Romit Maulik
Argonne National Laboratory
94 PUBLICATIONS **1,336** CITATIONS
SEE PROFILE

Saumil Sudhir Patel
Argonne National Laboratory
19 PUBLICATIONS **97** CITATIONS
SEE PROFILE

Bethany Lusch
Argonne National Laboratory
35 PUBLICATIONS **1,026** CITATIONS
SEE PROFILE

Elise Jennings
Argonne National Laboratory
25 PUBLICATIONS **410** CITATIONS
SEE PROFILE

Deploying deep learning in OpenFOAM with TensorFlow

Romit Maulik*, Himanshu Sharma†
Argonne Leadership Computing Facility, Argonne National Laboratory, Lemont IL-60439, USA

Saumil Patel
Computational Science Division, Argonne National Laboratory, Lemont, IL-60439, USA

Bethany Lusch, Elise Jennings ‡
Argonne Leadership Computing Facility, Argonne National Laboratory, Lemont IL-60439, USA

We outline the development of a data science module within OpenFOAM which allows for the in-situ deployment of trained deep learning architectures for general-purpose predictive tasks. This module is constructed with the TensorFlow C API and is integrated into OpenFOAM as an application that may be linked at run time. Notably, our formulation precludes any restrictions related to the *type* of neural network architecture (i.e., convolutional, fully-connected, etc.). This allows for potential studies of complicated neural architectures for practical CFD problems. In addition, the proposed module outlines a path towards an open-source, unified and transparent framework for computational fluid dynamics and machine learning.

I. Nomenclature

C_s	=	Dynamic Smagorinsky coefficient
h	=	Backward facing step-height
ν_t	=	Turbulent eddy-viscosity
Re_τ	=	Friction Reynolds number
R_{ij}	=	Reynolds stress tensor

II. Introduction

In recent times, physics-informed machine learning algorithms have generated a lot of interest for computational fluid dynamics (CFD) applications. These algorithms have been applied for wide variety of tasks such as closure modeling [1–13], control [14–18], surrogate modeling [19–32], inverse problems [33–37], uncertainty quantification [38–40], data assimilation [35, 41–43] and super-resolution [44, 45]. These studies have demonstrated that the ability of modern machine learning algorithms to learn complicated nonlinear relationships may be leveraged for improving accuracy of quantities of interest as well as significant reductions in computational costs. Indeed, these studies have generated exciting results in a wide range of research thrusts within computational physics such as for novel turbulence models, shock capturing methods, shape optimization, adaptive mesh refinement strategies, interface tracking algorithms. Exhaustive reviews of machine learning methods for fluid dynamics can be found in Brunton et al. [46] and Fukami et al. [47], and a review of machine learning opportunities for turbulence may be found in Duraisamy et al. [48]

Our present study is motivated by this explosion of data-driven algorithm development. Specifically, we seek to address the lack of a coherent framework for reproducible data-driven algorithm development and testing. Our goal, for this study, is to propose an in-situ coupling mechanism between data science and simulation that may be utilized for a large variety of simulation tasks as well as machine learning strategies. Crucially, we also wish to build on an well-established CFD package that has been tested for a wide variety of problems deployed both serially and in parallel. This would allow for the easy integration of machine learning into a previously established meshing, simulation and visualization assembly line.

*AIAA Member.
†Now at Pacific Northwest National Laboratory, Richland WA-99354, USA
‡Now at Irish Centre for High-End Computing at NUI Galway

We choose OpenFOAM [49], an open-source general-purpose CFD software under active development, as our simulation framework. For our data-science capability, we choose TensorFlow 1.15 [50], a well-known machine learning library that allows for the development of data-driven techniques as simple as linear, logistic and polynomial regression or as complicated as fully connected and convolutional neural networks, regular and variational autoencoders and generative adversarial networks. This article serves as an introduction as well as a tutorial to the proposed coupling mechanism. The coupling is introduced by way of a surrogate prediction task for the Reynolds-averaged Navier-Stokes (RANS) turbulence eddy viscosity for a canonical backward facing step problem (with greater detail available in [51]). We also demonstrate its viability for competitive compute times by deploying a deep learning surrogate to the dynamic Smagorinsky model [52] for a turbulent channel flow at $Re_\tau = 180$. We shall introduce our coupling mechanism through a step by step demonstration that can be implemented by an OpenFOAM user independently. In addition, all of our source code and data are available publicly [*].

III. Literature review

Despite the massive popularity of machine learning algorithm development for various tasks, there have been few open-source frameworks that have successfully allowed for direct embedding of general-purpose machine learning algorithms within simulation codes in an easy-to-reproduce manner. One such example is that of Geneva and Zabaras [39] who embed a neural network model from PyTorch into OpenFOAM 4.1. However, this procedure requires the installation of additional software packages such as ONNX and Caffe2 that may cause issues with dependencies. In addition, Caffe2 is deprecated (to be subsumed into PyTorch) and future incorporation of PyTorch models into OpenFOAM through this route is unclear. Another framework under active development is the Fortran-Keras Bridge [53] that successfully couples densely connected neural networks to Fortran simulation codes. However, this framework is yet to support more complicated architectures such as convolutional neural networks and development revolves around the programming of neural network subroutines in Fortran before a Keras model can be imported. In contrast, we utilize TensorFlow to represent a generic deep learning architecture as a graph on disk which is imported into OpenFOAM during runtime.

