

Online training of neural networks in a CFD solver for sensor placement and flow inference

By C. Laurent AND K. Maeda

1. Motivation and objectives

Despite renewed popularity in recent years (Vinuesa & Brunton 2022), many machine learning (ML) methods applied to flow field data of computational fluid dynamics (CFD) simulations rely on offline processing on fixed datasets, without being deployed for online processing in a production environment such as a CFD solver. This tendency can be explained by the technical difficulties to embed deep neural networks (DNN) into common simulation tools. Parallel CFD solvers are usually written in low-level languages such as C++ or Fortran, whereas popular DNN libraries are most easily used through their high-level Python application programming interfaces (APIs); interfacing with these programs therefore necessitates either implementing complex ad hoc bindings (Novati & Koumoutsakos 2019; Rabault & Kuhnle 2019; Bieker *et al.* 2020; Rasp 2020; Garnier *et al.* 2021; Wang *et al.* 2022), or to execute them in a more isolated fashion through different runtime instances (Terraz *et al.* 2017; Meyer *et al.* 2022). In addition, the strong heterogeneity between CFD and ML operations, and the high-throughput data transfer inherent to DNNs, induce difficulty in parallelizing and synchronizing (when necessary) their execution. Finally, many modern CFD solvers target graphics processing units (GPUs) for their performance, and therefore compete with ML algorithms for these computational resources. The use of these solvers may also lead to suboptimal utilization of the central processing units (CPUs), as GPUs are intensively utilized while the CPUs remain idle (Maeda & Teixeira 2021; Maeda *et al.* 2022).

This work explores a direction to facilitate the implementation and joint parallel execution of DNN training and CFD simulations on hybrid GPU/CPU machines. More precisely, we leverage task-based parallelism through the Legion programming system (Bauer *et al.* 2012) to seamlessly interface a CFD solver, that simultaneously runs multiple simulations on GPUs, to an ensemble of DNNs, whose training processes are executed on the available CPUs, by utilizing the computational framework of Maeda & Teixeira (2022). The simultaneous generation of data and training yield a specific training procedure referred to as online learning (Hadsell *et al.* 2020; Hoi *et al.* 2021), where training is carried out on a stream of data rather than on a fixed dataset. This setting is known to lead to a training instability called catastrophic forgetting (French 1999), caused by a biased dataset, which manifests itself in a loss of accuracy for the samples generated earlier in the simulations. Here, we mitigate catastrophic forgetting by using a limited-size replay buffer (Hayes *et al.* 2021) that dynamically stores a subset of the samples generated throughout the CFD simulations. The samples stored in the replay buffer are revisited multiple times during the training, which stabilizes the learning. For simplicity and clarity, this strategy is here applied to train an ensemble of independent models, as this setting minimizes internodes communication; we also leverage ensembles to explore hyperparameters space as well as to perform uncertainty quantification (UQ). More precisely, we train online a DNN ensemble intended to infer instantaneous flow quantities

from a sparse collection of sensors in a two-dimensional, two-components, compressible, transient jet. We use this ensemble learning to construct distinct DNN models trained on different sensor placements; the overall procedure is therefore a brute force approach to find a near-optimal sensor placement in order to reconstruct the jet instantaneous transverse density profile from sparse density measurements at the same time instant. Once near-optimal locations are found for the sensors in the set, a second DNN ensemble is trained to endow the flow reconstruction with uncertainty estimates. Our work is carried out in the Stanford PSAAP-III Center, aiming at predicting laser ignition in a subscale methane/oxygen rocket combustor (Wang *et al.* 2021; Maeda *et al.* 2022). In this context, ML-aided inference of density profiles is of particular interest, as it could be used to construct a data-driven, low-fidelity, surrogate model for the mixture ignitability.

The remainder of this brief is structured as follows: Section 2 briefly introduces the software components used to couple the CFD solver and the DNN training. It also details the joint data generation and training of a DNN ensemble, as well as the replay buffer used to mitigate catastrophic forgetting. Results, including training dynamics, exploration of sensor placement, and ensemble-based uncertainty quantification are presented in Section 3. Lastly, conclusions and possible future directions are given in Section 4.

2. Method

2.1. Task-based parallelism

In order to circumvent the inherent difficulty of implementing the parallel execution of an ensemble of CFD simulations coupled to an ensemble of DNN training procedures, we utilize the Legion task-based programming system (Bauer *et al.* 2012). Following an analysis of tasks privileges and coherence mode over data, the Legion runtime then schedules the parallel and distributed execution of the tasks over the available processors. This execution does not require explicit synchronization, thus eliminating the possibility of common bugs such as deadlocks or races. A key feature of Legion lies in the possibility for the developer to write tasks intended to be executed on GPUs or CPUs, thus permitting the efficient use of heterogeneous clusters (Maeda *et al.* 2022). This feature was further assessed in multifidelity ensemble simulations of laser-induced ignition by Maeda & Teixeira (2021) using the HTR solver (Di Renzo *et al.* 2020). Recently, a computational framework was introduced by Maeda & Teixeira (2022) to couple a CFD solver for ensemble simulations to *in-situ* data analysis tools. The present work builds upon this framework and seeks to utilize it for the specific case of online ML training.

