Transfer Learning Using Multi-fidelity Data on a Surrogate Convolutional Neural Network for Turbine Wake Modelling

by Ian Wan

Imperial College London, Department of Earth Science and Engineering

MSc Applied Computational Science and Engineering

Independent Research Project

GitHub username: acse-4bc892f6

Email: man.wan18@imperial.ac.uk

Submission date: 2nd September, 2022

Supervisors: Prof. Matthew Piggot, Dr. Simon Warder

Contents

Abstract

A workflow that uses multi-fidelity data for transfer learning (MFTL) is implemented to a surrogate convolutional neural network (CNN) to reduce training time and minimise training set size of high-fidelity (HF) data, which is usually expensive and time-consuming to generate, without compromising on accuracy.

PyWake package is used to generate synthetic training, validation and testing sets. Fuga wake model is used to generate HF data, and the other wake models are used to generate low-fidelity (LF) data. The performance of the CNNs that has undergone MFTL (MFTL CNNs) is compared with a "benchmark" Fuga CNN that is trained using 5000 Fuga model data. Performance is quantified by computing pixel-wise percentage error (pwpe) relative to a testing set that contains 500 entries generated from Fuga model. LF to Fuga training set size is varied to investigate the effect this ratio has on the pwpe of MFTL CNNs. The effect of LF model on pwpe of MFTL CNNs is also investigated.

MFTL CNNs outperform "benchmark" Fuga CNN when LF to Fuga training set size ratio is 2500/300, no matter the choice of LF model. MFTL CNNs at other ratios also outperform "benchmark" Fuga CNN, but is dependent on LF model. LF model affects the range and outliers of pwpe, but not median pwpe. Random seed used for testing set generation significantly affects the distribution, median, range and outliers of pwpe. LF model that is structurally similar to Fuga does not lead to better performance of MFTL CNN.

1 Introduction

Turbine wakes, shown in figure 1, is a region of reduced airflow velocity (velocity deficit) and higher turbulence intensity (TI) formed behind a turbine. Turbine wakes cause up to 10-20% drop in wind farm power output [1]. Wake modelling must be accurate, quick and cheap to maximise power output of a wind farm through layout optimisation as well as wake steering through active yaw control.

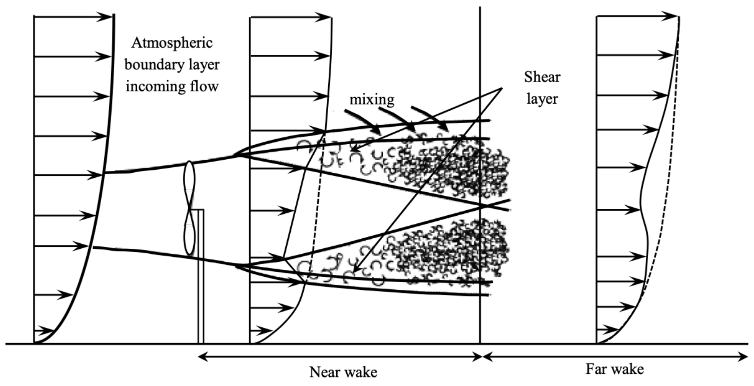


Figure 1: Evolution of turbine wake. Airflow slows down after interacting with the rotor, leading to velocity deficit and increased turbulence intensity. Shear layers are created that expand via mixing until they reach the wake axis. The direction which the wind speed vectors are pointing (to the right horizontally) is the downstream direction. Wind speed before airflow interacts with the rotor is the free stream wind speed. After [2].

1.1 Analytical models

Due to quick computation and reasonable accuracy, analytical models are often used for wind farm layout optimisation and active yaw control. Analytical models use simplified momentum equations to calculate velocity deficit and wake expansion based on an assumed initial velocity profile in the near-wake region [2] (Figure 1). Some assumptions are more realistic than others. For instance, normalised velocity deficit is assumed to follow the rectangular function in the Jensen model [3], while normalised velocity deficit is assumed to follow the Gaussian function in the Gaussian model [4], and the Gaussian assumption is more realistic than Jensen. Another simplification in analytical models is the assumption of constant wake growth rate inside a wind farm, which is unrealistic since wake growth rate increases with TI, and TI increases substantially under turbine wake flow [5].

1.2 Computational fluid dynamics solvers

Computational fluid dynamics (CFD) solvers are more accurate than analytical models, because CFD solvers incorporate more physics of the interaction between airflow and rotor, thus they are more computationally intensive and time-consuming. Two common CFD solvers are Reynolds averaged Navier–Stokes (RANS) and Large eddy simulation (LES). LES is roughly three orders of magnitude more computationally intensive than RANS [6] because of more complete physics, but LES is more accurate than RANS as a result.

A commonly used RANS solver is the $k - \epsilon$ eddy viscosity model. Predicted eddy viscosity from standard $k - \epsilon$ model is often too high. Higher eddy viscosity causes turbulent momentum to diffuse faster, leading to faster wake recovery and over-estimation of wind farm power production [7]. Methods used to suppress the predicted eddy viscosity in the standard $k - \epsilon$ model include adding an extra source term that dissipates the turbulence energy by approximating the transport equation [8], and adding a flow-dependent variable that suppresses eddy viscosity in high velocity gradient areas [6].

LES coupled with actuator-disk model with rotation (ADM-R) has been shown to produce accurate predictions of wind farm power outputs [9], which indicates accurate wake modelling. ADM-R computes turbine-induced forces by distributing the local lift and drag forces on the rotor disk area following the blade-element momentum theory [10].

LES and RANS are generally too expensive for layout optimisation due to the large number of turbine configurations that needs to be simulated, and their run times are too long for active yaw control.

1.3 Machine learning - surrogate neural networks

A multi-layer feed-forward neural network (FNN) can approximate any function [11]. More importantly, forward propagation of a trained neural network (NN) is cheaper and quicker than running CFD solvers [12].

In a data-driven approach, a NN acts as a surrogate model that predicts flow field from inflow conditions and from the object(s) which the air flows past, replacing CFD solvers. Surrogate NNs have been used for turbine wake modelling. A deep FNN was trained to predict flow velocity given inflow conditions and wind farm configuration, as shown in Figure 2. Computational time is reduced by an order of magnitude, and mean absolute error compared to results from analytical models is only 1.5% [13]. A NN was trained to predict wake flows from inflow conditions, and predictions from the NN closely matches the predictions produced from LES and field measurements [14].

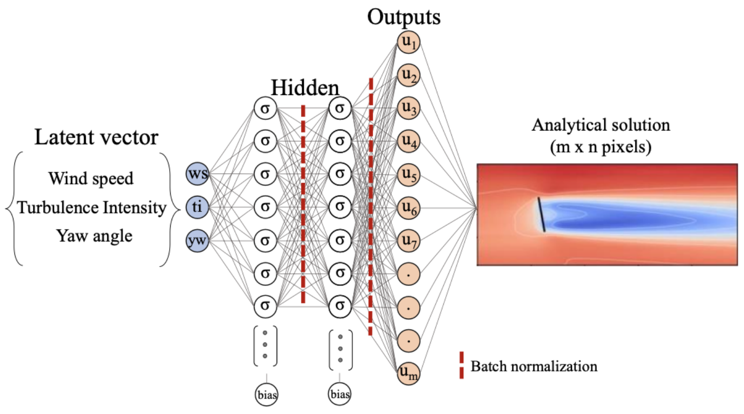


Figure 2: An example of a surrogate NN. This NN taks inflow conditions including wind speed, turbulence intensity and yaw angle as input and output flow velocity. After [13].

Surrogate NN is still limited by generating data using analytical model or CFD solver for training, validation and testing. For instance, data is generated using RANS coupled with ADM-R in [14]. Moreover, surrogate NN is difficult to analyse and interpret because the learned physics is scattered among the model parameters, and surrogate NN is prone to over-fitting [15].

1.4 Multi-fidelity transfer learning

Since high-fidelity (HF) data is usually expensive and time-consuming to generate, using large HF dataset to train a NN is not viable. Transfer learning transfers information from one domain to another related domain [16], enabling a NN trained for a specific task to solve another similar task. As shown in figure 3, the validation error is smaller in transfer learning than in standard training when the HF dataset is small, hence transfer learning reduces error when data generation budget is constrained.

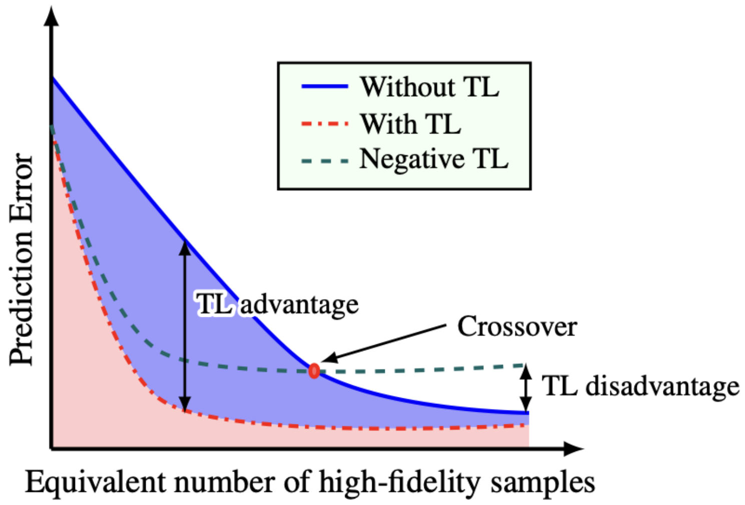


Figure 3: Transfer learning (TL) reduces prediction error, especially when the HF dataset is small. Negative TL occurs if the datasets used in TL are unrelated. After [17].

To apply transfer learning in practice, a NN is trained using a larger low-fidelity (LF) dataset, then it is further trained using smaller HF dataset.

This approach will be referred to as multi-fidelity transfer learning (MFTL).

MFTL was applied to a surrogate recurrent residual U-Net to predict subsurface flow. Computational cost is reduced by 90% and training time by 40%, while the median relative error in predicted saturation is comparable to HF solver [18]. A CNN was trained to solve multi-phase flow problems through MFTL. The final CNN gives accurate predictions comparable to fine-mesh PDE solvers while being two orders of magnitude faster. Moreover, data generation budget is decreased by seven fold compared to training the CNN on HF data alone. Additionally, for a fixed data generation budget, the root-mean-squared error of the predictions is smaller than both CNN trained only on HF dataset and CNN trained only on LF dataset [19].

