*J. Plasma Phys.* (2022), *vol.* 88, 895880602 © The Author(s), 2022.
Published by Cambridge University Press
doi:10.1017/S0022377822001180

# Machine-learning-based models in particle-in-cell codes for advanced physics extensions

**Chiara Badiali**[1,†,‡], **Pablo J. Bilbao**[1,‡], **Fábio Cruz**[1,2] and **Luís O. Silva**[1,†]

[1]GoLP/Instituto de Plasmas e Fusão Nuclear, Instituto Superior Técnico, Universidade de Lisboa, 1049-001 Lisbon, Portugal

[2]Inductiva Research Labs, Rua da Prata 80, 1100-420 Lisboa, Portugal

(Received 31 May 2022; revised 28 October 2022; accepted 1 November 2022)

In this paper we propose a methodology for the efficient implementation of machine learning (ML)-based methods in particle-in-cell (PIC) codes, with a focus on Monte Carlo or statistical extensions to the PIC algorithm. The presented approach allows for neural networks to be developed in a Python environment, where advanced ML tools are readily available to proficiently train and test them. Those models are then efficiently deployed within highly scalable and fully parallelized PIC simulations during runtime. We demonstrate this methodology with a proof-of-concept implementation within the PIC code OSIRIS, where a fully connected neural network is used to replace a section of a Compton scattering module. We demonstrate that the ML-based method reproduces the results obtained with the conventional method and achieves better computational performance. These results offer a promising avenue for future applications of ML-based methods in PIC, particularly for physics extensions where a ML-based approach can provide a higher performance increase.

**Key words:** plasma simulation

## 1. Introduction

The use of computer simulations has had a major impact in furthering our understanding of kinetic plasma processes in a variety of laboratory and astrophysical scenarios. The particle-in-cell (PIC) method (Dawson 1983; Hockney & Eastwood 1988) has been the flagship technique for the study of these processes over the past few decades. In the PIC method, plasma is described as a collection of charged particles interacting through self-consistent electromagnetic fields. In each simulation time step, the plasma charge and/or current densities are computed from the distribution of particles in configuration and velocity spaces and deposited on a grid that discretizes the simulation domain. These densities are then used to advance the electromagnetic field via Maxwell's equations, which in turn are used to advance particle momenta and positions.

† Email addresses for correspondence: chiara.badiali@tecnico.ulisboa.pt, luis.silva@tecnico.ulisboa.pt
‡ C. Badiali and P.J. Bilbao contributed equally to this work.

The PIC method has found remarkable success in modelling a vast range of systems in plasma physics. Leveraged on the high scalability of PIC in high-performance computing (HPC) infrastructures, state-of-the-art codes routinely simulate $\sim 10^{10}$ particles in grids with $\sim 1000^3$ cells describing very large systems (i.e. with sizes well beyond plasma kinetic scales) (Shukla *et al.* 2020; Arrowsmith *et al.* 2021).

The increasing complexity of the systems studied with the PIC algorithm has also led to the development of additional physics extensions to this method, e.g. to model Coulomb collisions (Nanbu 1997; Takizuka & Abe 1977), ionization (Kemp, Pfund & Meyer-ter Vehn 2004), radiative losses (Vranic *et al.* 2016*b*) and quantum electrodynamics processes (Vranic *et al.* 2016*a*). However, these modules often include advanced numerical methods with significant computational overhead, hindering the performance and scalability of PIC. Another example of costly algorithms in PIC simulations is that introduced by numerical solvers that account for vacuum polarization (Grismayer *et al.* 2021) or particle equations in non-trivial space–time metrics (Bacchini *et al.* 2018; Parfrey, Philippov & Cerutti 2019), which require computationally expensive iterative methods or algorithms highly tailored to capture specific nonlinearities.