In the framework, rather than using the low-level C++ API of Legion, two of its high-level components are used. On one hand, the Regent language (Slaughter *et al.* 2015) is used to implement the CFD solver. Thanks to its CUDA code generator, Regent trivializes the implementation of GPU-targeted tasks. On the other hand, Pygion (Slaughter & Aiken 2019), the high-level Python API to Legion is used for data analysis. In the present context, it is used to implement the DNN training procedure, as Pygion offers full compatibility with existing Python libraries, such as the popular ML framework PyTorch (Paszke *et al.* 2019). The use of GPUs for data generation can be justified by multiple studies that identified the data generation process as a limiting bottleneck in joint ML-CFD applications (Rabault & Kuhnle 2019; Garnier *et al.* 2021), thus suggesting that the most performant processors (i.e., the GPUs) should be reserved for CFD tasks. Note that the GPUs are not exposed to tasks written in Pygion; therefore ML tasks are executed only on the available CPUs, with support for OpenMP multithreading to

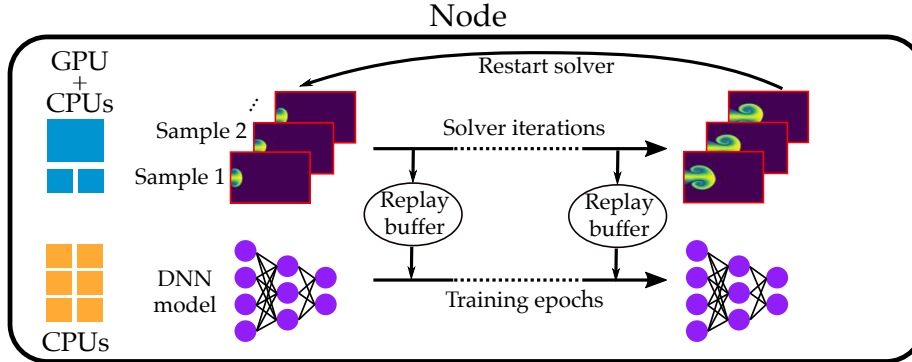


FIGURE 1. Schematic of the parallel execution on a single node comprising a GPU and multiple CPUs. The ensemble of multiple CFD simulations are executed on the GPU and a few CPUs, while a single DNN is simultaneously trained on the available CPUs. At a fixed iteration interval, called a cycle, the CFD solver passes data to a limited-size replay buffer, which serves as a time-evolving, streaming dataset to train the DNN. Once the CFD simulations are complete, and if more data are required to train the DNN, the CFD solver can be restarted with new conditions. Similar processes are executed in parallel on multiple nodes, such that an ensemble of DNN models is simultaneously trained online. All parallel task execution and data transfer are automatically handled by the Legion runtime.

accelerate tensor operations. It is also worth mentioning that even though the CFD solver and the ML tasks are not written in the same language, the implementation does not require any specific interfacing and the program runs under the same runtime instance, thus seamlessly allowing high interoperability. Further descriptions of the Regent-PyGion interoperation can be found in Maeda & Teixeira (2022).

2.2. CFD data generation

Regent is used to implement the CFD solver that resolves the two-components compressible Navier-Stokes equations. A third-order weighted essentially non-oscillatory (WENO) scheme (Borges *et al.* 2008) is employed for the spatial discretization, and a third-order Runge-Kutta scheme is used for the temporal advancement. Further description of the solver can be found in Maeda & Teixeira (2022). The case of interest is a transient methane (CH_4) jet of diameter D_{jet} , injected into quiescent oxygen (O_2). Simulation data of similar flows were previously used for offline training of DNNs (Maeda *et al.* 2022). The computational domain is two-dimensional of dimensions $L_x = 20D_{jet}$ and $L_y = 10D_{jet}$ and discretized with a uniform Cartesian grid, with $n_x = 1,000$ points in the axial direction and $n_y = 500$ points in the transverse direction.