2 Objectives

MFTL is a suitable approach to train surrogate NNs for turbine wake modelling, because LF data can be easily generated from cheap and efficient analytical models, while HF data is lacking due to expensive and time-consuming CFD solvers. No literature to date has been found to apply MFTL to surrogate NN for turbine wake modelling.

A surrogate CNN that takes wind speed, TI and yaw angle (inflow condition) as input to predict flow fields around a given configuration of wind turbines is available on GitHub [20, 21]. Building upon this work and the work from [18] and [19], the main objectives of this project are:

- Implement MFTL workflow following [18] and [19] to the surrogate CNN [20, 21] – the first implementation of MFTL on a surrogate NN for turbine wake modelling.
- Compare predictions generated from the surrogate CNNs that has undergone MFTL (MFTL CNNs) with predictions generated from a "benchmark" CNN only trained using HF data and with simulations generated from HF model.
- Vary the ratio of LF to HF training set size to determine the effect that this ratio has on the performance of MFTL CNN.
- Investigate the effect of the LF model and HF model combination on performance of MFTL CNN.

3 Software description

FLORIS [22] and PyWake [23] are two popular open-source Python packages for wind farm modelling and optimisation. Both packages contains a suite of wake models – FLORIS contains 4 wake models, while PyWake contains 10 wake models.

All PyWake wake models, key assumptions and average time to generate 100 simulations (on M1 chip, see Appendix section 8.4 for full table of simulation times) are listed below:

- NOJDeficit (Jensen)
  - Assume normalised velocity deficit D_u follows the rectangular function.
  - D_u is computed as shown in equation 1:

    $$D_u = \frac{1 - \sqrt{1 - C_T}}{(1 + \frac{2\alpha x}{D})^2}, \ D_u = \frac{\Delta U}{U_\infty} \tag{1}$$

    where C_T is the thrust coefficient, D is the rotor diameter, x is the downstream distance, and U_∞ is free stream wind speed, α is the rate of wake radius increase [24].
  - Assume α is linear with x:

    $$\alpha = a_1 + a_2 x \tag{2}$$

    where a_1 and a_2 are tunable parameters.
  - 7.69s per 100 simulations.
- TurboJensen
  - Based on Jensen model.
  - Modified wake expansion rate – assume it is a function of x, local TI and a model calibration constant. [25].
  - 7.68s per 100 simulations.

- FugaDeficit (Fuga)
  - Generate flow fields from a look-up table (LUT) that is computed from linearised RANS equations.
  - 7.68s per 100 simulations.
- BastankhahGaussianDeficit (BGauss)
  - Assume normalised velocity deficit $\frac{\Delta U}{U_\infty}$ follows Gaussian function.
  - $\frac{\Delta U}{U_\infty}$ computed as shown in equation 3:

    $$\frac{\Delta U}{U_\infty} = (1 - \sqrt{1 - \frac{C_T}{8(\frac{k^* x}{d_0} + \epsilon)^2}} \times \exp - \frac{1}{(\frac{2k^*}{d_0+\epsilon})^2}(\frac{z - z_h}{d_0})^2 + (\frac{y}{d_0})^2) \tag{3}$$

    where y is the spanwise distance, z is the vertical distance, z_h is the hub height, k^* is the wake growth rate, d_0 is rotor diameter, and ϵ is defined as shown in equation 4 [4]:

    $$\epsilon = 0.25\sqrt{\beta},\ \beta = \frac{11 + \sqrt{1 - C_T}}{2\sqrt{1 - C_T}} \tag{4}$$
  - Assume wake growth rate k^* is linear with x (like Jensen).
  - 7.70s per 100 simulations.
- IEA37SimpleBastankhahGaussianDeficit (SBGauss)
  - Simplified BGauss model.
  - Assume rate of wake area expansion β to be constant as shown in equation 5 [26, 27]:

    $$\beta \approx C_T = 0.9637188 \tag{5}$$
  - 7.70s per 100 simulations.
- NiayifarGaussianDeficit (NGauss)
  - Extension of BGauss model.

  - Assume wake growth rate as a function of local TI I:

$$k^* = a_1 I + a_2 \tag{6}$$

    a_1 and a_2 are 0.3847 and 0.003678 respectively, determined from LES data [5].
  - 7.74s per 100 simulations.

- ZhongGaussian (ZGauss)
  - Based on NGauss model.
  - Improved modelling of yawed wind turbine wakes [28, 26].
  - 8.13s per 100 simulations.

- CarbajofuertesGaussian (CGauss)
  - Based on the BGauss model.
  - Parameters for wake growth function and ϵ (equation 4) have been updated based on field experiment data obtained from a measurement campaign [29].
  - 7.67s per 100 simulations.

- TurboGaussianDeficit (TurboGauss)
  - Based on BGauss model.
  - Modified wake growth function – assume it is a function of both TI and wake-generated turbulence, the latter modelled from an analytical expression proposed by Frandsen [30].
  - 7.81s per 100 simulations.

- GCL (Larsen) [31]
  - Solves a system of simplified, thin shear layer approximated Navier-Stokes equation to first or second order.
  - Assumptions made include
    - Rotational symmetry in the wake
    - Pseudo-uniform inflow field

        * Incompressible fluid
    - Ignored viscous terms
    - 7.69s per 100 simulations.

Because more wake models are available in PyWake than FLORIS, and one objective of this project is to examine the effects of LF model and HF model combination used in MFTL on the performance of CNN, PyWake is used instead of FLORIS for this project.

NGauss allows users to input constants $a1, a2$ that controls the wake width expansion rate. The default values $a1 = 0.38, a2 = 0.004$ are used in this project i.e. the original values from NGauss LES data rounded to 2 decimal and 3 decimal places respectively.

The function 'PywakeWorkflow' is implemented to generate flow fields using PyWake. It takes inflow conditions and wake model as argument. The workflow of the function is shown in figure 4. The flow fields around the turbine(s) are simulated using an engineering wind farm model (WindFarmModel) in PyWake. Two types of WindFarmModel are available in PyWake: PropagateDownwind and All2AllIterative. The latter is used, because it accounts for upstream blockage effects and iterates until wind speed converges to a tolerance of 10^{-6} by default, which is the tolerance used in this project. The turbulence, deflection and superposition models are fixed at Crespo Hernandez [32], Jiminez [33], and squared sum [34] respectively. The goal of this project is to examine the effect of wake models used in MFTL on CNN performance rather than the effect of turbulence model, deflection model or superposition model used in MFTL on CNN performance. Therefore, the turbulence, deflection and superposition models should act as controlled variables that remain unchanged. The default domain size is $x \in [0, 3000], y \in [-200, 200]$, following the original implementation [20, 21].

The CNN is written using PyTorch [35] and its structure is shown in figure 5. Inflow conditions are passed into the CNN as a 1×3 tensor with a single channel, and the CNN outputs the flow field as a 163×163 tensor

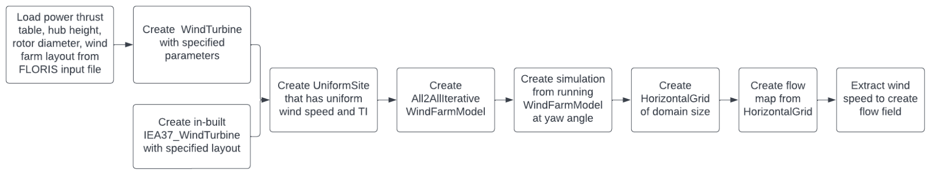

Load power thrust table, hub height, rotor diameter, wind farm layout from FLORIS input file
Create WindTurbine with specified parameters
Create in-built IEA37_WindTurbine with specified layout
Create UniformSite that has uniform wind speed and TI
Create All2AllIterative WindFarmModel
Create simulation from running WindFarmModel at yaw angle
Create HorizontalGrid of domain size
Create flow map from HorizontalGrid
Extract wind speed to create flow field

Figure 4: PywakeWorkflow function that generates flow field using PyWake.

with a single channel. The CNN structure have not been changed from the original implementation [20, 21].

Inflow conditions are first generated following a uniform distribution. This process is not completely random due to the random seed, which is needed to maintain consistent uniform distribution while generating data using different wake models to train different CNNs. To reduce the size of the training and evaluation sets, the range of values for wind speed, TI and yaw angle are set as $[3, 12]\mathrm{m\,s^{-1}}$, $[0.015, 0.25]$ and $[-30, 30]°$ respectively, which is the same as the original implementation [21]. Wind speed range is chosen as such because $3m/s$ is the lower operational limit, whereas potential power gain is limited beyond $12m/s$; TI range and yaw angle range are chosen based on measurements [13]. Flow fields are generated from wake model based on the inflow conditions using 'PywakeWorkflow' function (figure 4). Flow fields are combined with the inflow conditions to create Pytorch tensor training and evaluation sets. Random seed for training and evaluation set generation is defaulted at 1.

During all training stages, Adam optimizer [37] and a learning rate scheduler are used. Mean square error (MSE) cost function is used to compute the training loss as shown in equation 7:

$$MSE = \frac{1}{N_{train}} \sum_{n=0}^{N_{train}} (|y_n - predict_n|^2) \tag{7}$$

where N_{train} is the training set size, y is the PyWake generated flow field, and $predict$ is the flow field generated from the trained CNN.

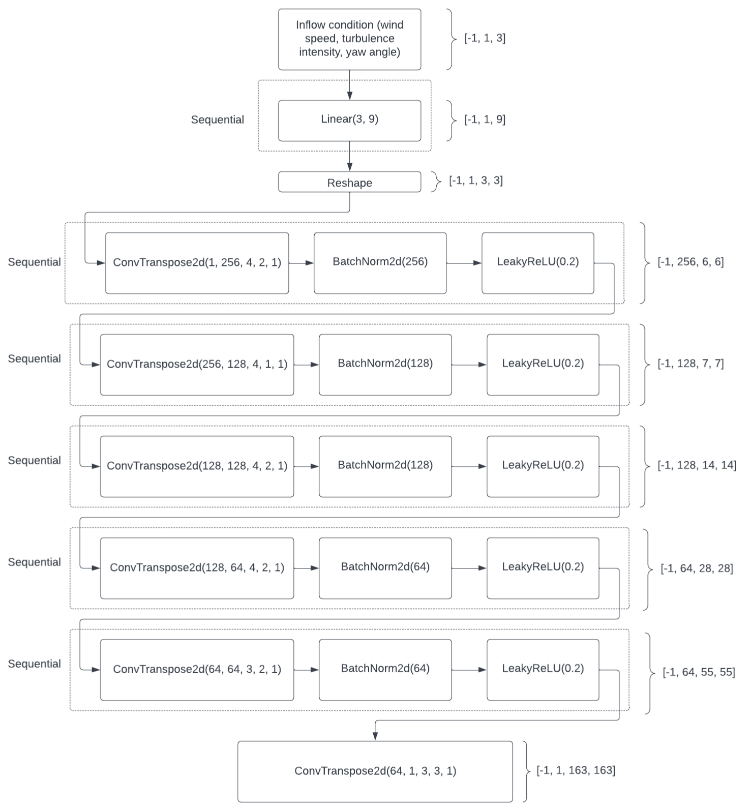


Figure 5: Structure of CNN, which contains 1 fully connected layer, 5 sequential layers, and a final deconvolutional layer. Each sequential layer contains 2D devoncolution, 2D batch normalisation, and leaky ReLU activation function [36]. Type of layer, number of features, number of filters, kernel size, stride, padding, and the shape of output after each layer are shown.

