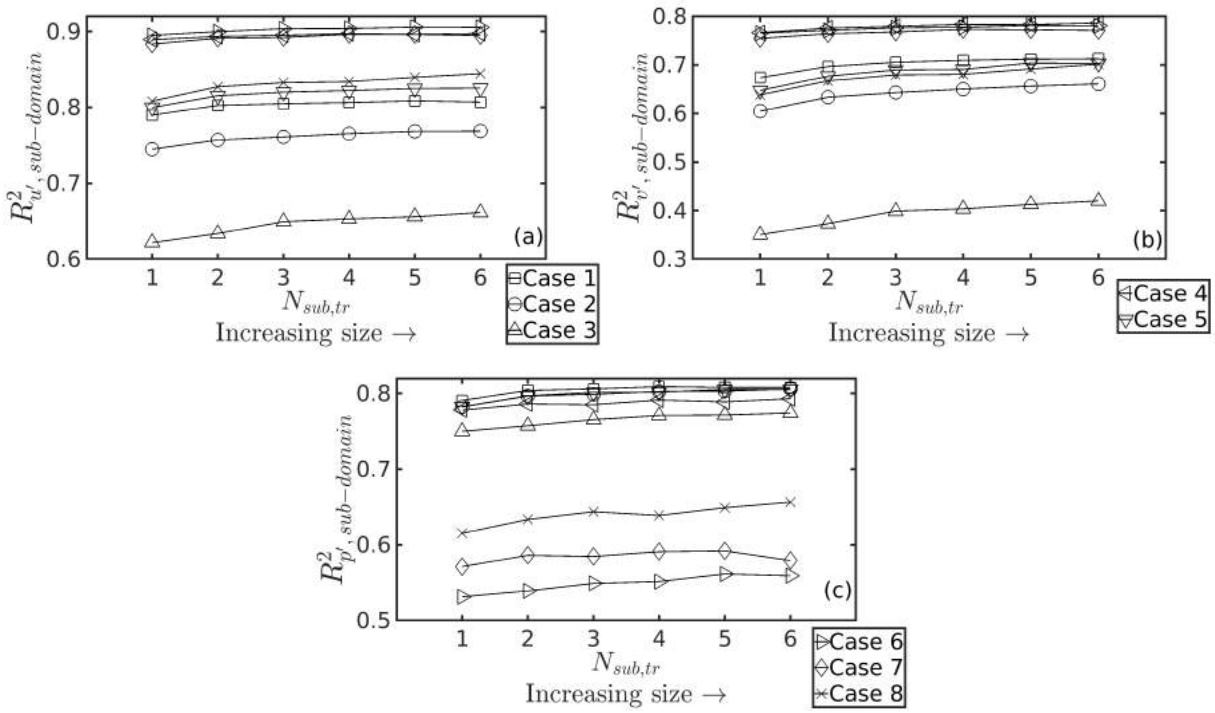


Figure 10: Performance evaluation of Generator ($\mathcal{G}$) for (a) Streamwise Velocity, (b) Transverse Velocities, and (c) Pressure

test realization i is varied from 1 to $M_c = 10$, the computed R^2 value will vary and this variation is denoted as performance variability (or uncertainty) and is defined as

$$\|R_i^2 - R^2\|_\infty \,, \tag{9}$$

where R_i^2 is the performance evaluated using only the i^{th} realization as the test realization, while R^2 is the corresponding average performance, averaged over $i = 1, \cdots M_c$ and $\| \cdot \|_\infty$ is L-infinity-norm. Since R^2 is already normalized further normalization of its variability is not necessary. A low value of performance variability would indicate that the GAN model is likely to yield close to average performance for any new random condition $\boldsymbol{c}$. The performance variability of $\mathcal{G}$ and $\mathcal{A}$ for all cases are shown in figures 12 & 13 respectively. Performance variability does not seem to depend strongly on the size of the training dataset. The largest pressure variability in $\mathcal{G}$ and $\mathcal{A}$ among all of the different test runs for the presented cases is approximately twice the largest variability for velocity components.

In general, low variation in the performance of $\mathcal{G}$ and $\mathcal{A}$, especially for velocity components, is certainly an indication that the realizations of a case are not completely unique. This low variability of the networks is an optimistic sign for their performance on unknown datasets.

4.6. Flow Field Reconstruction

The GAN methodology has been designed to predict the flow field in a sub-domain around the reference particle. Here we present a simple approach that uses the sub-domain results to predict the flow field over the entire computational domain. The entire domain around the particles is divided into voronoi cells [36] to achieve this particular task. Flow field at all the grid points that are inside a particle's voronoi volume is taken to be the GAN output of the sub-domain with that particle as the reference at the center. The entire flow field is created by assembling these individual solutions within all the voronoi volumes together. It was observed in earlier sections that the generator's predictions are more accurate closer to the reference particle. Hence, it is expected that the performance of the reconstructed flow field in the entire domain is even better than $\mathcal{G}$'s performance (see Figure 10) and closer to the performance of linear interpolation shown in Figure 11.