Figure 1 illustrates the CFD data generation, where Legion distributes and executes in parallel, over multiple nodes and processors, several CFD simulations with distinct conditions. In order to create more diversity in the generated data, the jet velocity U_{jet} is uniformly sampled from the interval [150m/s, 250m/s], which corresponds to Reynolds numbers in the range of $Re = 41.6 \times 10^3$ to $Re = 69.4 \times 10^3$. A low-amplitude random disturbance is also applied to the initial density field. At a fixed time interval $\Delta_t = 0.4\tau_{flow}$, where $\tau_{flow} = 5 \times 10^{-4}$ s is the expected value of the flow-through time (i.e., calculated with $U_{jet} = 200$ m/s), the CFD simulations send data extracted from the instantaneous density field $\rho_{CH_4}(t_j)$ to the DNN model being trained on the same node. We refer to the interval Δ_t between two data generation events as a cycle. Note that we are here interested only in the methane density $\rho_{CH_4} = \rho Y_{CH_4}$, where Y_{CH_4} is the

methane mass fraction; however, for conciseness, the notation ρ will be used to denote ρ_{CH_4} in the remainder of this brief.

2.3. Online training with replay memory

The joint data generation and DNN training described in Figure 1 leads to the online learning paradigm (Orabona 2019; Hadsell *et al.* 2020; Hayes *et al.* 2021), where the model has to be trained on a stream of samples, rather than on a fixed dataset as usually done in offline learning. Online learning usually suffers from catastrophic forgetting (French 1999; Kemker *et al.* 2018): the model adapts to new training samples as they become available, but it does so at the cost of losing its predictive accuracy on older samples previously generated. Catastrophic forgetting is ubiquitous in reinforcement learning (RL), as agents have to continuously adapt and interact with an evolving environment (Rabault & Kuhnle 2019; Bae & Koumoutsakos 2022). In RL, experience replay (Zhang & Sutton 2017; Hayes *et al.* 2021) has been proven to be effective to mitigate catastrophic forgetting. Inspired by this approach, we implement a variant of experience replay for supervised learning (Meyer *et al.* 2022): a limited-size replay buffer is used to store samples as they are generated by the CFD simulations. The DNN model is thus trained on the samples contained in the replay buffer, by incremental batch learning, based on batch optimization algorithms. For validation purposes, the replay buffer splits the streaming CFD samples into two train/test groups. Once the buffer reaches its maximal capacity, samples are randomly discarded based on their age: samples generated earlier have a higher probability of being discarded than more recent ones.

The approach illustrated in Figure 1 permits the simultaneous online training of an ensemble of N_M DNN models, based on data generated from $N_M \times N_S$ CFD samples, where N_S is the number of simulations on a single node (here set to $N_S = 32$). In the present case, our objective is to infer an instantaneous transverse CH_4 density profile $\rho_{t_j}(y)$, at $t = t_j$ and located at $x_c = 2D_{jet}$, from a set of sparse CH_4 density sensors $\mathcal{S}_{t_j} = \{\rho_{t_j}(x_i, y_i)\}_{1 \leq i \leq K_S}$, providing measurements at the same time-instant t_j , and where K_S is the number of sensors in a given collection. We therefore aim at instantaneous flow reconstruction, rather than encoding the dynamical history of the system. Instantaneous flow field inference from sparse measurements is a popular problem of data-driven fluid dynamics: it has been used for instance to estimate sea surface temperature fields from sparse satellite data (Erichson *et al.* 2020), or to reconstruct the velocity field in the wake of bluff bodies (Loiseau *et al.* 2018). In the present study, carried out in the broader context of laser ignition prediction in CH_4/O_2 rocket combustors, a DNN trained to infer the methane density could be used as a low-fidelity surrogate model to quantify the mixture ignitability. Indeed, in this two-component jet, the predicted CH_4 density directly gives access to the local flow composition, and therefore to its flammability; such a data-driven model might therefore be used to estimate the probability of a successful ignition.

The inference of a one-dimensional profile from a set of sparse point-wise measurements is comparable to a variant of spatial super-resolution, a task often handled thanks to convolutional neural networks. Therefore, the DNNs considered in this study consist of six residual blocks (He *et al.* 2016), combining upscaling transformations and one-dimensional spatial convolutions in the transverse direction, yielding 37,475 parameters per DNN. We also tested several variants of multilayer perceptrons, which gave similar results as the residual networks; generally speaking, the problem seemed to display only a weak sensitivity with respect to the model architecture. It is also worth noting that we do not impose any physical constraint; however, as methane density cannot be negative,

a positivity constraint imposed by a rectified linear unit (ReLU) activation on the DNNs output layer would probably yield more realistic predictions. For simplicity, the DNNs trained in parallel share a common architecture; this is not a requirement, as significantly different models could be trained simultaneously. Similarly, multiple models could be trained on the same node, on the conditions that they are trained sequentially and that their training does not overstrain the node CPU cores. The present setting is intentionally simplified for clarity, as well as to maximize the training efficiency of each DNN model. In addition, both generated data and trained models are relatively small, meaning that training offline on a fixed dataset would be feasible; however, such approach would not efficiently use the processors available on heterogeneous supercomputers, which is a key feature of our implementation.