Average pixel-wise percentage error (pwpe) is used to compute the validation error of the evaluation set as shown in equation 8:

$$average\_pwpe = \frac{1}{N_{eval}} \times \sum_{n=0}^{N_{eval}} (\frac{1}{N_{image}^2} \times \sum_{i=0}^{N_{image}^2} \frac{(100 * abs(y_{n,i} - predict_{n,i})}{max(y_{n,i})} \tag{8}$$

where N_{eval} is the evaluation set size, and N_{image} is the image size which is 163 with the current CNN structure i.e. image dimension is 163×163.

The training optimizer, cost function and validation error are the same as the original implementation [21].

A MFTL workflow is implemented as shown in figure 6, which follows the MFTL workflow from [18] and [19]. This MFTL workflow is chosen because it produced surrogate models that generate accurate results quickly while reducing training time and cost (section 1.4). Training set is generated from uniformly distributed lists of wind speed, TI and yaw angle, which are combined to form a tensor (figure 6), hence StratifiedShuffleSplit from scikit-learn is used to split the inflow condition lists from LF training set size to HF training set size before proceeding to the second training phase. This ensures the training sets across the two stages of training are consistent (see Appendix section 8.3 for illustration of splitting inflow condition lists). Evaluation set that has HF evaluation set size is generated separately using the same random seed of 1.

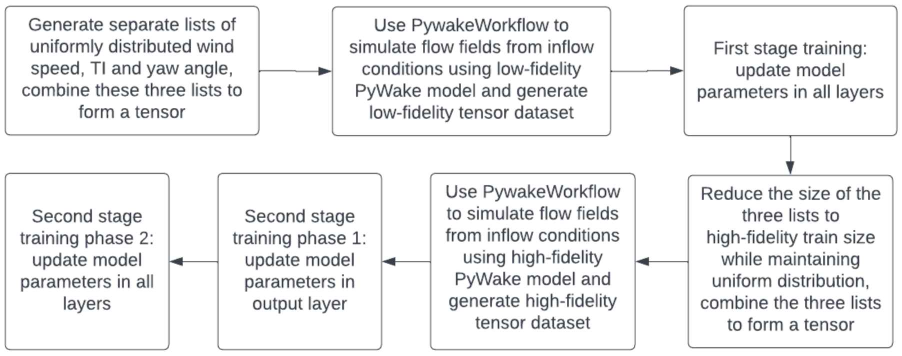

Generate separate lists of uniformly distributed wind speed, TI and yaw angle, combine these three lists to form a tensor
Use PywakeWorkflow to simulate flow fields from inflow conditions using low-fidelity PyWake model and generate low-fidelity tensor dataset
First stage training: update model parameters in all layers
Reduce the size of the three lists to high-fidelity train size while maintaining uniform distribution, combine the three lists to form a tensor
Use PywakeWorkflow to simulate flow fields from inflow conditions using high-fidelity PyWake model and generate high-fidelity tensor dataset
Second stage training phase 1: update model parameters in output layer
Second stage training phase 2: update model parameters in all layers

Figure 6: Workflow of multi-fidelity transfer learning.

Unit tests are written to ensure turbine information are loaded correctly from FLORIS yaml and json files, only model parameters in the output layer of the CNN have requires_grad=True when required, and all model parameters in the CNN have requires_grad=True when required.

The open-source libraries needed to run MFTL workflow are:

- floris 3.1.1

- py_wake
- torch
- numpy
- matplotlib
- sklearn

To run data processing scripts, unit test scripts and print out summary of the CNN model, the following open-source libraries are also needed:

- pandas
- pytest
- torch-summary

4 Methodology

Fuga wake model generates flow fields from a look-up table (LUT) that is computed from linearised RANS equations. The LUT is very general, so using the LUT does not compromise on the flexibility and accuracy of the Fuga wake model (see Appendix section 8.1 for more details). Additionally, it has been shown that Fuga wake model is comparable to CFD solvers while the cost is 5 to 8 orders of magnitude lower, and Fuga fits measurements from two offshore wind farms better than both Jensen and Larsen models [24]. Therefore, Fuga is chosen as the HF model in this project.

10 CNNs are trained on tensor dataset of size 5000 over 100 epochs for all 10 wake models in PyWake, each CNN trained on data generated from one wake model. The CNN trained using 5000 Fuga flow fields is used as a "benchmark" to compare with other MFTL CNNs. The other CNNs trained using 5000 LF flow fields are used as "benchmark" LF CNNs, each corresponding to their LF model. The number of epochs used in later training runs is reduced to 30, since validation error does not reduce and remains low after around 10 epochs, then stabilises after 20 epochs (see Appendix section 8.2 for validation error over epochs of training "benchmark" Fuga CNN).

4.1 Experiment 1

First, following the train set sizes in [18], LF train set size is fixed at 2500 while HF train set sizes of $100, 200, 300, 400$ are used.

Then, the LF to HF size ratio is reduced to 5.5, 3.3 and 1.1 in experiments 2-4 (sections 4.2-4.4), since the optimal ratio lies around $1.1 - 5.5$ according to a multilevel Monte Carlo study [19].

4.2 Experiment 2

Fixing HF train set size at 400, LF train set sizes of 2200, 1320 and 440 are used, so the LF to HF train set size is 2200/400, 1320/400 and 440/400.

4.3 Experiment 3

Fixing HF train set size at 250, LF train set sizes of $1375, 825, 275$ are used, so the LF to HF train set size ratio is 1375/250, 825/250 and 275/250.

4.4 Experiment 4

Lastly, fixing HF train set size at 100, LF train set sizes of 550, 330 and 110 are used, so the LF to HF train set size ratio is 550/100, 330/100 and 110/100.

4.5 Hyper-parameters

In all training runs, the initial learning rate is set at 0.003, and the learning rate scheduler decreases by a factor of 0.6 if average pwpe does not improve after 4 epochs, the same as the original implementation [21]. Batch size is set at 10% of the train set size, which is within the commonly adopted range in literature [13]. Validation set size is set at 20% of the train set size.

4.6 Wind farm layout and wind turbine

The CNNs are trained on 1×3 turbine configuration (see Appendix 8.5 for visualisation). The turbine configuration is obtained from example input files on FLORIS GitHub page [22]. The turbines are loaded from FLORIS input file, since it is more straightforward to examine and change the relevant wind turbine data including power thrust table, rotor diameter, hub height and layout if required than in-built PyWake WindTurbine.

4.7 Hardware

RTX6000 graphics processing unit (GPU) on Imperial high performance computing (HPC) and GPU on Google Colab are used to run experiments 1-3. Experiment 4 is run on Apple M1 chip on MacBookAir M1 2020 version.

4.8 Data Processing and Visualisation

The performance of MFTL CNNs from all experiments is quantified by calculating pwpe of all 500 entries in a synthetic testing set generated using Fuga model. The synthetic testing set is generated using a different random seed of 123. The computed pwpe for all 500 entries is stored in a dataset. pwpe is calculated in the same way as average pwpe (equation 8) but without summing up and dividing the set size:

$$pwpe = \frac{1}{image\_size^2} \times \sum_{i=0}^{image\_size^2} \frac{(100 * abs(y_i - predict_i)}{max(y_i)} \tag{9}$$

For a CNN model trained using only LF wake model data, pwpe quantifies the structural similarity between the LF model and the Fuga model. For MFTL CNNs, pwpe quantifies how closely the flow fields generated from the MFTL CNN matches the flow fields generated from the Fuga model.

Finally, for each PyWake wake model, pwpe computed from Fuga testing set over all LF to Fuga training set size ratios is extracted from the previously saved datasets. The extracted pwpe is plotted in a box and whisker diagram. This is to visualise the effects of changing the LF to Fuga training set size ratio on the performance of the CNN, as well as the effects of the choice of

LF model for the MFTL workflow on pwpe. These box and whisker diagrams are referred to in sections 5.1-5.4.

A unit test is implemented to ensure the pwpe data is extracted from the correct wake model and correct dataset.

5 Results

5.1 Experiment 1

Figure 7 shows the box and whisker plots of computed pwpe for each wake model over the LF to HF training set size ratios 2500/100, 2500/200, 2500/300 and 2500/400.

As expected, CNNs that have only been trained on LF model data performs worse than MFTL CNNs when tested on data generated from Fuga model, as shown by the larger pwpe. CNNs have larger pwpe when they are only trained on 2500 LF model data than when trained on 5000 LF model data. NNs are generally more prone to over-fitting when a dataset size is smaller, so the larger error when training set size is 2500 is likely caused by over-fitting.

The performance of some "benchmark" LF CNNs is comparable to the "benchmark" Fuga CNN, indicated by their similar pwpe. Those LF models are structurally similar to the Fuga model. LF models most structurally similar to Fuga model are Larsen and CGauss. Larsen is expected to be structurally similar to Fuga because, like Fuga, it is based on Navier-Stokes equations. CGauss has good performance possibly because the updated parameters are obtained from field data, thus the analytical expression become closer to the complete physics.

MFTL CNNs outperform "benchmark" Fuga CNN when the ratio is 2500/300 (see Appendix 8.5 for simulated flow fields from Fuga and MFTL CNN at 2500/300). At 2500/200, MFTL CNNs have lower pwpe distribution than "benchmark" Fuga CNN, but their pwpe outliers exceed the pwpe of "benchmark" Fuga CNN.

Structural similarity between LF model and Fuga does not affect pwpe – all MFTL CNNs have similar median pwpe for each LF to Fuga training set size ratio: $2-3\%$ at 2500/100, 1% at 2500/200 and 2500/300, and rises to $3-4\%$ at 2500/400. The similarity of median pwpe across all LF models is likely caused by model parameters in every layer being updated in the second phase of re-training.