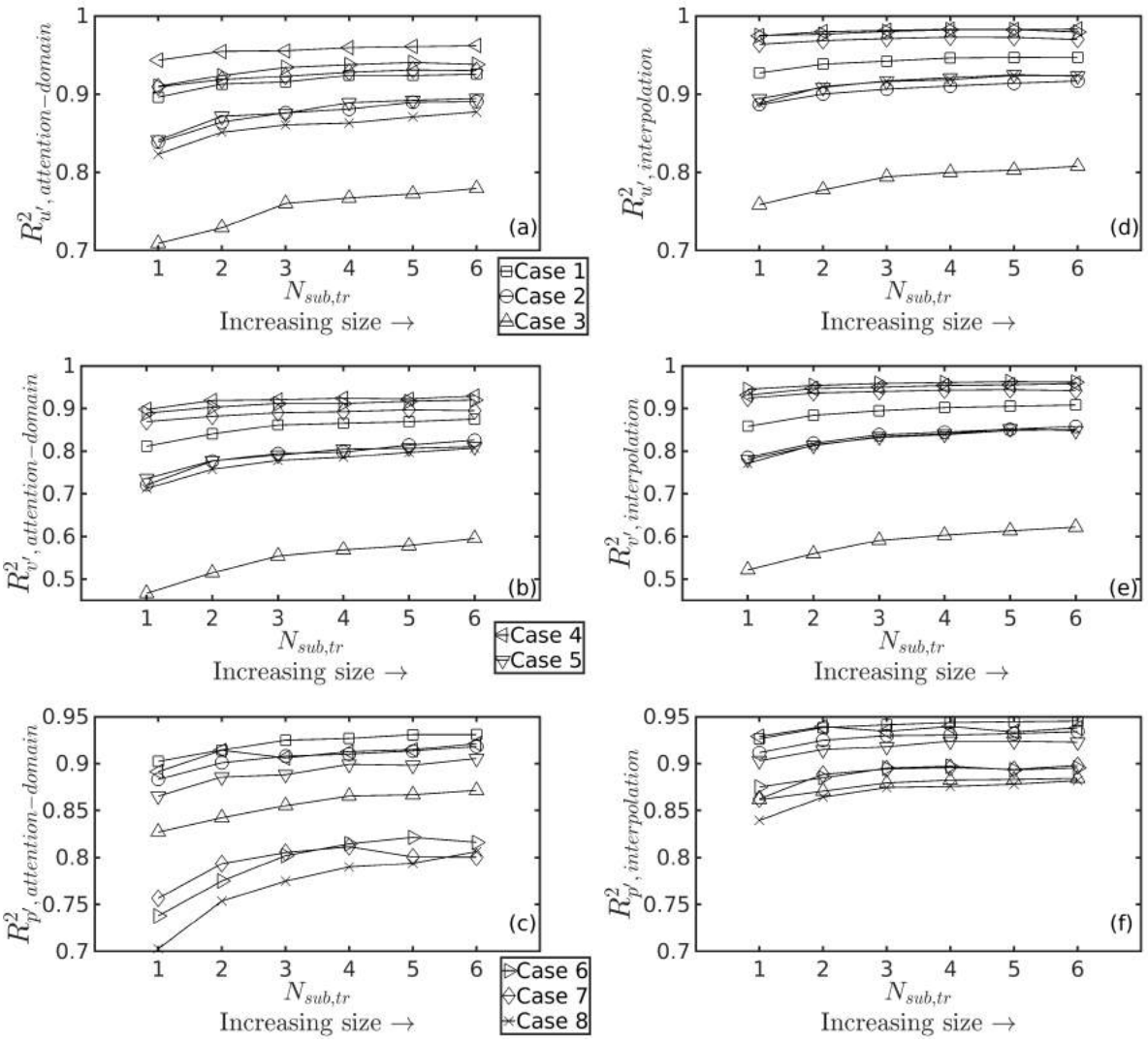


Figure 11: Performance of $\mathcal{A}$ for (a) Streamwise Velocity, (b) Transverse Velocities, (c) Pressure, and Linear interpolation for (d) Streamwise Velocity, (e) Transverse Velocities, (f) Pressure

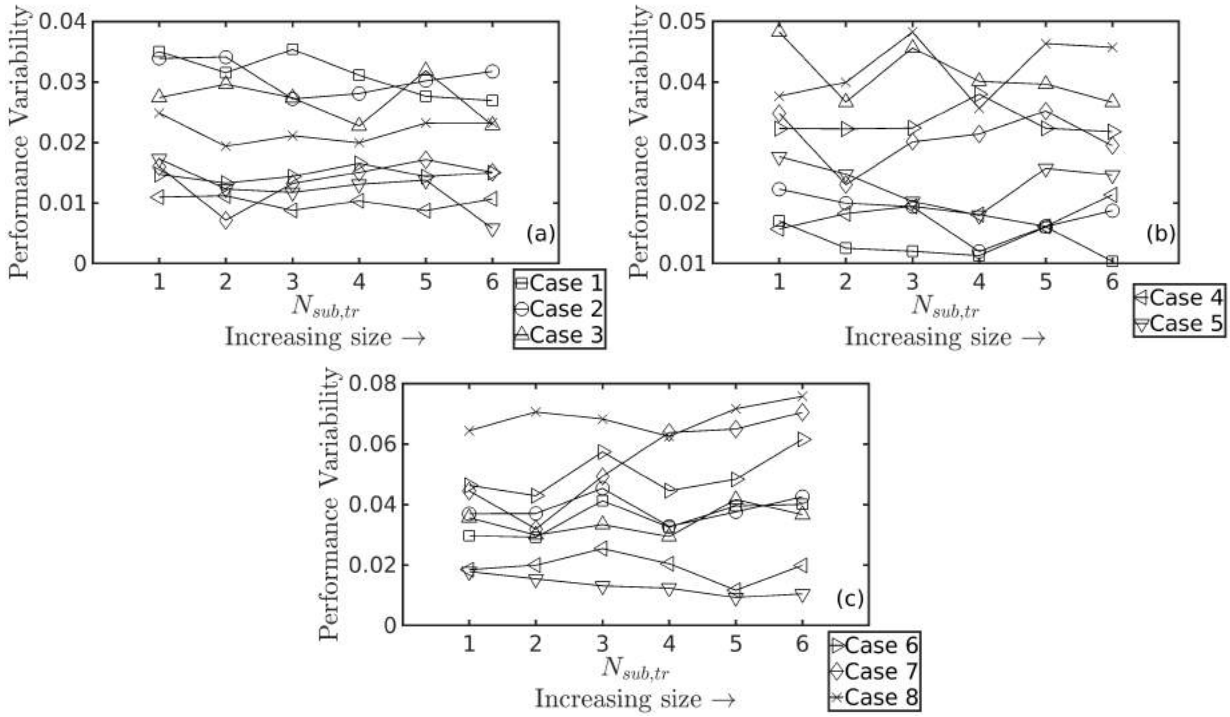


Figure 12: Performance variability of Generator ($\mathcal{G}$) for (a) Streamwise Velocity, (b) Transverse Velocities, and (c) Pressure