All computations are performed on the Lassen supercomputer, at Lawrence Livermore National Laboratory. Each node reserved by Legion comprises one NVIDIA V100 GPU, and 10 IBM Power9 CPU cores, including 2 cores dedicated to the runtime metatasks.

3. Results

3.1. Exploration of sensor placement

In this first application, the procedure described in Section 2 is used to train an ensemble of 16 DNN models, with the objective of finding an informative placement for a set of 4 sensors, that would allow the model to infer as accurately as possible the transverse CH_4 density profile $\rho_{t_j}(y)$ at $x = x_c$. To do so, a brute-force approach is employed: each one of the 16 DNNs is trained to infer $\rho_{t_j}(y)$ from a different set of sensors $\mathcal{S}_{t_j} = \{\rho_{t_j}(x_i, y_i)\}_{1 \leq i \leq 4}$, where the individual sensor locations $\{(x_i, y_i)\}_{1 \leq i \leq 4}$ are randomly chosen from the region $[0, 4D_{jet}] \times [-2D_{jet}, 2D_{jet}]$. Note that this random search strategy, with such a low number of placement candidates, has only little chance of finding a near-optimal solution. Even though this was not attempted in the present work, more efficient search algorithms could make use of Bayesian optimization (Frazier 2018), with the intention of finding the best (i.e. global optimum) sensor placement. Each one of the DNNs is trained in a supervised manner on pairs $(\mathcal{S}_{t_j}, \rho_{t_j}(y))$ generated by 32 distinct CFD simulations, for a total of $16 \times 32 = 512$ simulations with distinct conditions. Each simulation is run for nine cycles and restarted once, yielding a total of 620 training samples for each DNN. The size of the replay buffer is set to 192, and the models are optimized by minimizing the mean square error (MSE) $MSE = \|\rho_{t_j}(y) - \hat{\rho}_{t_j}(y)\|^2$, where $\hat{\rho}_{t_j}(y)$ is the model prediction. To do so, the Adam algorithm (Kingma & Ba 2014) iterates for 10 epochs over the replay buffer during each cycle. In addition to the test set cached in the streaming replay buffers, we also pregenerate a dataset \mathcal{D}_{off} containing 5,000 samples for offline cross-validation and comparison between models.

Figure 2(a) shows a significant dispersion in the accuracy of the 16 trained DNNs, which suggests that the sensor placement has a first-order effect on the trainability and the final predictive capability of the models. The cross-validation MSE for the best model, computed on the pregenerated dataset \mathcal{D}_{off} [red curve in Figure 2(b)] constantly decreases during the training, thus proving that the replay buffer successfully mitigates catastrophic forgetting. It is worth mentioning that the MSE computed from the replay buffer's test set (blue curve) suddenly increases at the beginning of each cycle, when new samples are provided by the CFD simulations. Several epochs are necessary for the model to adapt to these new samples, after which a cycle-to-cycle improvement is reached (orange curve). Multiple number of epochs per cycle were tested, and 10 appeared to be