The choice of LF model affects the range of pwpe, but no relationship between LF model and pwpe range is found. For example at 2500/200, the box of Larsen, representing the gap between the first and third quartile, is larger than that of TurgoGauss, TurboJensen and ZGauss, but Larsen box is roughly the same size as the other models.

The choice of LF model affects the pwpe outliers, but no relationship is found between the LF model and pwpe outliers. For instance at 2500/100, ZGauss have outliers from 8% to 17%, a larger range than Jensen outliers of $10-16\%$, while Larsen has the smallest range of outliers from 13% to 16%.

5.2 Experiment 2

Figure 8 shows the box and whisker plots of computed pwpe for each wake model over the LF to HF training set size ratios 2200/400, 1320/400, and 440/400.

MFTL CNNs have lower pwpe than "benchmark" Fuga CNN at 440/400, but their pwpe outliers exceed the pwpe of "benchmark" Fuga CNN. The only two exceptions are Larsen and TurboJensen, both of which have slightly better performance than "benchmark" Fuga CNN, indicated by their lower median pwpe and slightly lower pwpe distribution.

Consistent with experiment 1, structural similarity between LF model and Fuga does not affect pwpe – all MFTL CNNs have very similar median pwpe for each LF to Fuga training set size ratio: $1-2\%$ at 2200/400, increasing to $4-5\%$ at 1320/400, then decreasing back to $1-2\%$ at 440/400.

The choice of LF model affects the range of pwpe, consistent with experiment 1. This is most noticeable at 1320/400. The range of the pwpe for

Larsen, TurboJensen and ZGauss is $3\% - 10\%$, but the range of the pwpe for TurboGauss is $1\% - 7\%$. No relationship between LF model and pwpe range is found.

Also consistent with experiment 1, the choice of LF model affects pwpe outliers. At 2200/400, only Jensen and NGauss have no pwpe outliers. At 1320/400, only BGauss, TurboGauss and ZGauss have no pwpe outliers. CGauss, Larsen and SBGauss have outliers over all ratios used in this experiment. No relationship is found between LF model and outliers.

5.3 Experiment 3

Figure 9 shows the box and whisker plots of computed pwpe for each wake model over the LF to HF training set size ratios 1375/250, 825/250, and 275/250.

No MFTL CNNs outperformed "benchmark" Fuga CNN at 1375/250 and 825/250. At 275/250, all MFTL CNNs have similar median pwpe and pwpe distribution to "benchmark" Fuga CNN except BGauss, which has a noticeably higher median pwpe. Only SBGauss and ZGauss performed slightly better than "benchmark" Fuga CNN, shown by their lower median pwpe and lower pwpe distribution.

Consistent with experiments 1 and 2, structural similarity between LF model and Fuga does not affect pwpe – all MFTL CNNs have similar median pwpe, which is $2 - 3\%$ across all ratios in this experiment, although BGauss and TurboGauss have slightly higher median pwpe compared to other wake models across all ratios at $3 - 4\%$.

The choice of LF model has an impact on the range of pwpe, consistent with experiments 1 and 2. At 1375/250, Jensen and NGauss have narrower range of pwpe than other models. BGauss and TurboGauss have larger box than other models at 825/250 and at 275/250. NGauss also have a slightly larger box at 275/250. No relationship is found between LF model and the range of pwpe.

The choice of LF model impacts pwpe outliers, consistent with experiments 1 and 2. Only BGauss and TurboGauss have no outliers at 825/250. At 1375/250, Jensen, NGauss, and ZGauss have noticeably large range of outliers of $5-11\%$, $5-11\%$ and $7-13\%$ respectively. At 825/250, NGauss and TurboJensen have noticeably large range of outliers of $6-13\%$ and $7-13\%$ respectively. No relationship is found between LF model and pwpe outliers.

5.4 Experiment 4

Figure 10 shows the box and whisker plots of computed pwpe for each wake model over the LF to HF training set size ratios 550/100, 330/100, and 110/100.

The number and range of outliers is larger over all ratios in this experiment than the previous experiments. This is likely caused by over-fitting of the Fuga testing set, which is smaller in size than previous experiments.

MFTL CNNs at 330/100 have lower median pwpe and lower pwpe distribution than "benchmark" Fuga CNN, but their outliers exceed the pwpe of "benchmark" Fuga CNN. At other ratios, the MFTL CNNs have a wider and higher pwpe distribution than "benchmark" Fuga CNN. Thus, no MFTL CNNs outperform "benchmark" Fuga CNN in this experiment.

Consistent with previous experiments, structural similarity between LF model and Fuga does not affect pwpe – the median pwpe for all wake models at each ratio are similar: $2-3\%$ at 550/100, dropping to $1-2\%$ at 330/100, and rising back to $2-3\%$ at 110/100.

Also consistent with previous experiments, the choice of LF model impacts the range of pwpe. At 550/100, pwpe range of BGauss and TurboGauss is wider than other models. The box of NGauss at 110/100 is larger than other models. No relationship is found between LF model and pwpe range.

The choice of LF model affects pwpe outliers, consistent with previous experiments. At 550/100, TurboGauss and ZGauss have outliers up to 21%, lower than other models which have outliers up to $23 - 24\%$. SBGauss have the largest outlier at 550/100 beyond 24%. At 110/100, outliers of Jensen, NGauss, SBGauss and TurboJensen are larger than ZGauss, which has larger pwpe outlier than other models. No relationship is found between LF model and pwpe outliers.

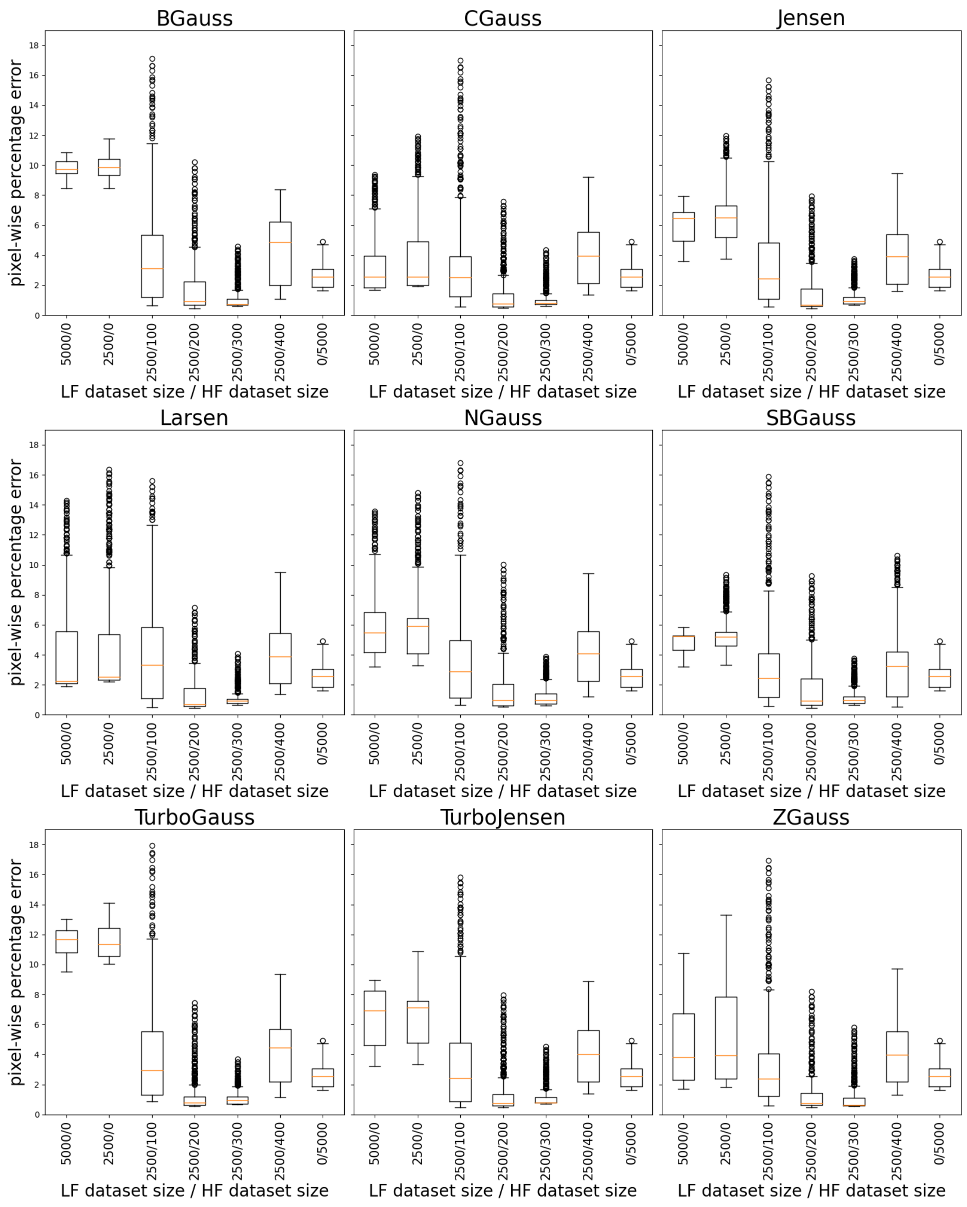


Figure 7: Box and whisker plots from experiment 1. HF model is Fuga wake model. In each subplot, the title shows the LF wake model, x-axis shows the LF to Fuga training set size, y-axis shows the computed pwpe from a testing set containing 500 Fuga model data. 5000/0 and 2500/0 on the x-axis mean the CNNs are trained using 5000 and 2500 LF model data respectively. 0/5000 on the x-axis means the CNN is the "benchmark" Fuga CNN trained using 5000 Fuga model data.