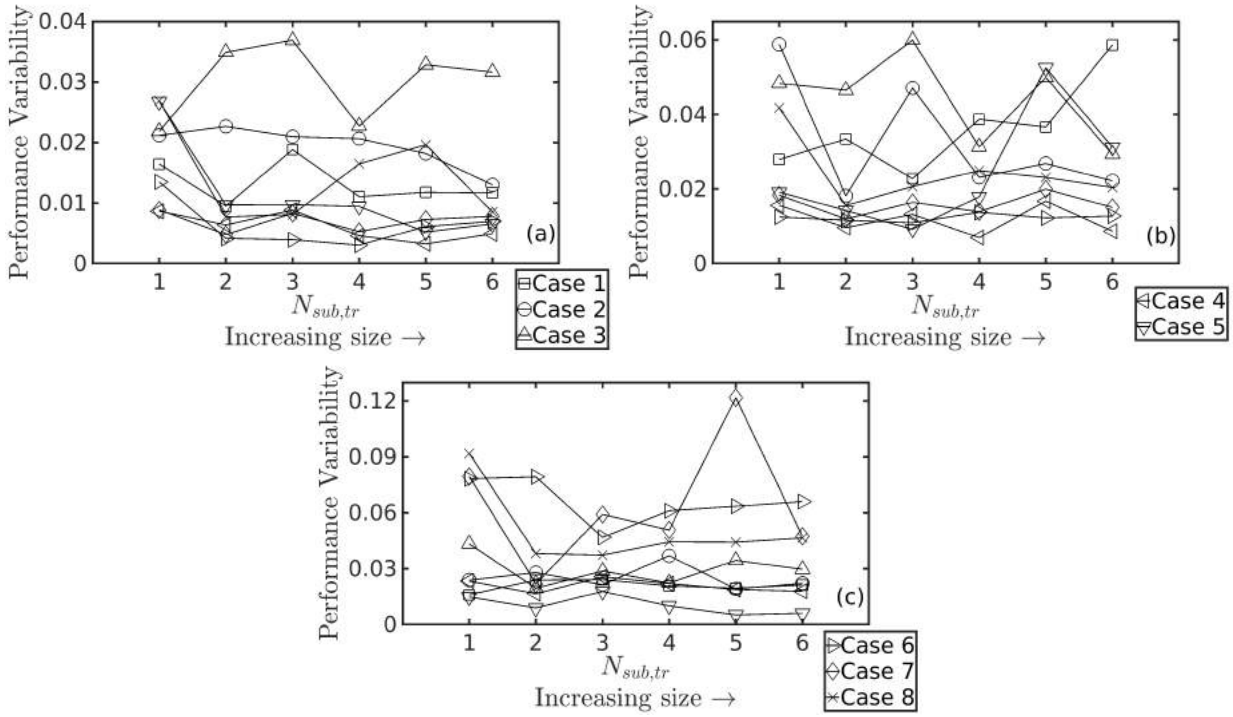
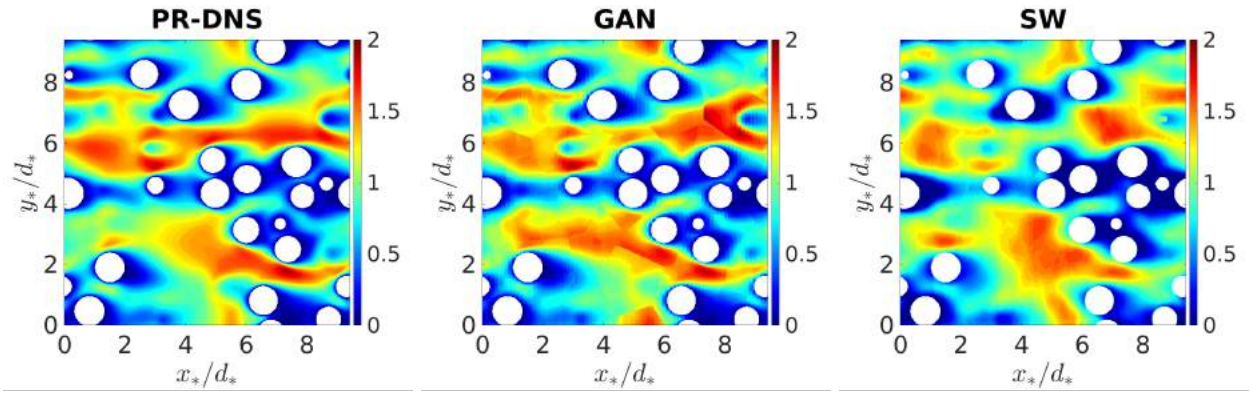
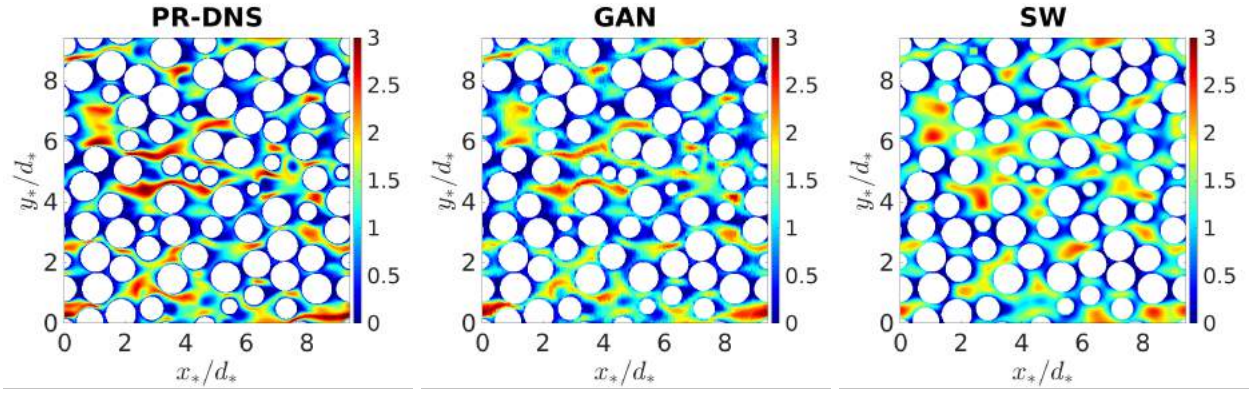


Figure 13: Performance variability of Attention-CNN ($\mathcal{A}$) for (a) Streamwise Velocity, (b) Transverse Velocities, and (c) Pressure



(a)



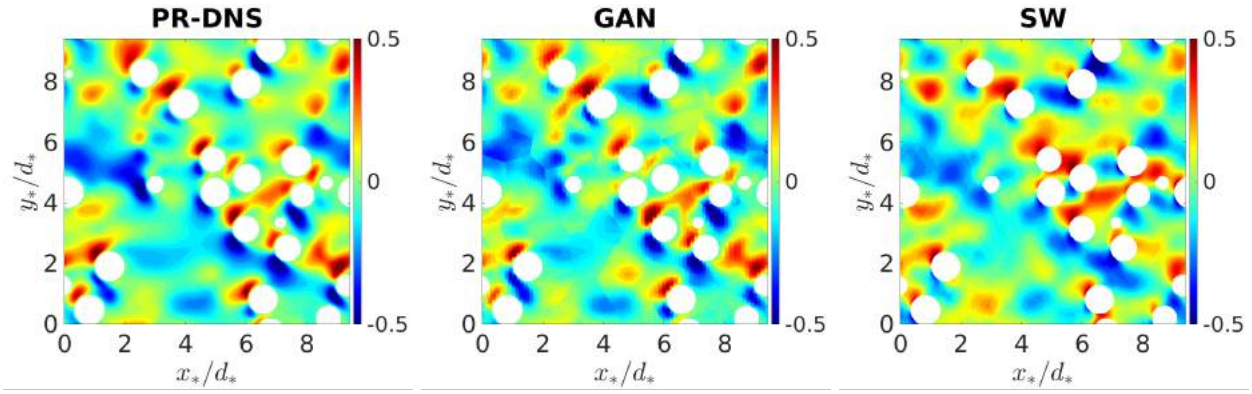
(b)