Advanced physics methods in PIC can introduce computational overhead for different reasons. In the specific case of binary collisions, particles are paired when close in configuration space and scattered according to a probability distribution function describing the collisional process (Takizuka & Abe 1977; Miller & Combi 1994; Sherlock 2008; Higginson 2017). Depending on the complexity of the process, the probability may be computed, either analytically or numerically, or interpolated from large data tables stored in memory or in files, which can be highly computationally inefficient.

Machine learning (ML) offers a promising avenue for the discovery of new algorithms that could facilitate the inclusion of advanced physics modules in PIC models via, for example, generalizable approximators of unknown functions or new solvers of partial differential equations. Early works incorporating ML-based methods in the PIC loop focused on assisting (Kube, Churchill & Sturdevant 2021) and replacing (Aguilar & Markidis 2021) the numerical solver of Maxwell's equations, showing that its accuracy is preserved.

In spite of the potential of ML-based methods for PIC algorithms, developing and deploying ML-based methods in state-of-the-art PIC codes presents a substantial challenge. While the former are developed in modern languages such as Python or Julia due to their flexibility and large pool of open-source resources (e.g. the Python libraries TensorFlow (Abadi *et al.* 2015), Keras (Chollet 2018) and PyTorch (Paszke *et al.* 2019)), the latter are usually written in lower-level languages such as Fortran or C++, to maximize performance and scalability in large HPC systems.

Integrating the ML model development and PIC production environments is essential to ensure an efficient model experimentation cycle. However, to the extent of our knowledge, this integration has not yet been discussed. In this work, we propose an interface to develop and deploy ML models in state-of-the-art PIC codes, in particular neural networks (NNs). This interface, based in open-source software, allows model training to be done in Python and final model parameters to be read and used for inference during a PIC simulation. We implement this interface in the PIC code OSIRIS (Fonseca *et al.* 2002, 2008) and demonstrate it in simulations using a Compton scattering module (Del Gaudio *et al.* 2020) available in OSIRIS, demonstrating how to generalize ML techniques to address the challenges of advanced physics modules in the PIC algorithm.

This paper is organized as follows. In § 2, a detailed description of the ML–PIC interface in presented. In § 3, we present a proof-of-concept use of that interface: the use case, concerning the evaluation of the probability of Compton scattering events, is detailed in

§ 3.1, the ML methods developed for this particular use case are presented in § 3.2 and a comprehensive study of the computational performance, accuracy and applicability of these methods in production PIC simulations is discussed in § 3.3. In § 4, our conclusions are presented.

## 2. The ML–PIC interface

Recent works on assisting plasma kinetic simulations (Aguilar & Markidis 2021; Kube *et al.* 2021) with ML-based methods offer promising prospects for efficiently solving challenging substeps of the PIC algorithm. It is thus expected that ML-based methods and plasma simulations will be used in tandem in the near future. Our work focuses on designing an efficient interface for the development and deployment of ML-based methods, in particular NNs, into state-of-the-art PIC codes. Such an interface is challenging to design, since ML-based models are usually developed in high-level environments, such as Python, while PIC codes tend to be written in lower-level languages such as Fortran. This duality of environments, although technically demanding to manage, allows users to leverage the advantages of each of them.

Fortran is a fast and efficient computational language. Due to its high scalability in large HPC systems, it is widely used in large scientific computing applications. For these reasons, many PIC codes are written in Fortran, e.g. OSIRIS (Fonseca *et al.* 2002, 2008), EPOCH (Arber *et al.* 2015), Tristan (Buneman 1993) and UPIC (Decyk 2007). On the other hand, ML models are typically trained and run in high-level languages, such as Python, whose libraries focused on ML typically interface with code written in more efficient, lower-level languages (such as C/C++). This allows Python users to efficiently train and develop ML models. These libraries are also capable of running in parallel during both training and inference in CPU, GPU and other advanced architectures.