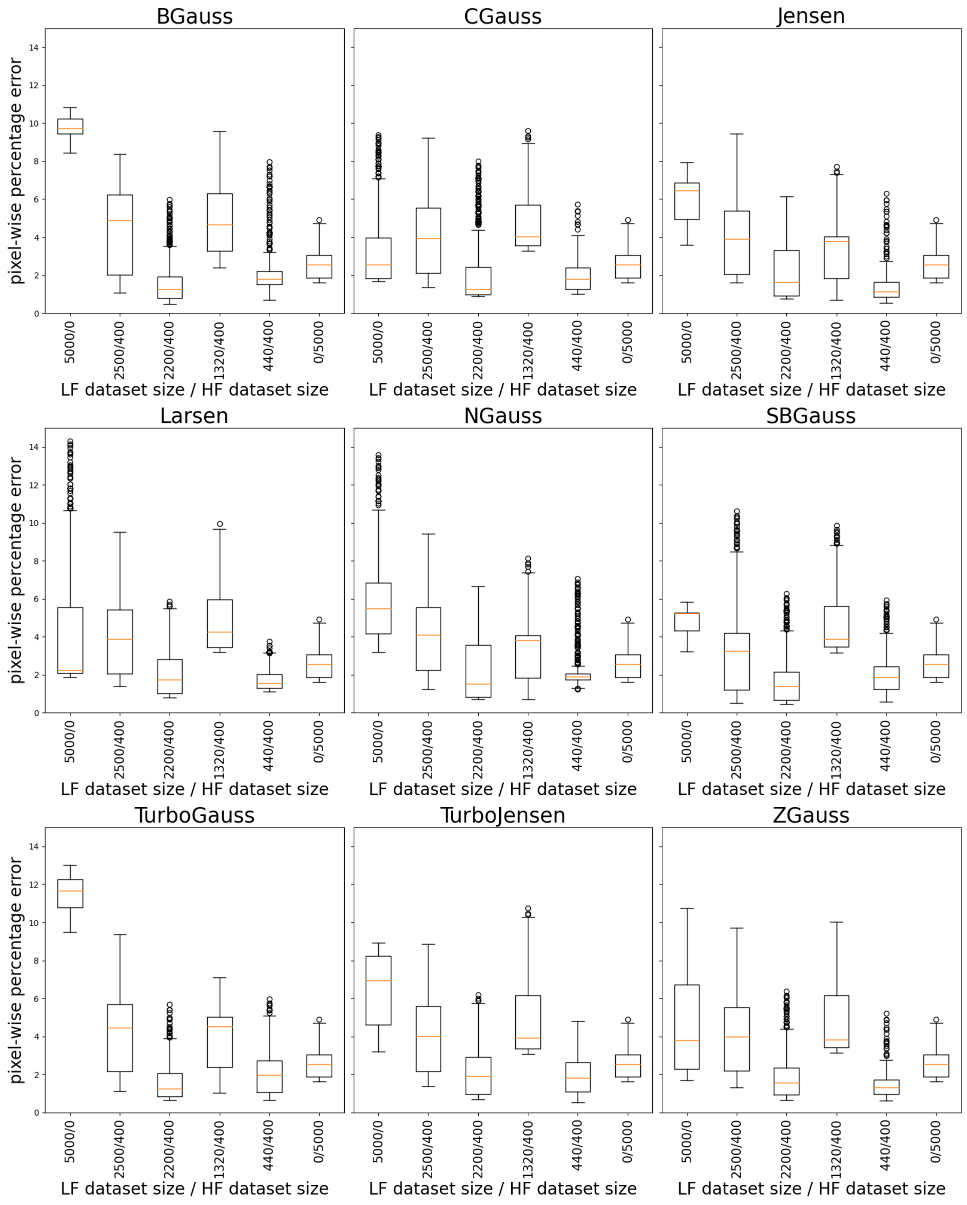


Figure 8: Box and whisker plots from experiment 2. HF model is Fuga wake model. In each subplot, the title shows the LF wake model, x-axis shows the LF to HF training set size, y-axis shows the computed pwpe from a testing set containing 500 Fuga model data. 5000/0 on the x-axis means the CNN is the "benchmark" LF CNN trained using 5000 LF model data. 0/5000 on the x-axis means the CNN is the "benchmark" HF CNN trained using 5000 Fuga model data. 2500/400 from experiment 1 is included for reference.

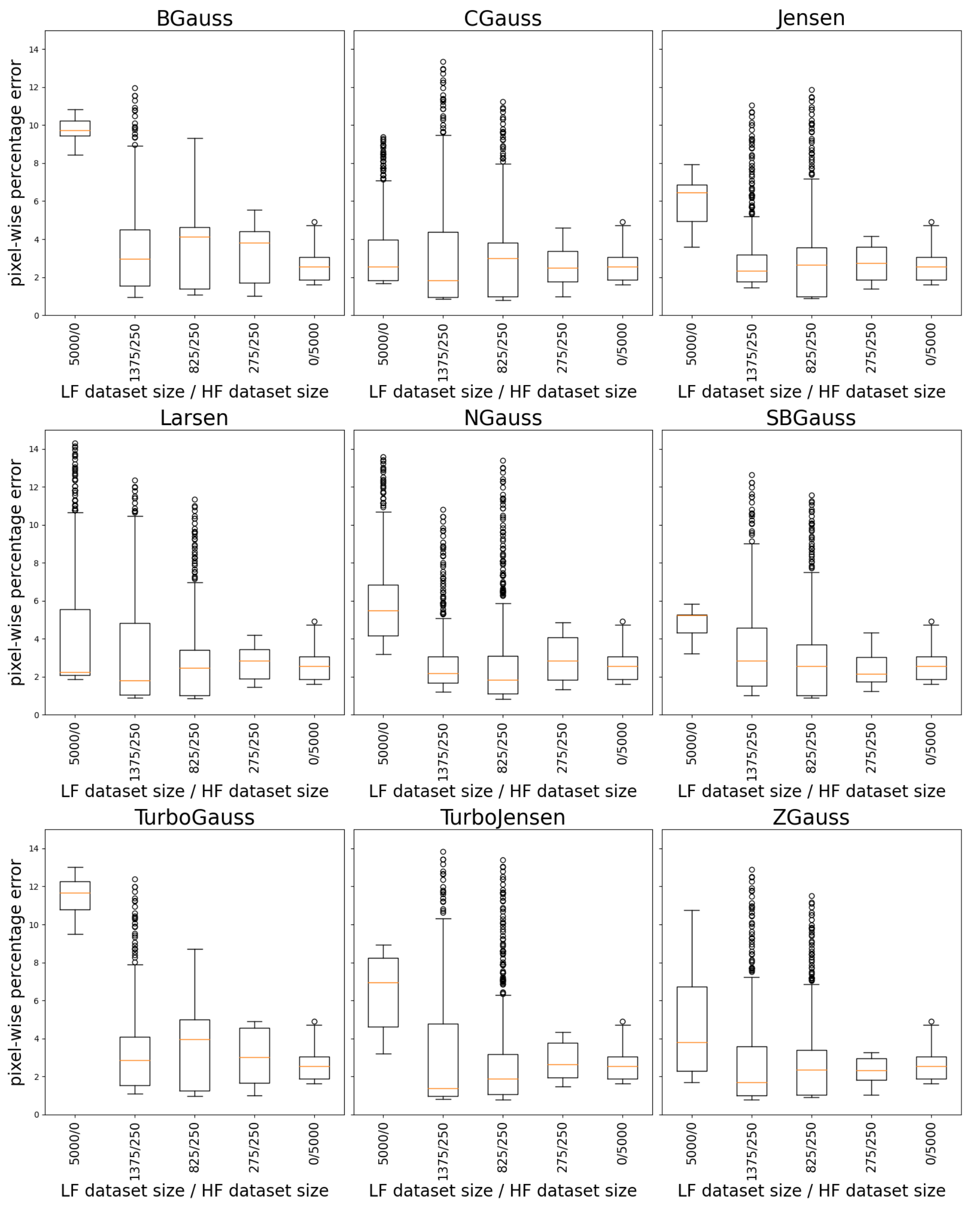


Figure 9: Box and whisker plots from experiment 3. HF model is Fuga wake model. In each subplot, the title shows the LF wake model, x-axis shows the LF to HF training set size, y-axis shows the computed pwpe from a testing set containing 500 Fuga model data. 5000/0 on the x-axis means the CNN is the "benchmark" LF CNN trained using 5000 LF model data. 0/5000 on the x-axis means the CNN is the "benchmark" HF CNN trained using 5000 Fuga model data.

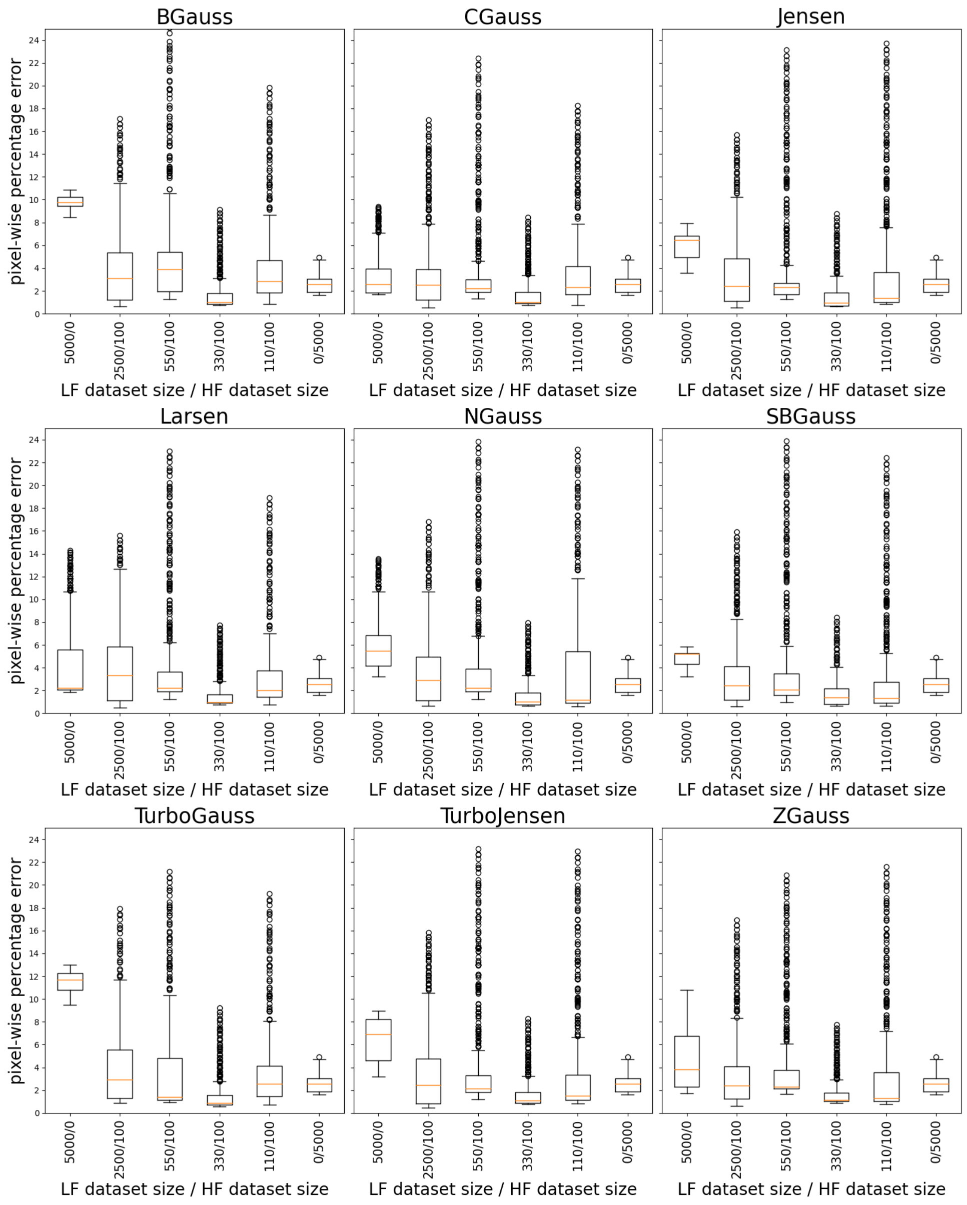


Figure 10: Box and whisker plots from experiment 4. HF model is Fuga wake model. In each subplot, the title shows the LF wake model, x-axis shows the LF to HF training set size, y-axis shows the computed pwpe from a testing set containing 500 Fuga model data. 5000/0 on the x-axis means the CNN is the "benchmark" LF CNN trained using 5000 LF model data. 0/5000 on the x-axis means the CNN is the "benchmark" HF CNN trained using 5000 Fuga model data. 2500/100 is from experiment 1 is included for reference.