Figure 14: Streamwise velocity for central $x-y$ plane of a test realization from (a) Case 1 and (b) Case 8

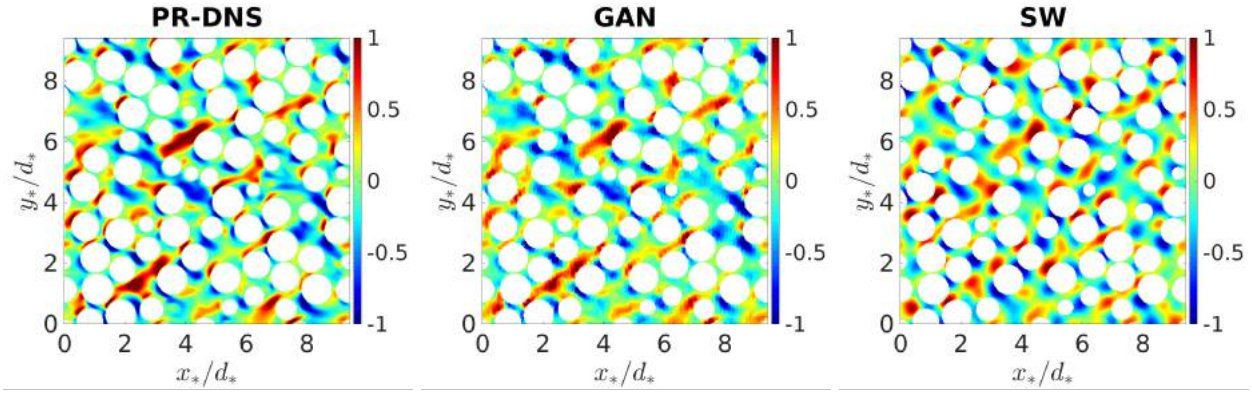
	GAN			SW		
	u	v	p	u	v	p
Case 1	0.9214	0.8609	0.9262	0.7184	0.6173	0.7275
Case 8	0.8416	0.8230	0.9043	0.5635	0.5458	0.5965

Table 3: Test performance of the models for the central $x-y$ plane of samples from cases 1 and 8.

For illustration purposes the $3\pi \times 3\pi \times 3\pi$ cubic domain's central $x-y$ plane has been presented for a test realization each from Cases 1 and 8. Contours of streamwise velocity, in-plane cross-stream velocity and pressure are presented in Figures 14, 15, & 16. The generator used to produce these results was trained using all of the training data, i.e., $N_{sub,tr} = N_{tr}$. In these figures the reconstructed flow field using the GAN methodology is compared with the corresponding PR-DNS solution and also with the flow field predicted using the superposable wake (SW) model [28]. It can be seen from the figures that the reconstructed flow field using GAN methodology is much closer to the PR-DNS solution compared to superposable wake model in both the low and high volume fraction samples. Test performance of the two models for the central $x-y$ slice in both samples have been tabulated in Table 3. The R^2 metric presented in the table for u, v, p is the same as that followed in Moore and Balachandar [28] and it measures the correspondence between the model and the PR-DNS. The superior performance of the GAN prediction over the superposable wake model is clear. In Moore and Balachandar [28] it was pointed out that the superposable wake yields the optimal solution within the pairwise interaction approximation. Thus, the improved performance of the GAN model is due to its ability to account for the N-body interaction among the particles. As a result, the improvement over superposable wake is higher in the higher volume fraction case.

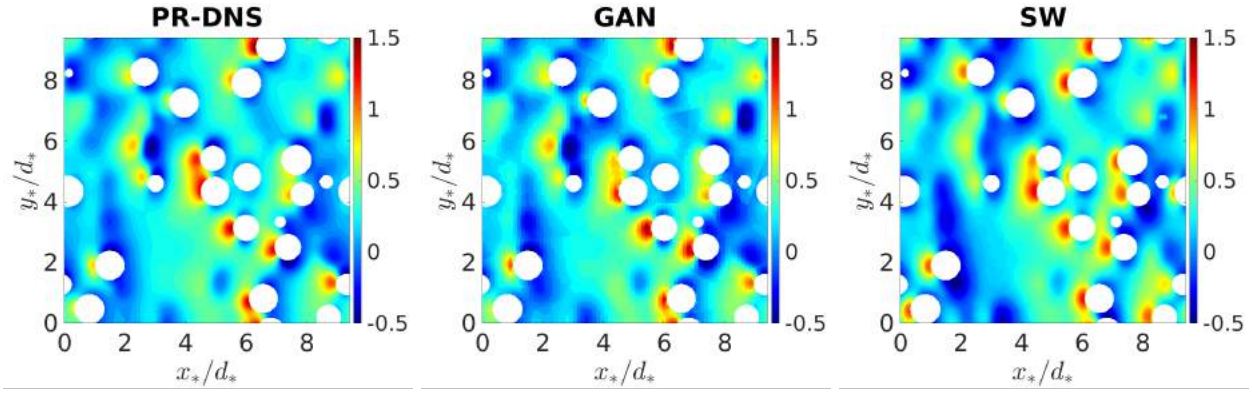


(a)

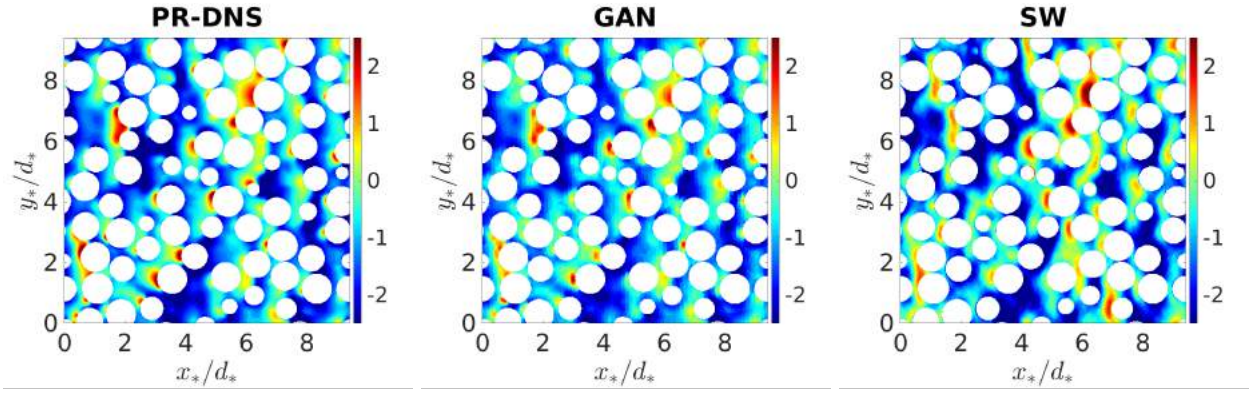


(b)