IV. The coupling mechanism

In the following section, we outline the procedure for using the TensorFlow C API in OpenFOAM. In addition we detail how to export a trained deep learning model from TensorFlow in Python to disk as a protobuffer (where all of the trainable parameters and operations of the deep learning framework are fixed). Finally, we outline how to load this protobuffer during the solution process and interface it with OpenFOAM data structures.

A. Exporting model from Python

A generic TensorFlow model may be defined in Python by using the Keras API. A simple fully-connected network is defined as follows:

```
from tensorflow import keras
def get_model(num_inputs,num_outputs,num_layers,num_neurons):
    '''
    num_inputs  : Number of model inputs
    num_outputs : Number of model outputs
    num_layers  : Number of hidden layers
    num_neurons : Number of neurons in hidden layers
    Returns: TensorFlow model
    '''

    # Input layer
    ph_input = keras.Input(shape=(num_inputs,),name='input_placeholder')

    # Hidden layers
    hidden_layer = keras.layers.Dense(num_neurons,activation='tanh')(ph_input)

    for layer in range(num_layers):
        hidden_layer = keras.layers.Dense(num_neurons,activation='tanh')(hidden_layer)
```

[*]https://github.com/argonne-lcf/TensorFlowFoam

```
    # Output layer
    output = keras.layers.Dense(num_outputs,activation='linear',name='output_value')(hidden_layer
    )

    model = keras.Model(inputs=[ph_input],outputs=[output])

    # Optimizer
    my_adam = keras.optimizers.Adam(lr=0.001, decay=0.0)

    # Compilation
    model.compile(optimizer=my_adam,loss={'output_value': 'mean_squared_error'})

    return model
```

Listing 1   Fully connected network definition using the Keras API of TensorFlow 1.15 in Python 3.6.8.

The abstractions of this API allow for significant complexity in model development. After this model has been defined and trained, the first step for exporting the model to a non-pythonic environment requires a function to freeze the model weights. This is achieved through the following method:

```
def freeze_session(session, keep_var_names=None, output_names=None, clear_devices=True):
    """
    Freezes the state of a session into a pruned computation graph. Creates a new computation
    graph where variable nodes are replaced by constants taking their current value in the
    session.
    session: The TensorFlow session to be frozen.
    keep_var_names: A list of variable names that should not be frozen, or None to freeze all the
     variables in the graph.
    output_names: Names of the relevant graph outputs.
    clear_devices: Remove the device directives from the graph for better portability.
    Returns: The frozen graph definition.
    """
    graph = session.graph
    with graph.as_default():
        freeze_var_names = list(set(v.op.name for v in tf.global_variables()).difference(
    keep_var_names or []))
        output_names = output_names or []
        output_names += [v.op.name for v in tf.global_variables()]
        input_graph_def = graph.as_graph_def()
        if clear_devices:
            for node in input_graph_def.node:
                node.device = ""
        frozen_graph = tf.graph_util.convert_variables_to_constants(
            session, input_graph_def, output_names, freeze_var_names)
        return frozen_graph

# Save the graph to disk
frozen_graph = freeze_session(keras.backend.get_session(),output_names=[out.op.name for out in
    model.outputs])
tf.train.write_graph(frozen_graph, './', 'ML_Model.pb', as_text=False)
```

Listing 2   Freezing trained neural network model to protobuffer format.

The model (in the protobuffer format) can now be imported using the C API.

B. The TensorFlow C API

Next, we present the procedure to call the TensorFlow C API for loading a graph and performing an inference within a general C++ code. This is accomplished as follows:

```
// TensorFlow C API header
#include <tensorflow/c/c_api.h>
// Function to load a graph from protobuffer
TF_Graph* LoadGraph(const char* graphPath) {
  if (graphPath == nullptr) {
    return nullptr;
  }
  TF_Buffer* buffer = ReadBufferFromFile(graphPath);
  if (buffer == nullptr) {
    return nullptr;
  }

  TF_Graph* graph = TF_NewGraph();
  TF_Status* status = TF_NewStatus();
  TF_ImportGraphDefOptions* opts = TF_NewImportGraphDefOptions();

  TF_GraphImportGraphDef(graph, buffer, opts, status);
  TF_DeleteImportGraphDefOptions(opts);
  TF_DeleteBuffer(buffer);

  if (TF_GetCode(status) != TF_OK) {
    TF_DeleteGraph(graph);
    graph = nullptr;
  }

  TF_DeleteStatus(status);

  return graph;
}

// Load graph from disk
graph_ = LoadGraph("./ML_Model.pb");

// Input operation
input_ph_ = {TF_GraphOperationByName(graph_, "input_placeholder"), 0};

// Output operation
output_ = {TF_GraphOperationByName(graph_, "output_value/BiasAdd"), 0};
```