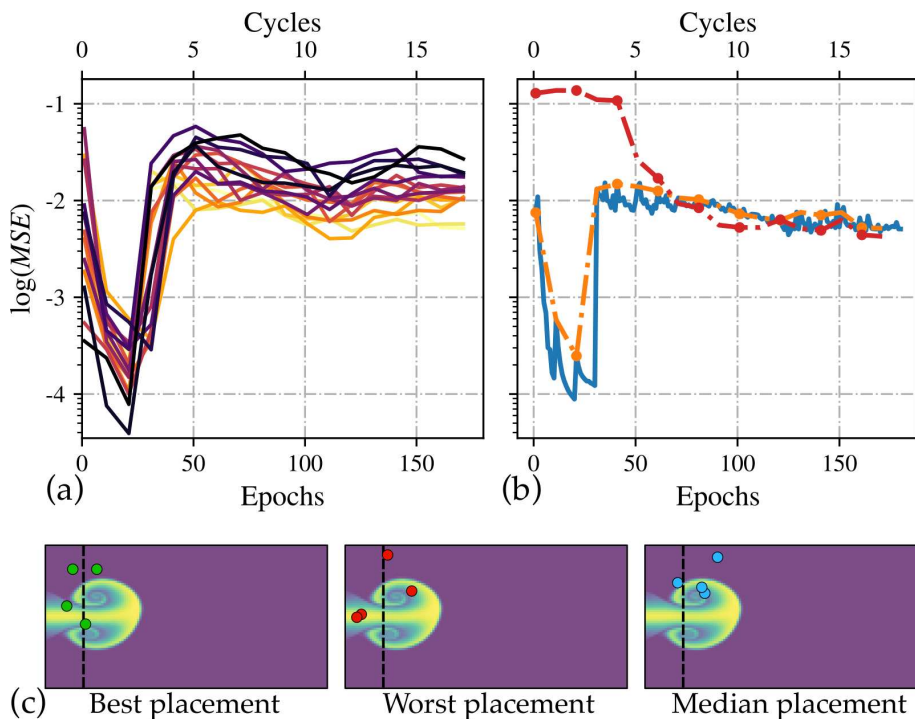


FIGURE 2. (a) Cross-validation MSE on the offline dataset \mathcal{D}_{off} for the 16 DNNs, recorded at the end of each cycle throughout their training. Colors indicate each individual model, based on their respective final accuracy after 180 epochs: darker colors are for the least accurate models, while lighter colors are for the most accurate ones. (b) Test and validation MSE for the best model: offline cross-validation MSE (red), streaming test MSE computed at every epoch from the replay buffer (blue), and streaming test MSE recorded at the end of every cycle (orange). (c) Sensor placements yielding the best, the worst, and the median models, superimposed onto the CH_4 density field extracted from one of the simulations at $t_j = 3.6\tau_{flow}$.

a lower limit for an adequate plasticity of the model—in other words, for its ability to quickly adapt to new data. Note that these sudden jumps are less pronounced after the simulations are restarted at the tenth cycle, which also indicates an effective mitigation of catastrophic forgetting. Examining the placement for the best, the worst, and the median set of sensors, in Figure 2(c), it appears that sensors close to each other yield a poor model predictive capability; this is unsurprising, as this spatial proximity results in highly correlated signals. In addition, the worst and the median placements have all sensors located in the upper part of the domain, which may result in inaccurate prediction in the lower part.

The inaccurate prediction noted above is confirmed by the profiles inferred in Figure 3(a), where the worst and median models are unable to predict the transverse methane density profile in the lower part of the jet (right side of the profile plot). By contrast, the best sensor placement (green) yields a DNN able to reconstruct local and transient structures of the jet. Interestingly, all models, including the one trained on the best sensor placement, predict locally negative CH_4 densities in the region surrounding the central jet; such values are unphysical, which indicates that predictions could be improved by enforcing a positivity constraint on the output layer of the DNNs.

Ideally, a model should be able to infer the flow features from a sparse set of measure-

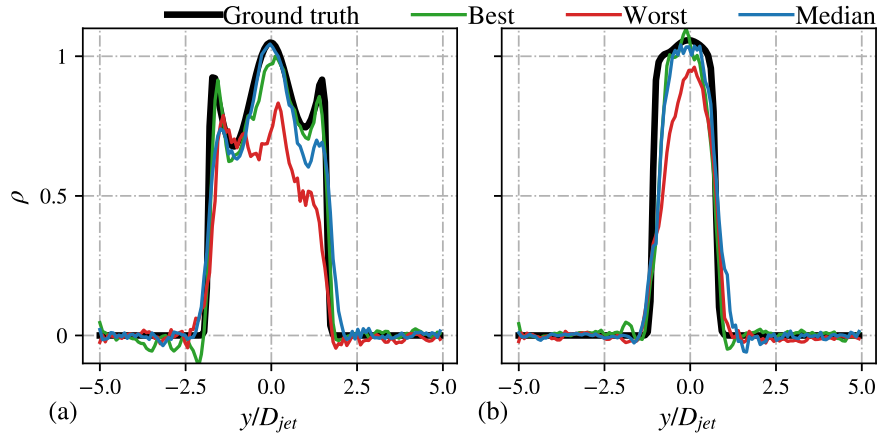


FIGURE 3. Transverse CH₄ density profile inference for two samples from the cross-validation dataset \mathcal{D}_{off} : (a) at $t_j = 3.6\tau_{flow}$ and (b) at $t_j = 2.8\tau_{flow}$. The DNNs trained with the best (green), the worst (red), and the median (blue) sensor placements are compared to the ground truth (black).