The lack of an interface between Fortran and Python environments makes the task of implementing ML-based models in highly scalable computational codes arduous. Recent efforts have significantly simplified this task. The micro-framework neural-fortran ( 2019) was developed with the intent of providing a library capable of performing the basic operations needed for a NN (e.g. gradient descent optimization, fully connected dense layer calculations) within a Fortran environment. Further work expanded this library into the Fortran–Keras Bridge (FKB) (Ott *et al.* 2020). With FKB, one can train NNs in a Python environment with the Keras library and export their parameters to files that are then read and used for inference by the FKB Fortran. This library constitutes one of the first attempts at bridging both environments and facilitates the use of NNs in highly scalable, computationally intensive Fortran codes.

In this work, neural-fortran and the FKB library are used in the OSIRIS code to run ML-based models in production PIC simulations. In figure 1 we outline the workflow necessary to efficiently train and utilize a NN model within OSIRIS. Firstly, a training dataset must be obtained. Our data were generated using algorithms previously implemented in OSIRIS and were later loaded into the Python environment and employed to train a NN.

After the training has been performed, the FKB Python library is used to export the model into a compatible format for the Fortran side of the workflow. OSIRIS has been modified to include a NN object which stores the model into memory, from the exported file, at the beginning of the simulation. Moreover, this NN object has a subroutine which can run the model and execute it for any given batch of inputs. The FKB originally included a version of neural-fortran. Thus, the FKB is capable of performing the same operations as neural-fortran. Over time, the libraries have diverged and neural-fortran has continued to be further optimized. For this reason the initial loading is done with the FKB Fortran

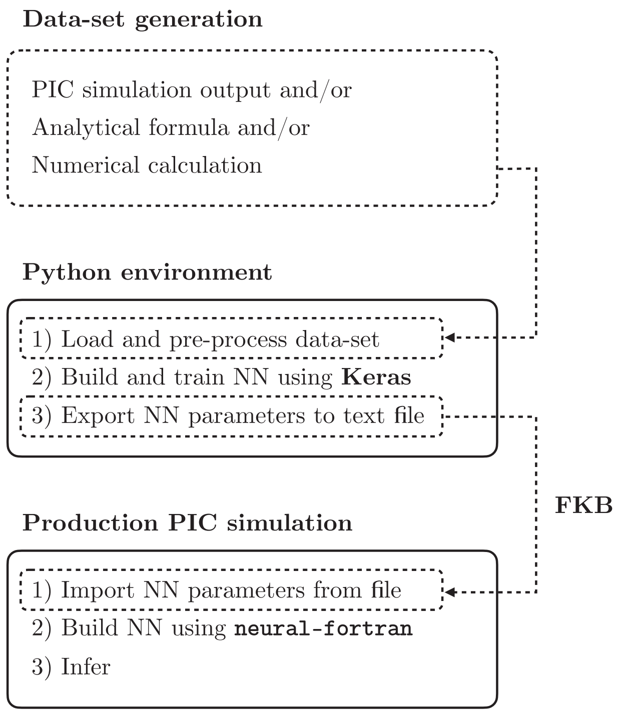
Data-set generation
PIC simulation output and/or
Analytical formula and/or
Numerical calculation
Python environment
1) Load and pre-process data-set
2) Build and train NN using Keras
3) Export NN parameters to text file
FKB
Production PIC simulation
1) Import NN parameters from file
2) Build NN using neural-fortran
3) Infer

FIGURE 1. Schematic representation of data acquisition, ML model training and production workflow. The ML model is loaded in production simulations during runtime using the FKB and interpreted with the neural-fortran microframework within PIC.

implementation and then neural-fortran is employed in the subroutine responsible for the model computation. During runtime, after the model has been loaded into memory, the NN object can be invoked and given inputs in the same manner one would call any other subroutine.

As far as we are aware, the workflow outlined here is the first of its kind for PIC simulations. In the next section, a NN is developed and deployed to replace a long analytical calculation in PIC simulations. A direct comparison between the computational performance of the NN inference and the analytical calculation in a production environment was performed, to demonstrate the potential advantages of ML-based techniques in the context of production PIC runs.