Figure 15: In-plane transverse velocity component (v) for central $x-y$ plane of a test realization from (a) Case 1 and (b) Case 8



(a)



(b)

Figure 16: Pressure for central $x - y$ plane of a test realization from (a) Case 1 and (b) Case 8

5. Conclusion and Future Scope

The article describes a methodology to predict flow field in a dispersed multiphase setup that involves randomly distributed monodispersive spheres at different combinations of particle volume fraction and volume-averaged Reynolds number. This methodology is based on a data-driven technique called Generative Adversarial Network (GAN). The networks used in this model essentially comprise of Convolutional Neural Networks. For accurately predicting fluid flow around each particle, the model considers a neighborhood around the particle, termed as the particle's sub-domain. This approach thus requires curation of the Direct Numerical Simulation data, which is required in the first place for training the GAN model. We first obtain normalized perturbations of the flow field within each sub-domain, which are then used as inputs and outputs for different networks. The model is made up of three neural networks namely, Generator, Discriminator, and Attention-CNN. The model's framework can be described as follows. The generator's objective is to produce DNS-like flow field on a coarse-grained mesh in a sub-domain for a given input information of locations of all neighboring particles within the sub-domain and volume-averaged Reynolds number. The training of the generator is pitted against the discriminator, whose objective is to correctly distinguish the actual DNS flow field from the generator produced synthetic flow. The purpose of Attention-CNN is to produce a fine-grained flow in a concentrated region around the particle of interest, known as attention-domain, from the flow output of the generator. Performance of the model (generator, attention-cnn) has been evaluated on test data for the different Reynolds number and volume fraction combinations. The results shows that the present GAN architecture is able to capture the fluid flow around the random distribution of particles to a very good extent.

The trained models are then used to recreate flow field over the entire domain using voronoi tessellation. Visual and quantitative evaluation of the GAN generated flow fields indicate that this methodology holds great potential. In particular, we observe that the GAN generated synthetic flow fields to be substantially better than those generated using superposable wake approximation presented in Moore and Balachandar [28]. Since the superposable wake model is based on pairwise interaction among the particles it can be concluded that the improved performance of GAN is due to its ability to account for the N-body interaction among the particles. Thus, the improvement over pairwise interaction approximation is stronger at higher volume fraction.

Inclusion of symmetries through equivariant networks [37] and physics based loss terms [14, 35] have indicated in other problems substantial potential for improvement of the generalization capabilities of neural networks. Thus, the performance of the GAN model presented in this paper can be further improved by incorporating such augmentation methods. In addition to these generalizing techniques, an interesting investigation that can be carried out is the study of model's performance when it is trained on multiple cases simultaneously. It can provide insight into how the model takes significant variation in particle volume fraction and Reynolds number among samples into consideration.

6. Acknowledgements

This work was sponsored by the Office of Naval Research (ONR) as part of the Multidisciplinary University Research Initiatives (MURI) Program, under grant number N00014-16-1-2617. This work was also partly supported and benefited from the U.S. Department of Energy, National Nuclear Security Administration, Advanced Simulation and Computing Program, as a Cooperative Agreement to the University of Florida under the Predictive Science Academic Alliance Program, under Contract No. de-na0002378. This work was also partly supported by National Science Foundation under Grant No. 1908299.

Appendix A.

Appendix A.1. Loss Functions

As described in section 3 the current ML model contains three neural networks namely, Generator ($\mathcal{G}$), Discriminator ($\mathcal{D}$), and Attention CNN ($\mathcal{A}$). Loss functions used for the training of these networks have been defined below. The loss functions of $\mathcal{G}$ and $\mathcal{D}$ are derived from the objective function given by (6).

Generator loss function:

$$\mathcal{L}_{\mathcal{G}} = \mathbb{E}_{\boldsymbol{c}\in P_c}\left[-\mathcal{D}(\mathcal{G}(\boldsymbol{c}))\right] + \gamma\|(\boldsymbol{\xi}-\boldsymbol{\eta}).I_f\|_1\,, \tag{A.1}$$

$inpt$	input to a network
$oupt$	output of a network
$chls_{in}$	incoming channels for a layer
$chls_{out}$	outgoing channels of a layer
Tsr_{in}	input tensor size of a layer
Tsr_{out}	output tensor size of a layer
$fltr$	size of filter (same is all three directions)
$strd$	stride in a layer
pdg	padding at each boundary in a layer
af	activation function
BN	batch normalization [40]
skp	skip connection
$Conv(Tsr_{in}, chls_{in}, chls_{out}, fltr, strd, pdg, Tsr_{out})$	convolution layer
$Linear(Tsr_{in}, chls_{in}, chlsout, Tsr_{out})$	linear layer
$TConv(Tsr_{in}, chls_{in}, chls_{out}, fltr, strd, pdg, Tsr_{out})$	transpose convolution layer

Table A.4: Neural network notation system

Discriminator loss function:

$$\mathcal{L}_{\mathcal{D}} = \mathbb{E}_{\boldsymbol{\xi}\in P_{\xi}}\left[-\log \mathcal{D}(\boldsymbol{\xi})\right] + \mathbb{E}_{\boldsymbol{c}\in P_{c}}\left[-\log(1-\mathcal{D}(\mathcal{G}(\boldsymbol{c})))\right] + \lambda \mathbb{E}_{\widetilde{\boldsymbol{\xi}}}\left[\left\|(\nabla \mathcal{D})_{\widetilde{\boldsymbol{\xi}}}\right\|_2^2\right] \tag{A.2}$$

Attention CNN loss function:

$$\mathcal{L}_{\mathcal{A}} = \gamma \left\|(\boldsymbol{\psi}-\boldsymbol{\zeta}).I_f\right\|_1, \tag{A.3}$$

where, $(\boldsymbol{\xi}-\boldsymbol{\eta})\,.\,I_f$ represents element-wise multiplication between $(\boldsymbol{\xi}-\boldsymbol{\eta})$ and I_f, which are arrays of same size along *x, y, z*, and $\|\cdot\|_1$ stands for $L1$-norm. Here I_f is the 3D indicator function defined in (7). In this study, (γ, λ) are chosen to be (200,1000) and the sensitivity of results to a $\pm 10\%$ change in these parameters is not large.