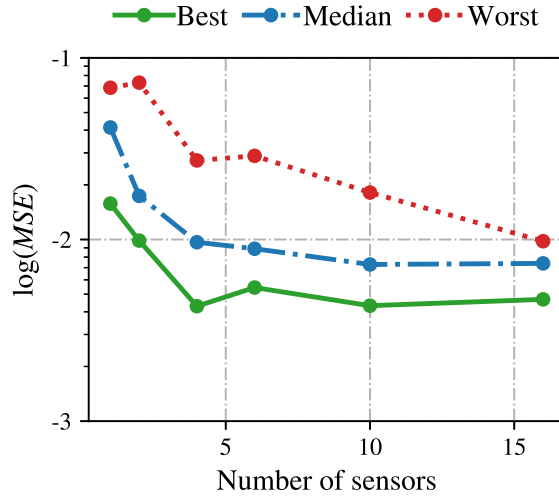


FIGURE 4. Cross-validation MSE computed on the dataset \mathcal{D}_{off} for multiple sets of (trained) models. A different number of sensors K_S is used to train each one of the sets. Each set of models for a given K_S consists of 16 DNNs trained on measurements from 16 distinct, randomly chosen, sensor placements. Only the best, worst, and median models for each set are represented.

ments. Therefore, in order to determine the minimal number of sensors K_S required to accurately predict $\rho_{t_j}(x_c, y)$, the procedure above is repeated for $K_S = 1$ to $K_S = 16$. Note that the DNN architecture has to be adapted in order to match the input-output dimensions. In order to keep the training procedure as similar as possible between different values of K_S , we preserve the first five residual blocks and change only the dimensions of the last convolutional layer in the sixth block; all other training hyperparameters are left unchanged.

Figure 4 shows that increasing the number of sensors decreases the dispersion in the trained models accuracy. In other words, for a dense set of sensors (i.e., $K_S \geq 16$), the

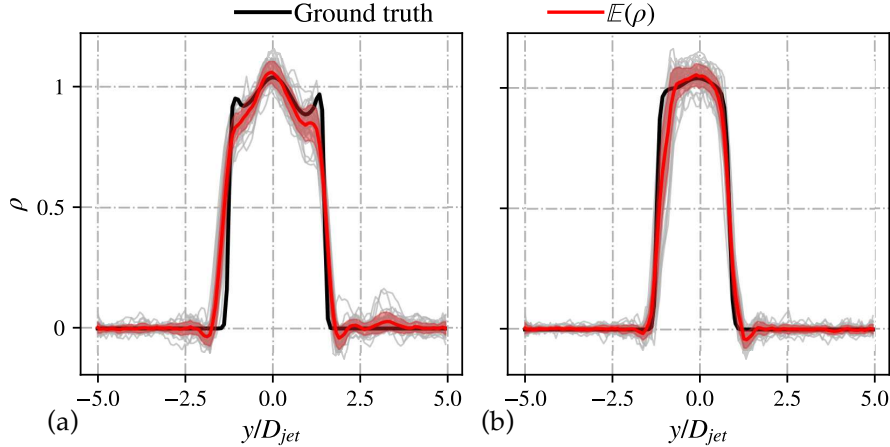


FIGURE 5. Methane density profile inference with ensemble-based UQ, on two samples from the cross-validation dataset \mathcal{D}_{off} : (a) at $t_j = 3.6\tau_{flow}$ and (b) at $t_j = 2.8\tau_{flow}$. Thin gray curves represent predictions of the individual models. The ensemble prediction comprises an expected value (red curve), computed as the mean of the 64 individual predictions, and a standard deviation (red area).

individual sensors locations have only little influence on the model predictive capability. It is also worth observing that a dense set of sensors does not necessarily improve the accuracy of the overall training procedure, as the predictive capability of the best model seems to saturate for $K_S \geq 4$. Note that four sensors appears as a critical lower limit, since models with $K_S \leq 4$ display poor accuracy. This might be partly alleviated by handcrafting the DNN architectures for these cases, which was not attempted in this work. However, in the very low data regime, improving the DNN architecture would probably be insufficient to counterbalance the lack of information. Figure 4 also shows that the dispersion between models is the largest for $K_S = 4$, which indicates that in this case, it is necessary to choose appropriate locations for the individual sensors in order to infer the CH_4 density profile accurately.

3.2. Prediction under uncertainty

The procedure described in Section 2 can also be used to train an ensemble of DNN models to perform uncertainty quantification (UQ). In this second application, inspired from UQ based on deep ensembles (Lakshminarayanan *et al.* 2017; Ganaie *et al.* 2021), the number of sensors is set to $K_S = 4$ and all models use the best sensor placement shown in Figure 2(c). We train online an ensemble of 64 DNNs, using 64 GPUs and 640 CPU cores. Note that this approach produces distinct models, as they are randomly initialized with different seeds and are not exposed to the same samples during training. This results in two sources of uncertainty: on one hand, a training uncertainty due to the models random initialization, and on the other hand, a data uncertainty stemming from the distinct CFD simulations used to trained different models. Here, we assume that the latter is prevalent, which implies that the variability between predictions made by different models mostly reflects the variability of the data.