Listing 3   Loading model saved in protobuffer

Note how the names of the operations defined in the Python model are required to identify operations in C++. An inference using this loaded graph may be performed using

```
TF_Tensor* CreateTensor(TF_DataType data_type,
                        const std::int64_t* dims, std::size_t num_dims,
                        const void* data, std::size_t len) {
  if (dims == nullptr) {
    return nullptr;
  }

  TF_Tensor* tensor = TF_AllocateTensor(data_type, dims, static_cast<int>(num_dims), len);
  if (tensor == nullptr) {
    return nullptr;
  }

  void* tensor_data = TF_TensorData(tensor);
  if (tensor_data == nullptr) {
    TF_DeleteTensor(tensor);
    return nullptr;
  }

  if (data != nullptr) {
    std::memcpy(tensor_data, data, std::min(len, TF_TensorByteSize(tensor)));
  }

  return tensor;
}

void DeleteTensor(TF_Tensor* tensor) {
  if (tensor != nullptr) {
    TF_DeleteTensor(tensor);
  }
}

void DeleteSession(TF_Session* session) {
  TF_Status* status = TF_NewStatus();
  TF_CloseSession(session, status);
  if (TF_GetCode(status) != TF_OK) {
    TF_CloseSession(session, status);
  }
  TF_DeleteSession(session, status);
  if (TF_GetCode(status) != TF_OK) {
    TF_DeleteSession(session, status);
  }
  TF_DeleteStatus(status);
}

TF_Status* status_ = TF_NewStatus();
TF_SessionOptions* options_ = TF_NewSessionOptions();
TF_Session* sess_ = TF_NewSession(graph_, options_, status_);

TF_Tensor* output_tensor_ = nullptr;
TF_Tensor* input_tensor_ = CreateTensor(TF_FLOAT,
                                      input_dims.data(), input_dims.size(),
                                      &input_vals, num_cells*num_inputs*sizeof(float));

// Arrays of tensors
TF_Tensor* input_tensors_[1] = {input_tensor_};
TF_Tensor* output_tensors_[1] = {output_tensor_};
// Arrays of operations
TF_Output inputs[1] = {input_ph_};
TF_Output outputs[1] = {output_};

TF_SessionRun(sess_,
            nullptr, // Run options.
            inputs, input_tensors_, 1, // Input tensor ops, input tensor values, number of inputs
    .
            outputs, output_tensors_, 1, // Output tensor ops, output tensor values, number of
    outputs.
            nullptr, 0, // Target operations, number of targets.
            nullptr, // Run metadata.
            status_ // Output status.
            );

// Cast output from TF C API to float*
const auto data = static_cast<float*>(TF_TensorData(output_tensors_[0]));

DeleteTensor(input_tensor_);
DeleteTensor(output_tensor_);
TF_DeleteSessionOptions(options_);
TF_DeleteStatus(status_);
DeleteSession(sess_);
```

Listing 4 Inference using TF C API in C++

where `input_vals` and `input_dims` are the data and the dimensions, respectively, for the input data.

C. Compiling an OpenFOAM turbulence model that calls TensorFlow

Once we have established how to deploy a trained machine learning model in C++, we may call for an inference within OpenFOAM. For the purpose of demonstration, we show how a new turbulence model library may be compiled while linking to the TensorFlow C Libraries. To start, we must give OpenFOAM the path to the TensorFlow C API in the `Make/Options` file as follows

```
EXE_INC = \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I/path/to/tensorflow/include \
LIB_LIBS = \
    -L/path/to/tensorflow/lib \
    -lincompressibleTransportModels \
    -lturbulenceModels \
    -lfiniteVolume \
    -lmeshTools \
    -ltensorflow \
    -lstdc++
```

Listing 5 Make/Options for OpenFOAM and TensorFlow linkage

One may now compile a new turbulence model using `wmake`.

V. Demonstration

In the following section, we shall outline some results from deploying deep learning inference within OpenFOAM 5.0 simulations for canonical turbulence problems. We shall also demonstrate the potential benefits of using ML within scientific computing with results that exhibit reduced time to solution. We note that all experiments within this section were carried out on an 8th-generation Intel CoreI7 processor with a clockspeed of approximately 1.90GHz and on an Ubuntu 18.04 operating system.