## 3. The ML–PIC proof of concept

As a proof of concept, here a NN that replaces an analytical calculation in a Compton scattering module for PIC is presented. The choice of this specific task has been driven by its simplicity, which allows us to have better control of the performance of the classical and ML-based methods.

### 3.1. *Compton scattering in PIC*

Compton scattering describes the most basic interaction of radiation with matter via the binary scattering between leptons and photons (Compton 1923). Monte Carlo techniques are usually employed to implement this phenomenon in PIC simulations, as the quantum nature of the process is intrinsically stochastic. Here, we quickly outline the current implementation of the Compton scattering module in the OSIRIS PIC code. For an in-depth description of this implementation, we refer readers to Del Gaudio *et al.* (2020).

The algorithm follows three steps. Firstly, the simulation space is divided into collision cells, in which the particles are binned. From this binning, a list of colliding leptons and photons is produced for each cell. Secondly, the no-time-counter method is employed to significantly shorten the list of pairs to be collided (Bird 1989). For each pair in this list, the probability of interaction is then calculated and compared against a random number. Finally, for each pair deemed to collide, the kinetic collision is resolved and the momentum of the particles is updated.

### 3.2. *Machine learning methods*

Our proof of concept aims at replacing the calculation of the probability of interaction via Compton scattering. This probability is a function of the weights of the macroparticles, their momentum and a normalization factor. This calculation is detailed in § 3 of Del Gaudio *et al.* (2020). The photon momentum is first Lorentz boosted to the lepton frame. Then, the probability is calculated using the Klein–Nishina cross-section and conservation of momentum. Finally, the evaluated cross-section is Lorentz boosted back into the simulation (laboratory) frame to obtain the probability of interaction between time steps. This calculation is a good candidate for a ML proof of concept, mainly because, being analytical and optimized within the PIC, it represents the worst-case testing scenario for a ML model to compete against. A NN is used to predict a value $y' \in [0, 1]$, corresponding to the probability of interaction. The NN takes as input the momenta of the macro-photons and macro-leptons in the simulation frame in units of $m_e c$ ($\boldsymbol{p}_\gamma$ and $\boldsymbol{p}_e$, respectively) and the maximum probability of interaction in each PIC cell $P_{\max} = 2\sigma_T c \Delta t \max [w_e, w_\gamma]$, where $\sigma_T$ is the Thompson cross-section, $\Delta t$ is the time step and $\max [w_e, w_\gamma]$ is the maximum macroparticle weight within the cell (Del Gaudio *et al.* 2020). During the training process, the analytical probability of interaction $y$ is used to compute a loss function and to update the NN via back-propagation.

The training data were collected from different 2D3V OSIRIS simulations. In these simulations, a uniform electron–positron–photon plasma with isotropic waterbag distributions in momentum space ($f(p) = n_0 H(p + p_0)$, where $n_0$ is a constant density, $H(p)$ is the Heaviside function and $p = |\boldsymbol{p}|$ is the magnitude of the particle momentum) was initialized and allowed to interact through Compton scattering. Every time a probability calculation was performed, the result was outputted alongside the relevant information employed in the calculation. Once the simulation was finished, all the data were collected for training. Three datasets were produced from three simulations with $p_0/m_e c = 25$, 50 and 100.

The NN architecture was chosen such that each output computation was computationally inexpensive. We used a small number of layers and neurons per layer, which translated into a small number of floating point operations. By keeping the neural layers as simple as possible (i.e. only employing fully connected layers), we ensure that all operations can be vectorized in all common compilers both in high- and low-level languages. Thus, the NN structure consisted of three hidden layers with 9, 12 and 9 neurons each and one neuron on the output layer. The hidden layers used ReLU (Fukushima 1969) as activation functions, whereas the output layers used sigmoid (Han & Moraga 1995), to guarantee that the predicted value was in the range [0, 1]. As seen in figure 2(*a*), the NN was trained by providing the relevant information of many photon–lepton pairs and their associated probability of interaction. The NN computes the probability of interaction between the macroparticles $y'$. During the training process, many targets and outputs of the NN are compared employing a loss function (Chollet 2017), that we took to be the mean absolute