6 Discussion

This project is the first implementation of MFTL workflow on a neural network to improve wake modelling. Results have shown that MFTL CNNs produced from this workflow can outperform "benchmark" CNN trained using 5000 Fuga data. This provides a starting point to use even higher fidelity model data from other packages for MFTL.

All PyWake wake models take $0.07 - 0.08$s on average to generate a simulation on M1 chip (section 3), hence savings in data generation budget, computational cost and simulation time in MFTL is negligible when using PyWake. However, these savings become more notable when a higher fidelity CFD solver based simulation package like SOWFA is used in MFTL.

Training time is notably reduced for MFTL CNNs compared to "benchmark" Fuga CNN. Training time of MFTL CNN at 2500/300 ($\sim$ 300s on Colab GPU) is 10% that of training "benchmark" Fuga CNN ($\sim$ 3000s on Colab GPU). Even if 30 epochs is used to train "benchmark" Fuga CNN ($\sim$ 1000s on Colab GPU), the training time of MFTL CNN is still 70% shorter.

MFTL CNNs performing worse at 2500/400 than at 2500/300 is counter-intuitive, since more HF data should lead to better performance for a fixed size of LF data. It is possible that this particular LF to HF training set size ratio leads to worse performance, since experiments 2-4 showed that more LF data relative to HF data does not necessarily lead to higher pwpe. Equivalently, more HF data relative to LF data does not necessarily lead to lower pwpe.

Generating testing set for 2500/400 and for 1320/400 using different random seeds does not change the performance of MFTL CNNs relative to "benchmark" Fuga CNN i.e. MFTL CNNs still perform worse than "benchmark" Fuga CNN, but pwpe distribution is lower for all CNNs compared to when random seed is 123 (see Appendix section 8.6 for more details). Thus, random seed has a significant impact on testing set generation, which in turn affects pwpe distribution, median pwpe, range of pwpe and outliers of pwpe. Changing the random seed to generate a wide range of testing sets would help spot anomalies and determine the average pwpe distribution, average median pwpe, average range of pwpe and average outliers of pwpe, hence better generalise and quantify the performance of MFTL CNNs.

The effects of the LF to HF training set size ratio on pwpe is inconclusive from this study, which could be further examined. The relationship between LF model, pwpe range and pwpe outliers is also inconclusive, and could be a topic for further research.

7 Conclusion

A multi-fidelity transfer learning (MFTL) workflow following [18] and [19] is implemented to a surrogate convolutional neural network (CNN) [20, 21], the first implementation of MFTL on neural network to improve wake modelling. Synthetic training, validation and testing sets are generated using PyWake package. Fuga model [38, 39] is used as high-fidelity (HF) model while other models are used as low-fidelity (LF) model.

Performance is quantified by computing the pixel-wise percentage error (pwpe). The implemented MFTL workflow produced CNNs that exceed the performance of Fuga trained CNN when LF to HF training set size ratio is 2500/300, no matter which LF model is used. However, at other ratios, whether a MFTL CNN outperforms "benchmark" Fuga CNN depend on the LF model. Additionally, random seed used to generate a testing set significantly affects the distribution, median, range and outliers of pwpe.

The choice of LF models does not affect median pwpe, but affects pwpe range and outliers. The relationship between LF model and median, range and outliers of pwpe is inconclusive from this study – LF model structurally similar to Fuga does not lead to lower pwpe.

Bibliography

[1] Rebecca Jane Barthelmie, K Hansen, Sten Tronæs Frandsen, Ole Rathmann, JG Schepers, W Schlez, J Phillips, K Rados, A Zervos, ESa Politis, et al. Modelling and measuring flow and wind turbine wakes in large wind farms offshore. *Wind Energy: An International Journal for Progress and Applications in Wind Power Conversion Technology*, 12(5):431–444, 2009.

[2] Nima Sedaghatizadeh, Maziar Arjomandi, Richard Kelso, Benjamin Cazzolato, and Mergen H Ghayesh. Modelling of wind turbine wake using large eddy simulation. *Renewable Energy*, 115:1166–1176, 2018.

[3] N O Jensen. Note on wind generator interaction, Nov 1983.

[4] Majid Bastankhah and Fernando Porté-Agel. A new analytical model for wind-turbine wakes. *Renewable Energy*, 70:116–123, 2014. Special issue on aerodynamics of offshore wind energy systems and wakes.

[5] Amin Niayifar and Fernando Porté-Agel. Analytical modeling of wind farms: A new approach for power prediction. *Energies*, 9(9):741, 2016.

[6] M Paul Van Der Laan, Niels N Sørensen, Pierre-Elouan Réthoré, Jakob Mann, Mark C Kelly, Niels Troldborg, J Gerard Schepers, and Ewan Machefaux. An improved k–ε model applied to a wind turbine wake in atmospheric turbulence. *Wind Energy*, 18(5):889–907, 2015.

[7] Mandar Tabib, Adil Rasheed, and Trond Kvamsdal. Les and rans simulation of onshore bessaker wind farm: analysing terrain and wake effects on wind farm performance. In *Journal of Physics: Conference Series*, volume 625, page 012032. IOP Publishing, 2015.

[8] Amina El Kasmi and Christian Masson. An extended k–ε model for turbulent flow through horizontal-axis wind turbines. *Journal of Wind Engineering and Industrial Aerodynamics*, 96(1):103–122, 2008.

[9] Yu-Ting Wu and Fernando Porté-Agel. Modeling turbine wakes and

power losses within a wind farm using les: An application to the horns rev offshore wind farm. *Renewable Energy*, 75:945–955, 2015.

[10] Fernando Porté-Agel, Yu-Ting Wu, Hao Lu, and Robert J Conzemius. Large-eddy simulation of atmospheric boundary layer flow through wind turbines and wind farms. *Journal of Wind Engineering and Industrial Aerodynamics*, 99(4):154–168, 2011.

[11] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[12] Tharindu P Miyanawala and Rajeev K Jaiman. An efficient deep learning technique for the navier-stokes equations: Application to unsteady wake flow dynamics. *arXiv preprint arXiv:1710.09099*, 2017.

[13] S Anagnostopoulos and MD Piggott. Offshore wind farm wake modelling using deep feed forward neural networks for active yaw control and layout optimisation. *Journal of Physics: Conference Series*, 2151(1):012011, jan 2022.

[14] Zilong Ti, Xiao Wei Deng, and Hongxing Yang. Wake modeling of wind turbines using machine learning. *Applied Energy*, 257:114025, 2020.

[15] Dimitris C Psichogios and Lyle H Ungar. A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, 38(10):1499–1511, 1992.

[16] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[17] Subhayan De, Jolene Britton, Matthew Reynolds, Ryan Skinner, Kenneth Jansen, and Alireza Doostan. On transfer learning of neural networks using bi-fidelity data for uncertainty propagation. *International Journal for Uncertainty Quantification*, 10(6), 2020.

[18] Su Jiang and Louis J Durlofsky. Use of multifidelity training data and transfer learning for efficient construction of subsurface flow surrogate models. *arXiv preprint arXiv:2204.11138*, 2022.

[19] Dong H Song and Daniel M Tartakovsky. Transfer learning on multi-fidelity data. *arXiv preprint arXiv:2105.00856*, 2021.

[20] Jens Bauer. Wind farm wake control using convolutional neural networks. *Independent Research Project, Imperial College London*, 2020.

[21] wakenet: Deep neural networks for wind farm wake modelling and optimisation. `https://github.com/soanagno/wakenet`. Accessed: 2022-06-28.

[22] Floris wake modeling and wind farm controls software. `https://github.com/NREL/floris`. Accessed: 2022-08-03.

[23] Pywake. `https://gitlab.windenergy.dtu.dk/TOPFARM/PyWake`. Accessed: 2022-08-03.

[24] Antonio Neiva, Vanessa Guedes, Caio Massa, and Daniel de Freitas. A review of wind turbine wake models for microscale wind park simulation. 10 2019.

[25] Nicolai Gayle Nygaard, Søren Trads Steen, Lina Poulsen, and Jesper Grønnegaard Pedersen. Modelling cluster wakes and wind farm blockage. In *Journal of Physics: Conference Series*, volume 1618, page 062072. IOP Publishing, 2020.

[26] Engineering windfarmmodels. `https://topfarm.pages.windenergy.dtu.dk/PyWake/notebooks/EngineeringWindFarmModels.html#Wake-deficit-models`. Accessed: 2022-08-16.

[27] Wake model description for optimization only case study. `https://github.com/byuflowlab/iea37-wflo-casestudies/blob/master/cs1-2/iea37-wakemodel.pdf`. Accessed: 2022-08-16.

[28] Carl R Shapiro, Dennice F Gayme, and Charles Meneveau. Modelling yawed wind turbine wakes: a lifting line approach. *Journal of Fluid Mechanics*, 841, 2018.

[29] Fernando Carbajo Fuertes, Corey D Markfort, and Fernando Porté-Agel. Wind turbine wake characterization with nacelle-mounted wind

lidars for analytical wake model validation. *Remote sensing*, 10(5):668, 2018.

[30] Turbopark. `https://github.com/OrstedRD/TurbOPark`. Accessed: 2022-08-17.

[31] Gunner C Larsen. A simple stationary semi-analytical wake model. 2009.