A. Science driver 1: Surrogate modeling for steady-state turbulent eddy-viscosity

In this section, we demonstrate how the direct prediction of the steady-state turbulent eddy-viscosity (as a function of initial conditions) can accelerate RANS simulations. Our framework is outlined in greater detail in [51]. Briefly, the initial conditions for a RANS simulation (coming from a low-fidelity potential solution) on an arbitrary mesh can be used to predict an approximation to the steady-state Spalart-Allmaras turbulent eddy-viscosity [54] via a neural network. Our training data is obtained by performing multiple RANS simulations using the Spalart-Allmaras model on different geometries. In this experiment, the different geometries are all backward facing steps with varying step heights (h).

Once trained, the steady-state eddy-viscosity emulator may be used at the start of the simulation (by observing the initial conditions) following which solely the pressure and velocity equations need to be solved to convergence. We outline results from one such experiment, where the geometry is 'unseen', in Figure 1. The time-to-solution of the proposed framework is significantly reduced, albeit at the cost of some increase in error, as shown in Figure 2. Potential extensions to such a framework include trying to bypass high fidelity turbulence models (two-equation) or generating models from temporally averaged DNS data.

B. Science driver 2: Surrogate modeling of dynamic Smagorinsky coefficient

In this section, we show how a deep learning framework to predict the Smagorinsky coefficient (dynamically) may be used within OpenFOAM. Our test case is given by a turbulent channel flow at $\mathrm{Re}_\tau = 395$. For a proof of concept, we use our data-driven model to predict the dynamic Smagorinsky coefficient C_s, given instantaneous measurements of the strain-rate tensor as well as the three velocity components at each cell center of the mesh. For preliminary results, our training data is generated from Dynamic Smagorinsky itself (i.e., with a standard scale-similarity based least-squares calculation following test-filtering). The goal, for this exercise, is to see if a deep neural network, despite a far greater number of floating point operations, can efficiently bypass the dynamic Smagorinsky framework. Success in this regard could lead to the development of closures from data obtained on higher fidelity meshes as well as from DNS while retaining efficient inference speeds on the coarser meshes.

We show a preliminary analysis in Figure 3 for the ensemble-averaged Reynolds stresses given by

$$R_{ij} = \langle u_i' u_j' \rangle - \langle u_i' \rangle \langle u_j' \rangle \tag{1}$$

where the angled-brackets imply time-averaging and $u_i^{'} = u_i - \langle u_i \rangle$ is the temporal fluctuation. It is observed that the ML surrogate recovers performance similar to Dynamic Smagorinsky without the use of test-filtering. The compute times to solution for both cases on one node (identical to the node used for the RANS experiments previously) were 1348 seconds for Dynamic Smagorinsky and 1297 seconds for the ML surrogate assuming a physical solution time of 1000 seconds. This is promising since the neural network requires far more floating point operations than the standard dynamic Smagorinsky coefficient calculation.

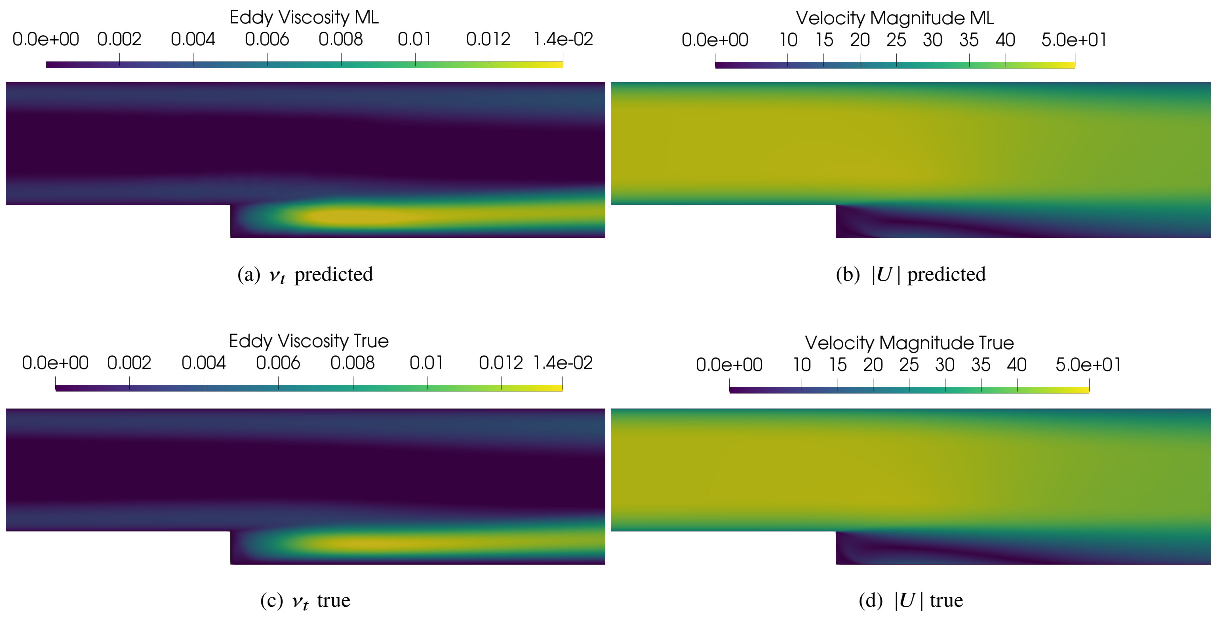


Fig. 1 Contour plots for a backward facing step. Note that the training of the ML surrogate did not include data for the shown step height.