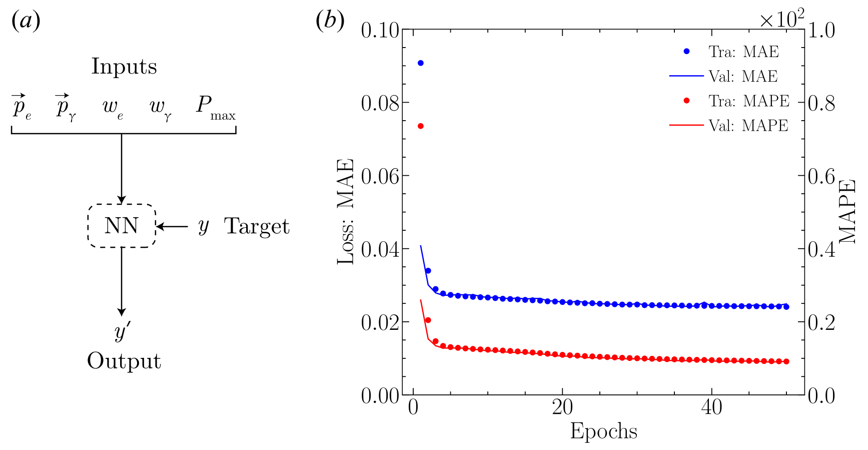

FIGURE 2. (*a*) Depiction of the inputs of the NN during the training process: $\boldsymbol{p}_e$ and $\boldsymbol{p}_\gamma$ are the three momenta of the macroparticles undergoing collisional process, $w_e$ and $w_\gamma$ are their numerical weights and $P_{\max}$ is the maximum probability of interaction in each PIC cell. Finally, the target $y$ constitutes the probability of interaction according to an analytical model and $y'$ is the prediction of the NN. (*b*) Loss score, i.e. the mean absolute error (MAE) and mean absolute percentage error (MAPE), as accuracy for training and validation data as a function of the training epoch for the training data with $p_0 = 25\ m_e c$.

error $= (\sum_{i=1}^{n} |y'_i - y_i|)/n$, where $n$ is the sample size, $y'_i$ is the $i$th predicted value and $y_i$ is the $i$th true value. The loss function characterizes the inaccuracy of the model prediction, and is used to repeatedly update the NN parameters (in our case, using the Adam optimizer with a learning rate $r = 0.001$ (Kingma & Ba 2014)). As more training data are fed to the NN, it learns the target function and the loss score is minimized. A total of 5000 epochs were used in the training. The results of the training process for the dataset with $p_0 = 25\ m_e c$ are shown in figure 2(*b*) for both the training and the validation datasets (corresponding respectively to a randomly selected 80 % and 20 % fraction of the total data, that comprised $\approx 10^7$ Compton scattering events).

To achieve an efficient training process, two techniques were employed. First, the training data were balanced (Bello *et al.* 2021), so that Compton scattering events with different probabilities appeared a similar amount of times. This is a critical step for efficiently learning the probability of interaction via Compton scattering, as it quickly decays with increasing photon momenta in the lepton frame. Second, we employed the multistage training process transfer learning (TL) (Tan *et al.* 2018). This technique trains the NN in stages corresponding to different subsets of the input parameter space $S$, allowing it to train on the full probability distribution while relaxing the condition that the training data have to be identical in all subsets of $S$.

We used three stages of TL. In the first stage, the NN was trained with the dataset containing samples of low-energy interactions between macro-leptons and macro-photons ($p_0/m_e c = 25$), corresponding to high probabilities of interaction. In the second and third stages, the NN was smoothly introduced to higher values of particle energies (and hence smaller probabilities of interaction), through datasets containing high-energy ($p_0/m_e c = 50$) and very-high-energy ($p_0/m_e c = 100$) interactions, respectively. In each TL stage, the particle energy range includes also that of previous TL stages in order to avoid catastrophic forgetting (Kirkpatrick *et al.* 2017), i.e. having the NN lose information about the previous training stage while focusing on the relevant information of the current task.