Appendix A.2. Architectural and Training Details

Neural network construction and training was performed using PyTorch framework [38]. The notation system given in table A.4 was followed in describing architectures of networks. The details of the 14 layers of the generator $\mathcal{G}$ are presented in table A.5 following the notation given in table A.4. The details of the 6 layers of the discriminator $\mathcal{D}$ are presented in table A.6. The details of the 3 layers of the attention-cnn $\mathcal{A}$ are presented in table A.7. All ML models reported in this work were trained for forty epochs. Asymptotic nature of loss values for the networks was observed in all cases before the end of fortieth epoch. Adam optimizer [39] was used for optimizing the coefficients/parameters of the three networks. Learning rate scheduler of type ReduceLROnPlateau was used for the generator and attention-cnn. An initial learning rate of 10^{-4} was used for $\mathcal{G}$ and $\mathcal{A}$. In $\mathcal{D}$ the initial learning rate was 10^{-5}. All the convolutional layers were initialized with a normal distribution having a mean of 0 and a standard deviation of 0.2, and the PyTorch default weight initializaton was used for Batch Normalization. A batch size of eight was used for all training processes to ensure sufficient parameter/coefficient update iterations occur by the end of forty epochs.

Evolution of mean training loss per epoch has been shown in Figure A.17 for the generator and the attention-cnn trained on a $N_{sub,tr} = N_{tr}/2$ dataset belonging to case 1. $L1$-norm presented in subfigure (a) corresponds to $\|(\boldsymbol{\xi}-\boldsymbol{\eta}).I_f\|_1$ for $\mathcal{G}$ and $\|(\boldsymbol{\psi}-\boldsymbol{\zeta}).I_f\|_1$ for $\mathcal{A}$. Training loss is also shown in terms of R^2 in subfigure (b) as R^2 is the metric used to quantify networks' test performance. A similar training loss behavior exists for the remaining model training runs presented in this work.

$inpt = \boldsymbol{c}$
$B1$: $Conv(64^3, 1, 16, 3, 1, 1, 64^3)$, BN, af = LeakyReLU(0.2)
$B2$: $Conv(64^3, 16, 32, 3, 1, 1, 64^3)$, BN,af = LeakyReLU(0.2)
$B3$: $Conv(64^3, 32, 64, 4, 2, 1, 32^3)$, BN, af = LeakyReLU(0.2)
$B4$: $Conv(32^3, 64, 128, 4, 2, 1, 16^3)$, BN, af = LeakyReLU(0.2)
$B5$: $Conv(16^3, 128, 256, 4, 2, 1, 8^3)$, BN, af = LeakyReLU(0.2)
$B6$: $Conv(8^3, 256, 512, 4, 2, 1, 4^3)$, BN, af = LeakyReLU(0.2)
$B7$: $TConv(4^3, 512, 256, 4, 2, 1, 8^3)$, BN, af = LeakyReLU(0.2), $skp = 5$
$B8$: $TConv(8^3, 512, 128, 4, 2, 1, 16^3)$, BN, af = LeakyReLU(0.2), $skp = 4$
$B9$: $TConv(16^3, 256, 64, 4, 2, 1, 32^3)$, BN, af = LeakyReLU(0.2), $skp = 3$
$B10$: $TConv(32^3, 128, 32, 4, 2, 1, 64^3)$, BN, af = LeakyReLU(0.2), $skp = 2$
$B11$: $TConv(64^3, 64, 16, 3, 1, 1, 64^3)$, BN, af = LeakyReLU(0.2), $skp = 1$
$B12$: $TConv(64^3, 32, 128, 3, 1, 1, 64^3)$, BN, af = LeakyReLU(0.2)
$B13$: $TConv(64^3, 128, 64, 3, 1, 1, 64^3)$, BN, af = LeakyReLU(0.2)
$B14$: $Conv(64^3, 64, 4, 3, 1, 1, 64^3)$
$oupt = \boldsymbol{\eta}$

Table A.5: $\mathcal{G}$ architecture

$inpt = \boldsymbol{c} \cup \{\boldsymbol{\xi}$ or $\boldsymbol{\eta}\}$,
concatenation of $\boldsymbol{\xi}$ or $\boldsymbol{\eta}$ and $\boldsymbol{c}$ along channels
$B1$: $Conv(64^3, 5, 16, 3, 1, 1, 64^3)$, af = LeakyReLU(0.2)
$B2$: $Conv(64^3, 16, 32, 3, 1, 1, 64^3)$, af = LeakyReLU(0.2)
$B3$: $Conv(64^3, 32, 64, 4, 2, 1, 32^3)$, af = LeakyReLU(0.2)
$B4$: $Conv(32^3, 64, 128, 4, 2, 1, 16^3)$, af = LeakyReLU(0.2)
$B5$: $Conv(16^3, 128, 256, 4, 2, 1, 8^3)$, af = LeakyReLU(0.2)
flatten (256 channels, 8^3 tensor)
$B6$: $Linear$(flattened tensor,256*8*8*8,1, single value), af = sigmoid
$oupt = q$

Table A.6: $\mathcal{D}$ architecture

$inpt = \boldsymbol{\eta}$
$B1$: $TConv(16^3, 4, 64, 4, 2, 1, 32^3)$, BN, af = LeakyReLU(0.2)
$B2$: $TConv(32^3, 64, 128, 4, 2, 1, 64^3)$, BN, af = LeakyReLU(0.2)
$B3$: $Conv(64^3, 128, 4, 3, 1, 1, 64^3)$
$oupt = \boldsymbol{\zeta}$