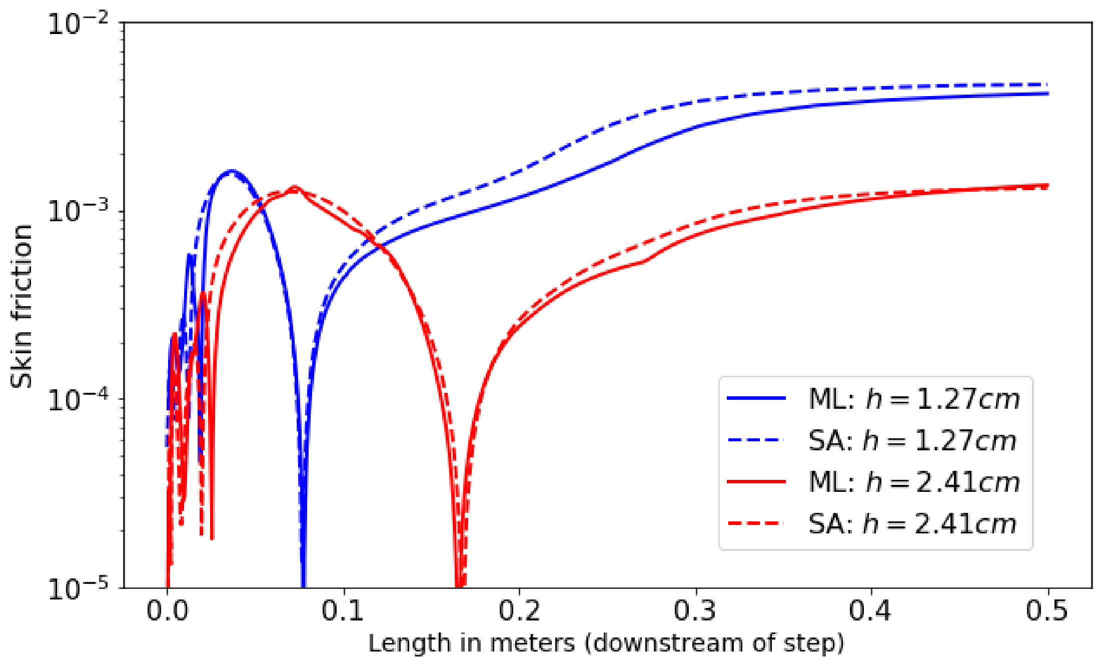
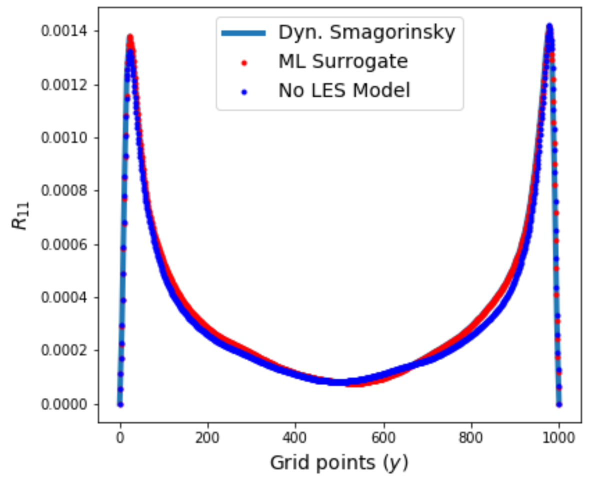
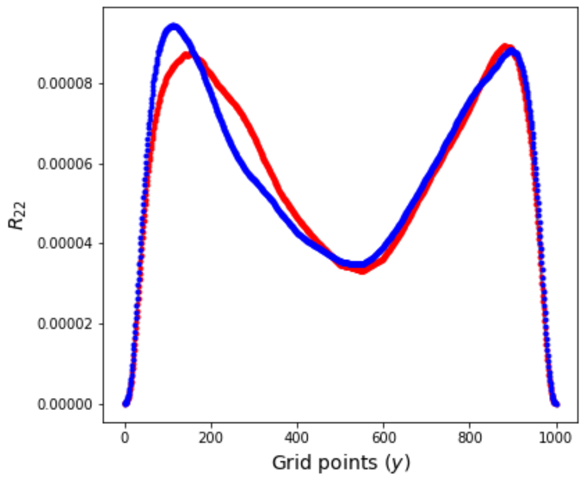