The ensemble prediction can then be estimated by computing the expected value of the individual model predictions, and their standard deviation can be used to quantify the uncertainty of this prediction. Figure 5 shows that the ensemble expected value successfully approximates the transverse CH_4 density profile; in addition, it yields smoother

profiles in comparison to the models shown in Figure 3, indicating a lower sensitivity to noise. The uncertainty is the largest near the flow sharp, local features, some of which might even be classified as outliers for the distribution of the ensemble predictions [e.g., Figure 5(a)].

4. Conclusions

We leveraged task-based parallelism through the Legion programming system in order to execute a parallel and distributed workflow by utilizing the framework of Maeda & Teixeira (2022), where ensembles of CFD simulations generate data to train online an ensemble of ML models. Most importantly, Legion high-level APIs, as well as the interoperation of the Regent-based CFD solvers with ML libraries via Pygion, greatly facilitated the implementation of a program that would be tedious to write using classical parallel programming systems. The present implementation, by targeting GPUs for the CFD solver and CPUs for the ML training of an ensemble of small-scale models, is effective on heterogeneous supercomputers, where all type of processors can be used efficiently. We demonstrated that this approach is suitable for the online training of ensembles of independent ML models at small scale. The online learning is achieved thanks to a replay buffer that proved to successfully mitigate catastrophic forgetting, as well as to enable a sufficient plasticity of the ML models to quickly adapt to new streaming data.

Ongoing work includes extending the present study to three-dimensional, complex, turbulent fluid flows, where larger ML models are required to predict the flow features. We also envision applications demanding feedback from the ML models to the CFD simulations; those include for instance online reinforcement learning and active learning. For example, in the application presented earlier, the ML models could be coupled with a Bayesian optimization setting in order to instruct the CFD simulations to dynamically optimize the sensor placement, with the objective of finding the placement global optimum. In highly parallel configurations, internode communications may become prevalent. Further exploration of the existing features of Legion, including the task mapping interface, could mitigate such a performance bottleneck.

Acknowledgments

This investigation was funded by the Advanced Simulation and Computing (ASC) program of the US Department of Energy’s National Nuclear Security Administration (NNSA) via the PSAAP-III Center at Stanford, Grant No. DE-NA0002373. The authors are grateful to Dr. Thiago Teixeira and Dr. Elliott Slaughter for assistance with Legion, as well as Prof. Gianluca Iaccarino for useful comments and discussions.

REFERENCES

- BAE, H. J. & KOUMOUTSAKOS, P. 2022 Scientific multi-agent reinforcement learning for wall-models of turbulent flows. *Nat. Commun.* **13**, 1–9.
- BAUER, M., TREICHLER, S., SLAUGHTER, E. & AIKEN, A. 2012 Legion: expressing locality and independence with logical regions. In *SC’12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, IEEE, pp. 1–11.

- BIEKER, K., PEITZ, S., BRUNTON, S. L., KUTZ, J. N. & DELLNITZ, M. 2020 Deep model predictive flow control with limited sensor data and online learning. *Theor. Comp. Fluid Dyn.* **34**, 577–591.
- BORGES, R., CARMONA, M., COSTA, B. & DON, W. S. 2008 An improved weighted essentially non-oscillatory scheme for hyperbolic conservation laws. *J. Comput. Phys.* **227**, 3191–3211.
- DI RENZO, M., FU, L., & URZAY, J. 2020 HTR solver: An open-source exascale-oriented task-based multi-GPU high-order code for hypersonic aerothermodynamics *Comput. Phys. Comm.* **255**, 107262.
- ERICHSON, N. B., MATHELIN, L., YAO, Z., BRUNTON, S. L., MAHONEY, M. W. & KUTZ, J. N. 2020 Shallow neural networks for fluid flow reconstruction with limited sensors. *Proc. R. Soc. A* **476**, 20200097.
- FRAZIER, P. I. 2018 A tutorial on bayesian optimization. [Preprint] *arXiv:1807.02811*.
- FRENCH, R. M. 1999 Catastrophic forgetting in connectionist networks. *Trends Cogn. Sci.* **3**, 128–135.
- GANAIE, M. A., HU, M., MALIK, A. K., TANVEER, M. & SUGANTHAN, P. N. 2021 Ensemble deep learning: a review. [Preprint] *arXiv:2104.02395*.
- GARNIER, P., VIQUERAT, J., RABAULT, J., LARCHER, A., KUHNLE, A. & HACHEM, E. 2021 A review on deep reinforcement learning for fluid mechanics. *Comput. Fluids* **225**, 104973.
- HADSELL, R., RAO, D., RUSU, A. A. & PASCANU, R. 2020 Embracing change: continual learning in deep neural networks. *Trends Cogn. Sci.* **24**, 1028–1040.
- HAYES, T. L., KRISHNAN, G. P., BAZHENOV, M., SIEGELMANN, H. T., SEJNOWSKI, T. J. & KANAN, C. 2021 Replay in deep learning: current approaches and missing biological elements. *Neural Comput.* **33**, 2908–2950.
- HE, K., ZHANG, X., REN, S. & SUN, J. 2016 Deep residual learning for image recognition. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.
- HOI, S. C., SAHOO, D., LU, J. & ZHAO, P. 2021 Online learning: a comprehensive survey. *Neurocomputing* **459**, 249–289.
- KEMKER, R., MCCLURE, M., ABITINO, A., HAYES, T. & KANAN, C. 2018 Measuring catastrophic forgetting in neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 3390–3398.
- KINGMA, D. P. & BA, J. 2014 Adam: a method for stochastic optimization. [Preprint] *arXiv:1412.6980*.
- LAKSHMINARAYANAN, B., PRITZEL, A. & BLUNDELL, C. 2017 Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems* pp. 6405–6416.
- LOISEAU, J.-C., NOACK, B. R. & BRUNTON, S. L. 2018 Sparse reduced-order modelling: sensor-based dynamics to full-state estimation. *J. Fluid Mech.* **844**, 459–490.
- MAEDA, K. & TEIXEIRA, T. 2021 Application-oriented investigation of task-based ensemble co-processing on heterogeneous supercomputers. *Annual Research Briefs*, Center for Turbulence Research, Stanford University, pp 217–224.
- MAEDA, K. & TEIXEIRA, T. 2022 Task-based framework for physics-based ensemble simulation and in-situ data processing. *Annual Research Briefs*, Center for Turbulence Research, Stanford University, pp 97–110.