[32] Antonio Crespo, J Herna, et al. Turbulence characteristics in wind-turbine wakes. *Journal of wind engineering and industrial aerodynamics*, 61(1):71–85, 1996.

[33] Ángel Jiménez, Antonio Crespo, and Emilio Migoya. Application of a les technique to characterize the wake deflection of a wind turbine in yaw. *Wind energy*, 13(6):559–572, 2010.

[34] I Katic, Jørgen Højstrup, and Niels Otto Jensen. A simple model for cluster efficiency. In *European wind energy association conference and exhibition*, volume 1, pages 407–410. A. Raguzzi Rome, Italy, 1986.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[36] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Søren Ott, Jacob Berg, and Morten Nielsen. Linearised cfd models for wakes. 2011.

[39] Søren Ott and Morten Nielsen. Developments of the offshore wind turbine wake model fuga. *E-0046 Report*, 2014.

8 Appendix

8.1 Fuga

Steady-state RANS equation that ignores buoyancy and Coriolis force is written in terms of normalised pressure p at uniform density ρ ($p = P/\rho$), drag force field f_i and Reynolds stress tensor τ:

$$u_j \frac{\partial u_i}{\partial x_j} = \frac{\partial \tau_{ij}}{\partial x_j} - \frac{\partial p}{\partial x_i} + f_i + \nu \nabla^2 u_i \tag{10}$$

where u_j is a component of 3D flow field $\mathbf{u} \equiv (u, v, w)$ and x_j is a component of 3D spatial coordinates $\mathbf{x} \equiv (x, y, z)$.

τ is defined in terms of eddy viscosity K as

$$\tau_{ij} = K(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}) + \frac{1}{3}\delta_{ij}\tau_{kk} \tag{11}$$

RANS is coupled and solved together with the continuity equation

$$\frac{\partial u_j}{\partial x_j} = 0 \tag{12}$$

Using simple closure to specify K

$$K = \kappa u_* z \tag{13}$$

where κ is the von Karman constant and u_* is the friction (shear) velocity, RANS equation can be re-written as

$$u_j \frac{\partial u_i}{\partial x_j} = \frac{\partial}{\partial x_j} \kappa u_* z \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{\partial p}{\partial x_i} + \xi f_i \tag{14}$$

in which u and p are functions of the parameter ξ. After expanding the variables u, p and f as Taylor series to first order in ξ and substitute them into equation 14, Fourier transform is applied to the expanded equation 14 along two horizontal directions x, y. As a result, the flow field is a function of two wave numbers k_1, k_2 and the vertical direction z, and are represented in the following the coupled partial differential equations:

$$u^0 ik_1 u^1 + w^1 \frac{\partial u^0}{\partial z} = -\mathbf{k}^2 K u^1 + \frac{\partial}{\partial z} K \frac{\partial u^1}{z} + \frac{\partial K}{\partial z} ik_1 w^1 - ik_1 p^1 + f_1^1 \qquad (15)$$

$$u^0 ik_1 v^1 = -\mathbf{k}^2 K v^1 + \frac{\partial}{\partial z} K \frac{\partial v^1}{\partial z} + \frac{\partial K}{\partial z} ik_2 w^1 - ik_2 p^1 \qquad (16)$$

$$u^0 ik_1 w^1 = -\mathbf{k}^2 K w^1 + \frac{\partial}{\partial z} K \frac{\partial w^1}{\partial z} + \frac{\partial K}{\partial z} \frac{\partial w^1}{\partial z} - \frac{\partial p^1}{\partial z} \qquad (17)$$

$$ik_1 u^1 + ik_2 v^1 + \frac{\partial w^1}{\partial z} = 0 \qquad (18)$$

where $\mathbf{k}^2 = k_1^2 + k_2^2$. Equation 17 can be elimitated using equation 18. The following new variables are defined:

$$s = kz \qquad (19)$$

$$k_1 = k \cos \beta, \ k_2 = k \sin \beta \qquad (20)$$

$$\tilde{u} = \frac{u^1 u_* k}{f_{n,\mathbf{k}}}, \ \tilde{v} = \frac{v^1 u_* k}{f_{n,\mathbf{k}}}, \ \tilde{w} = \frac{w^1 u_* k}{f_{n,\mathbf{k}}} \qquad (21)$$

$$\tilde{q} = \frac{p^1 k}{f_{n,\mathbf{k}} s} \text{ when } s > 1, \text{ or } \tilde{p} = \frac{p^1}{k f_{n,\mathbf{k}}} \text{when } s < 1 \qquad (22)$$

where $f_{n,\mathbf{k}}$ contains information about thrust coefficient and turbine configuration. The independent variable of the coupled equations 15-18 is s when $s > 1$ and $t = \log \frac{z}{z_0}$ when $s < 1$. The variables $\tilde{q}, \tilde{p}$ and independent variables s, t switches at $s = 1 = kz_0$. z_0 is the surface roughness length.

The newly defined variables are substituted back into the coupled equations 15-18. Using first order expansion in Taylor series, the orthogonal chasing numerical method is applied to solve the coupled equations. Orthogonal chasing determines the variables from imposed boundary conditions through the use of dual variables and equations. The solution becomes a function of the two variables β, kz_0, and these two variables are the entries to the LUT of the Fuga model.

An extra entry is added to the LUT that improves the flexibility and accuracy of the Fuga wake model after an update in 2014. Eddy viscosity K is now stability dependent due to the stability dependent function $\phi_m(z/L)$:

$$K = \frac{u_* \kappa z}{\phi_m(z/L)} \tag{23}$$

where L is the Monin-Obukhov length scale from the Monin-Obukhov similarity theory. L is approximated in the first iteration from measured wind speed, current, air and water temperatures, and an estimated z_0. Wind speed U and potential temperature θ is calculated from the stability dependent function $\psi(\xi)$ where $\xi \equiv z/L$ is the non-dimensional length.

ξ determines stability: $\xi > 0$ is stable and $\xi < 0$ is unstable. $\xi_0 = z_0/L$ is the extra entry to the LUT, and is the starting point of the next iterations.

For more details, please see the Fuga documentations Ott, Berg and Nielsen (2011) [38] and Ott and Nielsen (2014) [39].

8.2 Benchmark Fuga CNN training figure

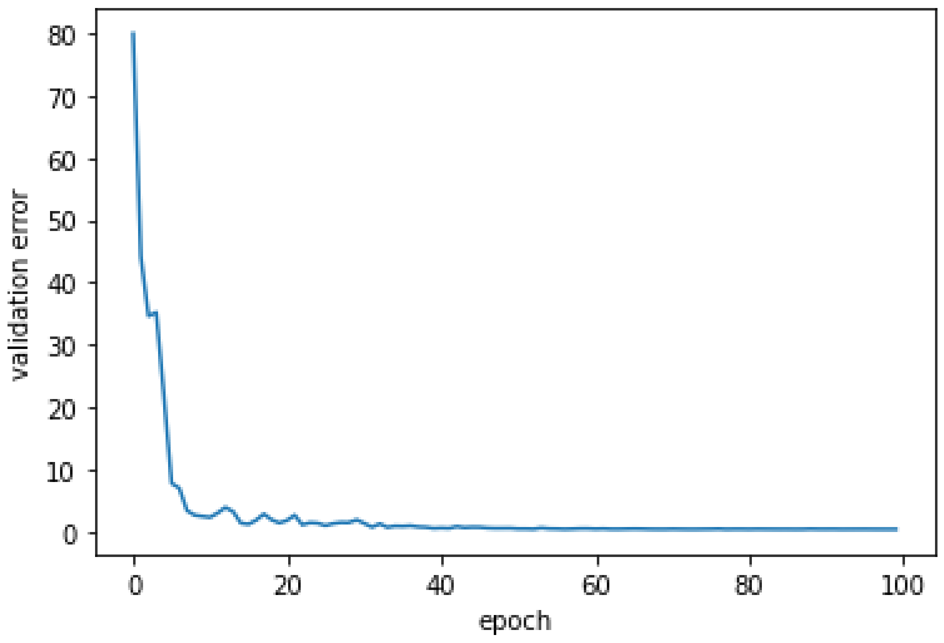
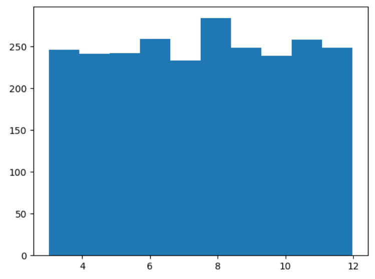
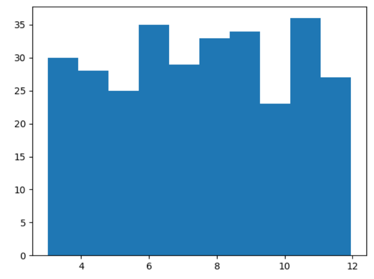


Figure 11: Validation error (pixel-wise percentage error (pwpe)) over epochs when training "benchmark" Fuga CNN. pwpe remains low after roughly 10 epochs, then stabalises after around 20 epochs.

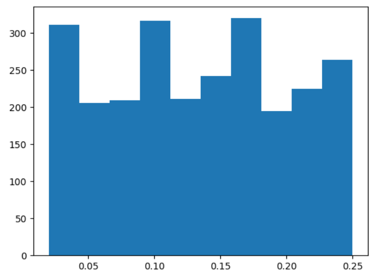
8.3 Split training set



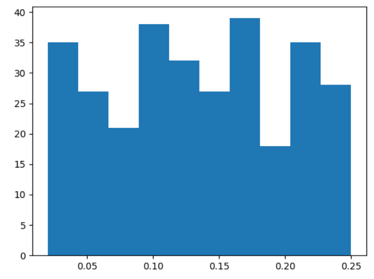
(a) Distribution of 2500 wind speeds in training set.



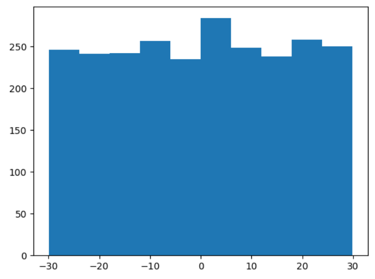
(b) Distribution of 300 wind speeds in new training set.



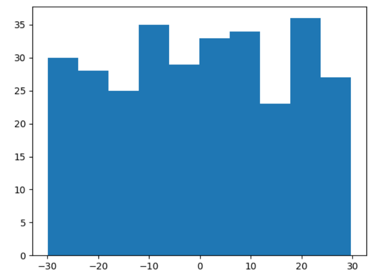
(c) Distribution of 2500 TI in training set.