Table A.7: $\mathcal{A}$ architecture

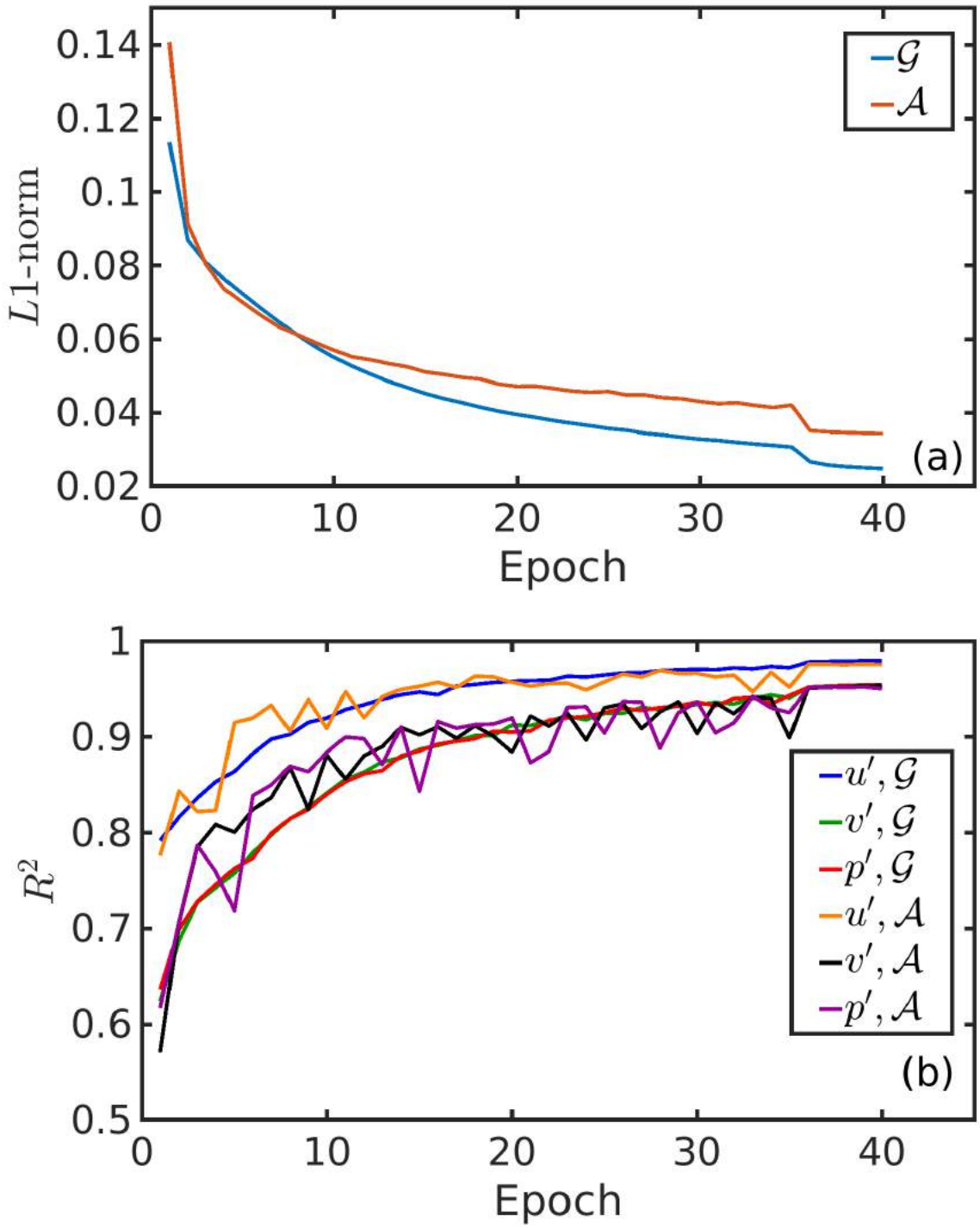


Figure A.17: Training loss evolution with epochs for a $N_{sub,tr} = N_{tr}/2$ subset from Case $Re = 39.47$, $\phi = 0.11$ using (a) $L1$-norm and (b) R^2

Appendix A.3. Computational Cost

Neural network training and testing was carried out using 4 Nvidia GeForce RTX 2080Ti GPUs for all of the results presented in this work. Training time required for a single batch of size 8 is 1.83 seconds per epoch. Testing time taken for a single batch containing 8 samples is 0.573 seconds.

References

[1] S. Balachandar, J. K. Eaton, Turbulent dispersed multiphase flow, Annual Review of Fluid Mechanics 42 (2010) 111–133. URL: https://doi.org/10.1146/annurev.fluid.010908.165243. doi:10.1146/annurev.fluid.010908.165243. arXiv:https://doi.org/10.1146/annurev.fluid.010908.165243.

[2] P. Magnico, Hydrodynamic and transport properties of packed beds in small tube-to-sphere diameter ratio: pore scale simulation using an eulerian and a lagrangian approach, Chemical Engineering Science 58 (2003) 5005 – 5024. URL: http://www.sciencedirect.com/science/article/pii/S0009250903002823. doi:https://doi.org/10.1016/S0009-2509(03)00282-3.

[3] M. Uhlmann, An immersed boundary method with direct forcing for the simulation of particulate flows, Journal of Computational Physics 209 (2005) 448 – 476. URL: http://www.sciencedirect.com/science/article/pii/S0021999105001385. doi:https://doi.org/10.1016/j.jcp.2005.03.017.

[4] S. Tenneti, R. Garg, S. Subramaniam, Drag law for monodisperse gassolid systems using particle-resolved direct numerical simulation of flow past fixed assemblies of spheres, International Journal of Multiphase Flow 37 (2011) 1072 – 1092. URL: http://www.sciencedirect.com/science/article/pii/S0301932211001170. doi:https://doi.org/10.1016/j.ijmultiphaseflow.2011.05.010.

[5] A. A. Zaidi, T. Tsuji, T. Tanaka, A new relation of drag force for high stokes number monodisperse spheres by direct numerical simulation, Advanced Powder Technology 25 (2014) 1860 – 1871. URL: http://www.sciencedirect.com/science/article/pii/S0921883114002015. doi:https://doi.org/10.1016/j.apt.2014.07.019.

[6] G. Akiki, S. Balachandar, Immersed boundary method with non-uniform distribution of lagrangian markers for a non-uniform eulerian mesh, Journal of Computational Physics 307 (2016) 34 – 59. URL: http://www.sciencedirect.com/science/article/pii/S0021999115007597. doi:https://doi.org/10.1016/j.jcp.2015.11.019.

[7] R. Beetstra, M. A. van der Hoef, J. A. M. Kuipers, Drag force of intermediate reynolds number flow past mono- and bidisperse arrays of spheres, AIChE Journal 53 (2007) 489–501. URL: https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.11065. doi:10.1002/aic.11065. arXiv:https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.11065.

[8] L. Rong, K. Dong, A. Yu, Lattice-boltzmann simulation of fluid flow through packed beds of uniform spheres: Effect of porosity, Chemical Engineering Science 99 (2013) 44 – 58. URL: http://www.sciencedirect.com/science/article/pii/S0009250913003679. doi:https://doi.org/10.1016/j.ces.2013.05.036.