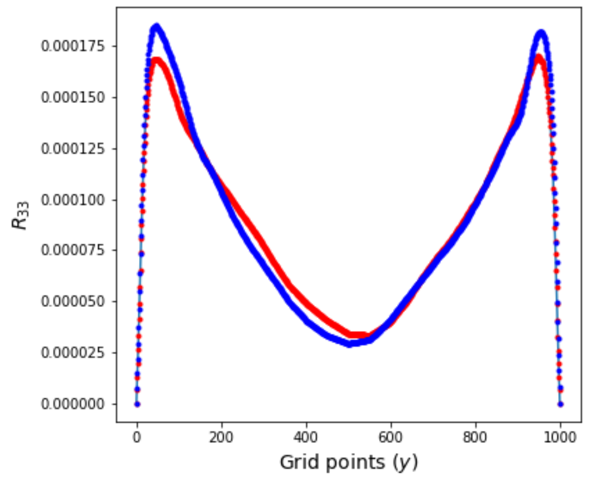
Fig. 2 Skin-friction coefficient predictions downstream of the step when deploying the ML framework on two test step heights. The ML framework is seen to introduce some errors. Downstream skin friction inaccuracies suggest adaptive sampling may be necessary for improved generalization across geometries. However, the reattachment point is recovered well.



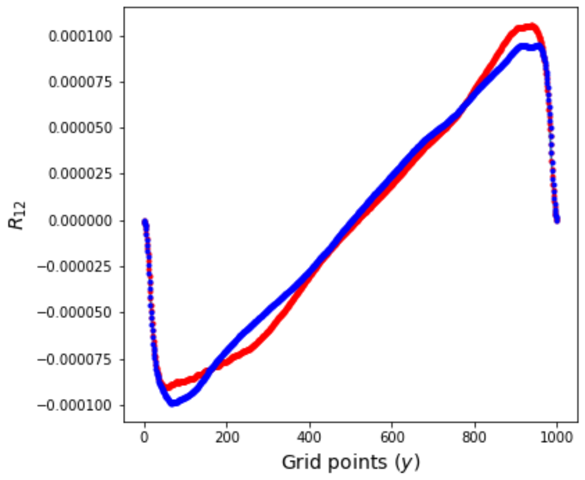
(a) R_{11}



(b) R_{22}



(c) R_{33}



(d) R_{12}

Fig. 3 Ensemble averaged Reynolds stresses for channel flow at $Re_\tau = 395$, where $R_{ij} = \langle u_i' u_j' \rangle - \langle u_i' \rangle \langle u_j' \rangle$. It is seen that the the surrogate model using a neural network for Smagorinsky coefficient calculation is able to recreate the Dynamic Smagorinsky solution with comparable times to solution. The "No LES Model" curve indicates a computation without the use of any turbulence modeling and therefore causes deviation from the that obtained by Dynamic Smagorinsky.

VI. Conclusion and Future Work

Ongoing work related to the content introduced in this article is focused on the assessment of ML inference at scale. It is important to assess how in-situ ML inference frameworks may affect CFD workflows optimized for distributed parallelism. In addition, we have also focused our efforts on developing a viable 'training-online' strategy (also within OpenFOAM) where a neural architecture may be trained while a simulation is actually running. As stated previously, the presence of these capabilities in a common framework will allow for greater transparency and reproducibility of data-driven algorithms for scientific computing.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research was funded in part and used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This paper describes objective technical results and analysis. HS acknowledges support from the ALCF Director's Discretionary (DD) program for CFDML project. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. DOE or the United States Government.

References

[1] Beck, A., Flad, D., and Munz, C.-D., "Deep neural networks for data-driven LES closure models," *Journal of Computational Physics*, Vol. 398, 2019, p. 108910.

[2] Maulik, R., San, O., Jacob, J. D., and Crick, C., "Sub-grid scale model classification and blending through deep learning," *J. Fluid Mech.*, Vol. 870, 2019, pp. 784–812.

[3] Singh, A. P., Medida, S., and Duraisamy, K., "Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils," *AIAA Journal*, 2017, pp. 2215–2227.

[4] Gamahara, M., and Hattori, Y., "Searching for turbulence models by artificial neural network," *Phys. Rev. Fl.*, Vol. 2, No. 5, 2017, p. 054604.

[5] Ling, J., Kurzawski, A., and Templeton, J., "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *J. Fluid Mech.*, Vol. 807, 2016, pp. 155–166.

[6] Ling, J., and Templeton, J., "Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier Stokes uncertainty," *Phys. Fluids*, Vol. 27, No. 8, 2015, p. 085103.

[7] Matai, R., and Durbin, P., "Zonal eddy viscosity models based on machine learning," *Flow, Turbulence and Combustion*, Vol. 103, No. 1, 2019, pp. 93–109.

[8] Taghizadeh, S., Witherden, F. D., and Girimaji, S. S., "Turbulence closure modeling with data-driven techniques: physical compatibility and consistency considerations," *arXiv preprint arXiv:2004.03031*, 2020.