(d) Distribution of 300 TI in new training set.



(e) Distribution of 2500 yaw angles in training set.



(f) Distribution of 300 yaw angles in new training set.

Figure 12: Original training set size is 2500. The uniformly distributed lists of wind speed, TI and yaw angle are reduced to size 300 using Stratified-ShuffleSplit to maintain the uniform distribution of values for wind speed, TI and yaw angle.

8.4 Data generation times

Model	10	20	30	40	50	60	70	80	90	100	Mean
BGauss	7.34	7.99	7.24	7.60	7.56	7.26	7.66	7.42	8.07	8.88	7.70
CGauss	7.40	8.01	7.20	7.62	7.54	7.23	7.77	7.44	8.05	8.67	7.69
Jensen	7.43	7.95	7.23	7.75	7.56	7.22	7.69	7.42	8.03	8.63	7.69
Larsen	7.32	7.98	7.28	7.64	7.52	7.25	7.70	7.39	8.07	8.72	7.69
NGauss	7.23	8.02	7.23	7.71	7.55	7.24	7.70	7.39	8.36	8.98	7.74
SBGauss	7.24	7.99	7.24	7.60	7.54	7.26	7.67	7.41	8.07	9.00	7.70
TurboGauss	7.69	7.98	7.34	7.76	7.88	7.41	7.65	7.47	8.08	8.87	7.81
TurboJensen	7.43	8.03	7.20	7.64	7.55	7.24	7.71	7.44	8.04	8.53	7.68
ZGauss	7.41	7.96	7.26	7.65	7.54	7.24	7.69	7.44	8.06	8.48	7.67
Fuga	7.30	7.97	7.26	7.69	7.52	7.29	7.74	7.45	8.07	8.53	7.68

Table 1: Times in seconds needed to generate 100 simulations for each wake model at different random seeds. Column entries $10, 20 \ldots, 100$ show the random seed used to generate 100 uniformly distributed inflow conditions. Simulations are run on the Apple M1 chip.

8.5 Simulation example: 2500/300

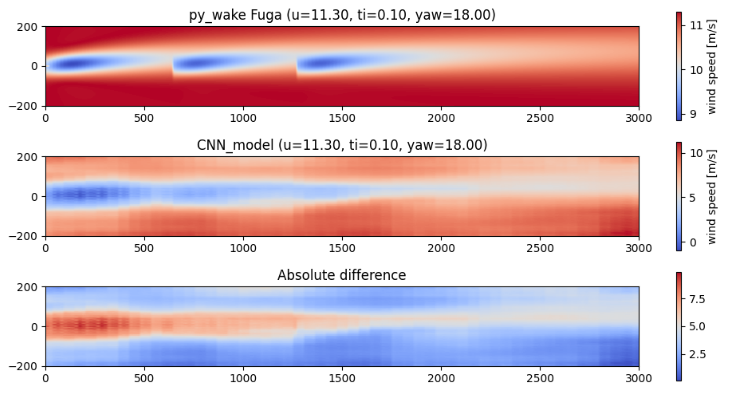


Figure 13: Flow field from a CNN first trained with 2500 Jensen data, then re-trained with 300 Fuga data. It is compared with flow field generated using Fuga. The absolute difference between the two is also shown. 1×3 turbine configuration is used, with turbines located at $(0, 0), (630, 0)$, and $(1260, 0)$

8.6 Different random seeds

8.6.1 Experiment 1: 2500/400

Changing random seed from 123 to 100 and 10 for testing set generation, pwpe distribution is lower in the latter two cases than the first case, as shown in figures 14 - 16.

8.6.2 Experiment 2: 1320/400

Changing random seed from 123 to 100 and 10 for testing set generation, pwpe distribution is lower in the latter two cases than the first case, as shown in figures 17 - 19.

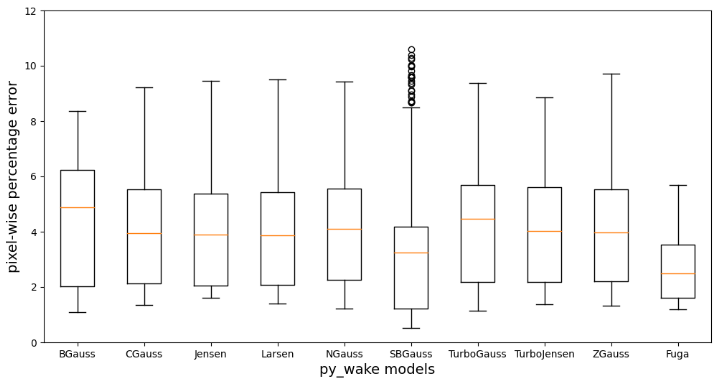


Figure 14: Box and whisker plots for MFTL CNNs from experiment 1 when random seed is 123. "Benchmark" Fuga CNN is included as reference.

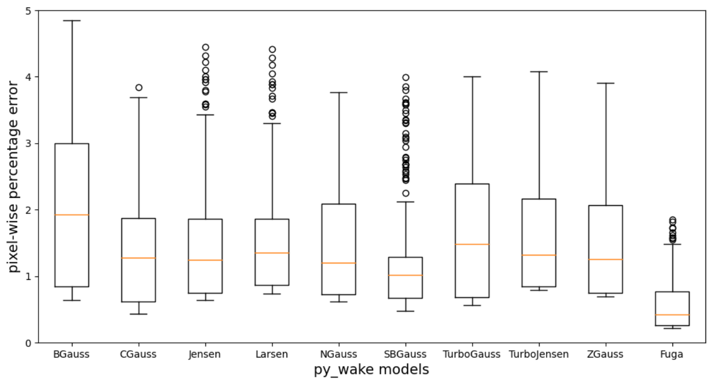


Figure 15: Box and whisker plots for MFTL CNNs from experiment 1 when random seed is 100. "Benchmark" Fuga CNN is included as reference. pwpe distribution for all CNNs are lower than when random seed is 123 (figure 14).

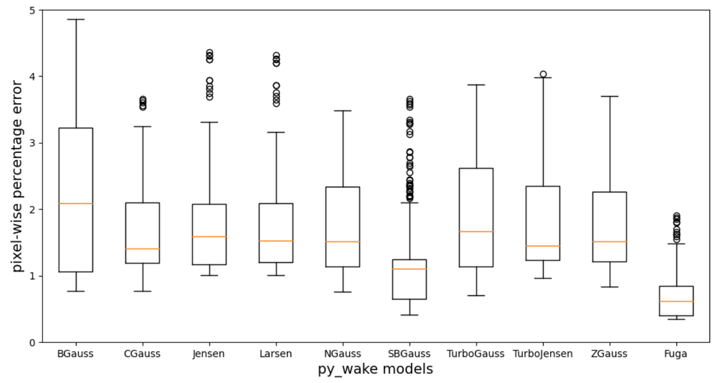


Figure 16: Box and whisker plots for MFTL CNNs from experiment 1 when random seed is 10. "Benchmark" Fuga CNN is included as reference. pwpe distribution for all CNNs are lower than when random seed is 123 (figure 14).

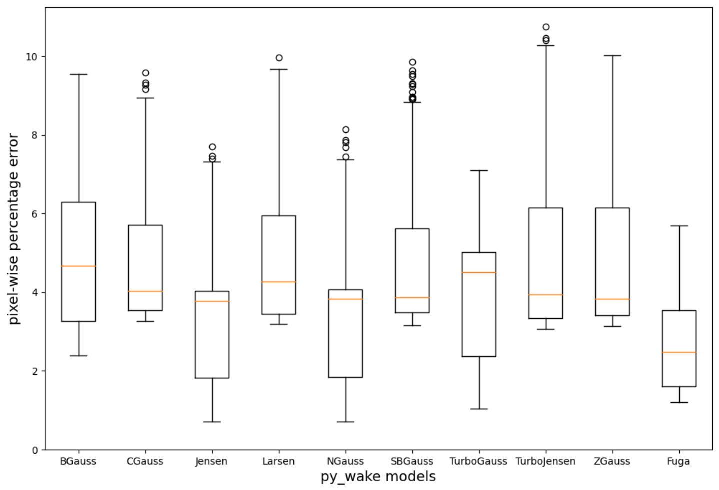


Figure 17: Box and whisker plots for MFTL CNNs from experiment 2 when random seed is 123. "Benchmark" Fuga CNN is included as reference.

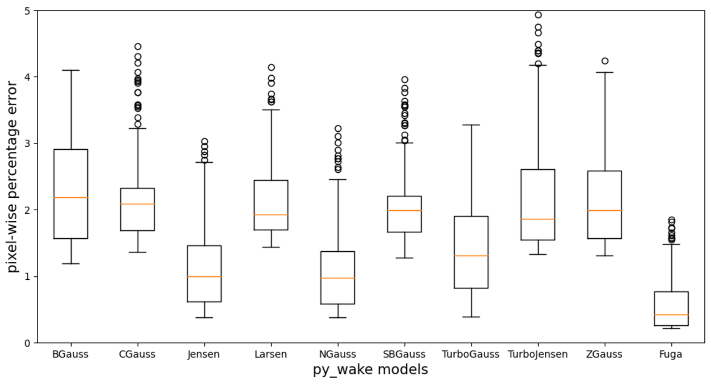


Figure 18: Box and whisker plots for MFTL CNNs from experiment 2 when random seed is 100. "Benchmark" Fuga CNN is included as reference. pwpe distribution for all CNNs are lower than when random seed is 123 (figure 17).

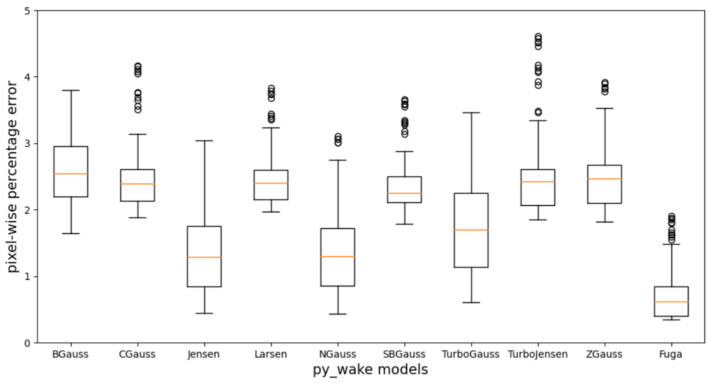


Figure 19: Box and whisker plots for MFTL CNNs from experiment 2 when random seed is 10. "Benchmark" Fuga CNN is included as reference. pwpe distribution for all CNNs are lower than when random seed is 123 (figure 17).