[9] S. Bogner, S. Mohanty, U. Rde, Drag correlation for dilute and moderately dense fluid-particle systems using the lattice boltzmann method, International Journal of Multiphase Flow 68 (2015) 71 – 79. URL: http://www.sciencedirect.com/science/article/pii/S0301932214001803. doi:https://doi.org/10.1016/j.ijmultiphaseflow.2014.10.001.

[10] S. L. Brunton, B. R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annual Review of Fluid Mechanics 52 (2020) 477–508. URL: https://doi.org/10.1146/annurev-fluid-010719-060214. doi:10.1146/annurev-fluid-010719-060214. arXiv:https://doi.org/10.1146/annurev-fluid-010719-060214.

[11] K. Duraisamy, G. Iaccarino, H. Xiao, Turbulence modeling in the age of data, Annual Review of Fluid Mechanics 51 (2019) 357–377. URL: https://doi.org/10.1146/annurev-fluid-010518-040547. doi:10.1146/annurev-fluid-010518-040547. arXiv:https://doi.org/10.1146/annurev-fluid-010518-040547.

[12] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 481–490.

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

[14] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017. arXiv:1711.10561.

[15] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations, 2017. arXiv:1711.10566.

[16] Z. Mao, A. D. Jagtap, G. E. Karniadakis, Physics-informed neural networks for high-speed flows, Computer Methods in Applied Mechanics and Engineering 360 (2020) 112789. URL: http://www.sciencedirect.com/science/article/pii/S0045782519306814. doi:https://doi.org/10.1016/j.cma.2019.112789.

[17] Y. Qi, J. Lu, R. Scardovelli, S. Zaleski, G. Tryggvason, Computing curvature for volume of fluid methods using machine learning, Journal of Computational Physics 377 (2019) 155161. URL: http://dx.doi.org/10.1016/j.jcp.2018.10.037. doi:10.1016/j.jcp.2018.10.037.

[18] A. B. Farimani, J. Gomes, V. S. Pande, Deep learning the physics of transport phenomena, 2017. arXiv:1709.02432.

[19] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, 2014. arXiv:1406.2661.

[20] I. Goodfellow, Nips 2016 tutorial: Generative adversarial networks, 2016. arXiv:1701.00160.

[21] Y. Xie, E. Franz, M. Chu, N. Thuerey, Tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow, ACM Trans. Graph. 37 (2018). URL: https://doi.org/10.1145/3197517.3201304. doi:10.1145/3197517.3201304.

[22] Q. Zhou, L.-S. Fan, Direct numerical simulation of moderate-reynolds-number flow past arrays of rotating spheres, Physics of Fluids 27 (2015) 073306. URL: https://doi.org/10.1063/1.4927552. doi:10.1063/1.4927552. arXiv:https://doi.org/10.1063/1.4927552.

[23] Y. Tang, E. A. J. F. Peters, J. A. M. Kuipers, S. H. L. Kriebitzsch, M. A. van der Hoef, A new drag correlation from fully resolved simulations of flow past monodisperse static arrays of spheres, AIChE Journal 61 (2015) 688–698. URL: https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.14645. doi:10.1002/aic.14645. arXiv:https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.14645.

[24] G. Akiki, T. L. Jackson, S. Balachandar, Force variation within arrays of monodisperse spherical particles, Phys. Rev. Fluids 1 (2016) 044202. URL: https://link.aps.org/doi/10.1103/PhysRevFluids.1.044202. doi:10.1103/PhysRevFluids.1.044202.

[25] G. Akiki, T. L. Jackson, S. Balachandar, Pairwise interaction extended point-particle model for a random array of monodispersespheres, Journal of Fluid Mechanics 813 (2017) 882928. doi:10.1017/jfm.2016.877.

[26] G. Akiki, W. Moore, S. Balachandar, Pairwise-interaction extended point-particle model for particle-laden flows, Journal of Computational Physics 351 (2017) 329 – 357. URL: http://www.sciencedirect.com/science/article/pii/S0021999117306848. doi:https://doi.org/10.1016/j.jcp.2017.07.056.

[27] W. Moore, S. Balachandar, G. Akiki, A hybrid point-particle force model that combines physical and data-driven approaches, Journal of Computational Physics 385 (2019) 187 – 208. URL: http://www.sciencedirect.com/science/article/pii/S0021999119301111. doi:https://doi.org/10.1016/j.jcp.2019.01.053.

[28] W. C. Moore, S. Balachandar, Lagrangian investigation of pseudo-turbulence in multiphase flow using superposable wakes, Phys. Rev. Fluids 4 (2019) 114301. URL: https://link.aps.org/doi/10.1103/PhysRevFluids.4.114301. doi:10.1103/PhysRevFluids.4.114301.

[29] H. Thanh-Tung, T. Tran, S. Venkatesh, Improving generalization and stability of generative adversarial networks, in: International Conference on Learning Representations, 2019. URL: https://openreview.net/forum?id=ByxPYjC5KQ.

[30] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, D. J. Schwab, A high-bias, low-variance introduction to machine learning for physicists, Physics Reports 810 (2019) 1124. URL: http://dx.doi.org/10.1016/j.physrep.2019.03.001. doi:10.1016/j.physrep.2019.03.001.

[31] L. Mescheder, A. Geiger, S. Nowozin, Which training methods for gans do actually converge?, 2018. arXiv:1801.04406.

[32] T. Che, Y. Li, A. P. Jacob, Y. Bengio, W. Li, Mode regularized generative adversarial networks, 2016. arXiv:1612.02136.

[33] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann, C. Sutton, Veegan: Reducing mode collapse in gans using implicit variational learning, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 3308–3318. URL: http://papers.nips.cc/paper/6923-veegan-reducing-mode-collapse-in-gans-using-implicit-variational-learning.pdf.

[34] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training gans, 2016. arXiv:1606.03498.

[35] A. Subramaniam, M. L. Wong, R. D. Borker, S. Nimmagadda, S. K. Lele, Turbulence enrichment using physics-informed generative adversarial networks, 2020. arXiv:2003.01907.

[36] C. Rycroft, Voro++: a three-dimensional voronoi cell library in c++, 2009, Chaos 19 (2009) 041111.

[37] R. Wang, R. Walters, R. Yu, Incorporating symmetry into deep dynamics models for improved generalization, 2020. arXiv:2002.03061.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[39] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014. arXiv:1412.6980.

[40] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. arXiv:1502.03167.