[9] Sotgiu, C., Weigand, B., Semmler, K., and Wellinger, P., "Towards a general data-driven explicit algebraic Reynolds stress prediction framework," *International Journal of Heat and Fluid Flow*, Vol. 79, 2019, p. 108454.

[10] Wu, J.-L., Xiao, H., and Paterson, E., "Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework," *Phys. Rev. Fl.*, Vol. 3, No. 7, 2018, p. 074602.

[11] Wu, J.-L., Michelén-Ströfer, C., and Xiao, H., "Physics-informed covariance kernel for model-form uncertainty quantification with application to turbulent flows," *Computers & Fluids*, Vol. 193, 2019, p. 104292.

[12] Xiao, H., Wu, J.-L., Wang, J.-X., Sun, R., and Roy, C., "Quantifying and reducing model-form uncertainties in Reynolds-averaged Navier–Stokes simulations: A data-driven, physics-informed Bayesian approach," *Journal of Computational Physics*, Vol. 324, 2016, pp. 115–136.

[13] Zhang, X., Wu, J., Coutier-Delgosha, O., and Xiao, H., "Recent progress in augmenting turbulence models with physics-informed machine learning," *Journal of Hydrodynamics*, Vol. 31, No. 6, 2019, pp. 1153–1158.

[14] Müller, S., Milano, M., and Koumoutsakos, P., "Application of machine learning algorithms to flow modeling and optimization," *Annual Research Briefs*, 1999, pp. 169–178.

[15] Lee, C., Kim, J., Babcock, D., and Goodman, R., "Application of neural networks to turbulence control for drag reduction," *Physics of Fluids*, Vol. 9, No. 6, 1997, pp. 1740–1747.

[16] Gautier, N., Aider, J.-L., Duriez, T., Noack, B., Segond, M., and Abel, M., "Closed-loop separation control using machine learning," *Journal of Fluid Mechanics*, Vol. 770, 2015, pp. 442–457.

[17] Duriez, T., Brunton, S. L., and Noack, B. R., *Machine learning control-taming nonlinear dynamics and turbulence*, Vol. 116, Springer, 2017.

[18] Raibaudo, C., Zhong, P., Noack, B. R., and Martinuzzi, R. J., "Machine learning strategies applied to the control of a fluidic pinball," *Physics of Fluids*, Vol. 32, No. 1, 2020, p. 015108.

[19] Zhu, L., Zhang, W., Kou, J., and Liu, Y., "Machine learning methods for turbulence modeling in subsonic flows around airfoils," *Phys. Fluids*, Vol. 31, No. 1, 2019, p. 015105.

[20] Trehan, S., Carlberg, K. T., and Durlofsky, L. J., "Error modeling for surrogates of dynamical systems using machine learning," *International Journal for Numerical Methods in Engineering*, Vol. 112, No. 12, 2017, pp. 1801–1827.

[21] San, O., and Maulik, R., "Extreme learning machine for reduced order modeling of turbulent geophysical flows," *Physical Review E*, Vol. 97, No. 4, 2018, p. 042322.

[22] San, O., and Maulik, R., "Machine learning closures for model order reduction of thermal fluids," *Applied Mathematical Modelling*, Vol. 60, 2018, pp. 681–710.

[23] Renganathan, S. A., Maulik, R., and Rao, V., "Machine learning for nonintrusive model order reduction of the parametric inviscid transonic flow past an airfoil," *Physics of Fluids*, Vol. 32, No. 4, 2020, p. 047110.

[24] Qian, E., Kramer, B., Peherstorfer, B., and Willcox, K., "Lift & Learn: Physics-informed machine learning for large-scale nonlinear dynamical systems," *Physica D: Nonlinear Phenomena*, Vol. 406, 2020, p. 132401.

[25] Lee, K., and Carlberg, K. T., "Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders," *Journal of Computational Physics*, Vol. 404, 2020, p. 108973.

[26] Hasegawa, K., Fukami, K., Murata, T., and Fukagata, K., "Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes," *Theoretical and Computational Fluid Dynamics*, 2020, pp. 1–17.

[27] Mohan, A. T., and Gaitonde, D. V., "A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks," *arXiv preprint arXiv:1804.09269*, 2018.

[28] Mohan, A., Daniel, D., Chertkov, M., and Livescu, D., "Compressed convolutional LSTM: An efficient deep learning framework to model high fidelity 3D turbulence," *arXiv preprint arXiv:1903.00033*, 2019.

[29] Maulik, R., Lusch, B., and Balaprakash, P., "Non-autoregressive time-series methods for stable parametric reduced-order models," *Physics of Fluids*, Vol. 32, No. 8, 2020, p. 087115.

[30] Maulik, R., Mohan, A., Lusch, B., Madireddy, S., Balaprakash, P., and Livescu, D., "Time-series learning of latent-space dynamics for reduced-order model closure," *Physica D: Nonlinear Phenomena*, Vol. 405, 2020, p. 132368.