- MAEDA, K., TEIXEIRA, T., WANG, J. M., HOKANSON, J. M., MELONE, C., DI RENZO, M., JONES, S., URZAY, J. & IACCARINO, G. 2022 An integrated heterogeneous computing framework for ensemble simulations of laser-induced ignition. *[Preprint] arXiv:2202.02319*.
- MEYER, L., RIBÉS, A. & RAFFIN, B. 2022 Simulation-based parallel training. *[Preprint] arXiv:2211.04119*.
- MITTAL, S. & VETTER, J. S. 2015 A survey of CPU-GPU heterogeneous computing techniques. *ACM Comput. Surv.* **47**, 1–35.
- NOVATI, G. & KOUMOUTSAKOS, P. 2019 Remember and forget for experience replay. *Proceedings of the 36th International Conference on Machine Learning*, PMLR **97**, 4851–4860.
- ORABONA, F. 2019 A modern introduction to online learning. *[Preprint] arXiv:1912.13213*.
- PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L. & DESMAISON, A. 2019 Pytorch: an imperative style, high-performance deep learning library. *In 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, pp. 8024–8035.
- RABAULT, J. & KUHNLE, A. 2019 Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. *Phys. Fluids* **31**, 094105.
- RASP, S. 2020 Coupled online learning as a way to tackle instabilities and biases in neural network parameterizations: general algorithms and Lorenz 96 case study (v1.0). *Geosci. Model Dev.* **13**, 2185–2196.
- SLAUGHTER, E. & AIKEN, A. 2019 Pygion: flexible, scalable task-based parallelism with Python. *2019 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI (PAW-ATM)*, pp. 58–72.
- SLAUGHTER, E., LEE, W., TREICHLER, S., BAUER, M. & AIKEN, A. 2015 Regent: a high-productivity programming language for HPC with logical regions. *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, pp. 1–12.
- TERRAZ, T., RIBES, A., FOURNIER, Y., IOOSS, B. & RAFFIN, B. 2017 Melissa: large scale in transit sensitivity analysis avoiding intermediate files. *SC'17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, pp. 1–14.
- VINUESA, R. & BRUNTON, S. L. 2022 Enhancing computational fluid dynamics with machine learning. *Nat. Comput. Sci.* **2**, 358–366.
- WANG, J., DI RENZO, M., WILLIAMS, C., URZAY, J. & IACCARINO, G. 2021 Progress on laser ignition simulations of a CH₄/O₂ subscale rocket combustor using a multi-GPU task-based solver. *Annual Research Briefs*, Center for Turbulence Research, Stanford University, pp. 129–142.
- WANG, Q., YAN, L., HU, G., LI, C., XIAO, Y., XIONG, H., RABAULT, J. & NOACK, B. R. 2022 DRLinFluids—an open-source python platform of coupling deep reinforcement learning and OpenFOAM. *[Preprint] arXiv:2205.12699*.
- ZHANG, S. & SUTTON, R. S. 2017 A deeper look at experience replay. *[Preprint] arXiv:1712.01275*.