[31] Fukami, K., Nakamura, T., and Fukagata, K., "Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data," *Physics of Fluids*, Vol. 32, No. 9, 2020, p. 095110.

[32] Maulik, R., Egele, R., Lusch, B., and Balaprakash, P., "Recurrent Neural Network Architecture Search for Geophysical Emulation," *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, IEEE Computer Society, Los Alamitos, CA, USA, 2020, pp. 89–102. https://doi.org/10.1109/SC41405.2020.00012, URL https://doi.ieeecomputersociety.org/10.1109/SC41405.2020.00012.

[33] Raissi, M., Yazdani, A., and Karniadakis, G. E., "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, Vol. 367, No. 6481, 2020, pp. 1026–1030.

[34] Raissi, M., Perdikaris, P., and Karniadakis, G. E., "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, Vol. 378, 2019, pp. 686–707.

[35] Gao, H., and Wang, J.-X., "A Bi-fidelity Ensemble Kalman Method for PDE-Constrained Inverse Problems," *arXiv preprint arXiv:2003.11912*, 2020.

[36] Sun, L., and Wang, J.-X., "Physics-Constrained Bayesian Neural Network for Fluid Flow Reconstruction with Sparse and Noisy Data," *arXiv preprint arXiv:2001.05542*, 2020.

[37] Sun, L., Gao, H., Pan, S., and Wang, J.-X., "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, Vol. 361, 2020, p. 112732.

[38] Kawai, S., and Shimoyama, K., "Kriging-model-based uncertainty quantification in computational fluid dynamics," *32nd AIAA Applied Aerodynamics Conference*, 2014, p. 2737.

[39] Geneva, N., and Zabaras, N., "Quantifying model form uncertainty in Reynolds-averaged turbulence models with Bayesian deep neural networks," *Journal of Computational Physics*, Vol. 383, 2019, pp. 125–147.

[40] Maulik, R., Fukami, K., Ramachandra, N., Fukagata, K., and Taira, K., "Probabilistic neural networks for fluid flow surrogate modeling and data recovery," *Physical Review Fluids*, Vol. 5, No. 10, 2020, p. 104401.

[41] Tang, M., Liu, Y., and Durlofsky, L. J., "A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems," *Journal of Computational Physics*, 2020, p. 109456.

[42] Casas, C. Q., Arcucci, R., Wu, P., Pain, C., and Guo, Y.-K., "A Reduced Order Deep Data Assimilation model," *Physica D: Nonlinear Phenomena*, Vol. 412, 2020, p. 132615.

[43] Pawar, S., and San, O., "Data assimilation empowered neural network parameterizations for subgrid processes in geophysical flows," *arXiv preprint arXiv:2006.08901*, 2020.

[44] Fukami, K., Fukagata, K., and Taira, K., "Super-resolution reconstruction of turbulent flows with machine learning," *Journal of Fluid Mechanics*, Vol. 870, 2019, pp. 106–120.

[45] Liu, B., Tang, J., Huang, H., and Lu, X.-Y., "Deep learning methods for super-resolution reconstruction of turbulent flows," *Physics of Fluids*, Vol. 32, No. 2, 2020, p. 025105.

[46] Brunton, S. L., Noack, B. R., and Koumoutsakos, P., "Machine learning for fluid mechanics," *Annual Review of Fluid Mechanics*, Vol. 52, 2020, pp. 477–508.

[47] Fukami, K., Fukagata, K., and Taira, K., "Assessment of supervised machine learning methods for fluid flows," *Theoretical and Computational Fluid Dynamics*, 2020, pp. 1–23.

[48] Duraisamy, K., Iaccarino, G., and Xiao, H., "Turbulence modeling in the age of data," *Annu. Rev. Fluid Mech.*, Vol. 51, 2019, pp. 357–377.

[49] Weller, H. G., Tabor, G., Jasak, H., and Fureby, C., "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Computers in physics*, Vol. 12, No. 6, 1998, pp. 620–631.

[50] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," , 2015. URL https://www.tensorflow.org/, software available from tensorflow.org.

[51] Maulik, R., Sharma, H., Patel, S., Lusch, B., and Jennings, E., "A turbulent eddy-viscosity surrogate modeling framework for Reynolds-Averaged Navier-Stokes simulations," *Computers & Fluids*, 2020, p. 104777.

[52] Germano, M., Piomelli, U., Moin, P., and Cabot, W. H., "A dynamic subgrid-scale eddy viscosity model," *Physics of Fluids A: Fluid Dynamics*, Vol. 3, No. 7, 1991, pp. 1760–1765.

[53] Ott, J., Pritchard, M., Best, N., Linstead, E., Curcic, M., and Baldi, P., "A Fortran-Keras Deep Learning Bridge for Scientific Computing," *arXiv preprint arXiv:2004.10652*, 2020.

[54] Spalart, P., and Allmaras, S., "A one-equation turbulence model for aerodynamic flows," *30th aerospace sciences meeting and exhibit*, 1992, p. 439.