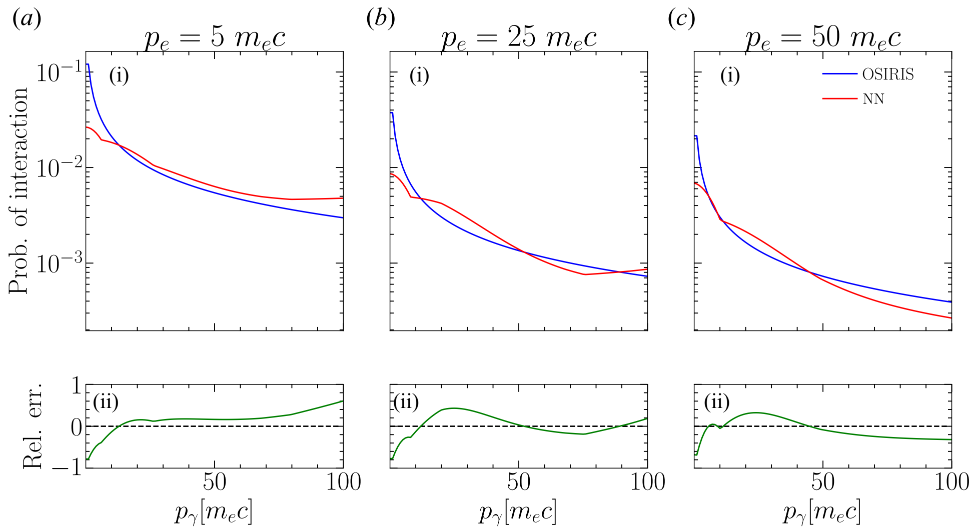

FIGURE 3. Probability of interaction estimated for the no-time-counter method as a function of the incoming photon energy colliding head on with an electron with momentum $p_e = (a1)$ 5, $(b1)$ 25 and $(c1)$ $50m_ec$. The estimate calculated by OSIRIS (blue) and the prediction by NN module for OSIRIS (red) are directly compared and their relative error is shown in $(a2–c2)$.

## 3.3. *Results*

In order to assess the value of our ML-assisted Compton scattering module for PIC codes, we studied: (i) its accuracy, i.e. its ability to correctly capture the physical properties of the collisional process and produce results that match those of analytical approaches, and (ii) its computational performance, to determine whether this method yields a speeding up relative to the implementation described in § 3.1 (and in Del Gaudio *et al.* (2020) in more detail).

The first benchmark corresponded to a comparison between the prediction of the NN and the analytical probability of interaction. This test aimed at explicitly showcasing how ML-based methods can learn and replicate the Compton scattering process. In figure 3(*a–c*), we show the scattering probability between electrons with momenta $p_e/m_ec = 5, 25$ and 50 and photons with a wide range of momenta $p_\gamma$. The good agreement between the two methods results in relative errors smaller than order unity for all tested $p_\gamma$. The NN performs worse in the region of low $p_\gamma$ because training was focused in higher-energy collisions, with each step in the TL scheme moving further into higher energies. To improve this result, another TL training could be done in the region of low photon energy.

To further test the accuracy of our method, we performed a benchmark simulation consisting of an electron beam interacting with a monoenergetic photon gas via inverse Compton scattering (Blumenthal & Gould 1970), which has also been performed with the conventional implementation of Compton scattering in OSIRIS presented in a previous work (Del Gaudio *et al.* 2020).

In figure 4, we show the photon spectra obtained from simulating the interaction of an electron beam of momenta $p_e/m_ec = 25$, 50 and 100 with an initial monoenergetic photon distribution of energy $\varepsilon_\gamma/m_ec^2 = 2.5$ using both the analytical and NN inference of the probability of interaction via Compton scattering. In all simulations (different $p_e$, analytical or NN), we observe a peaked photon momentum distribution developing

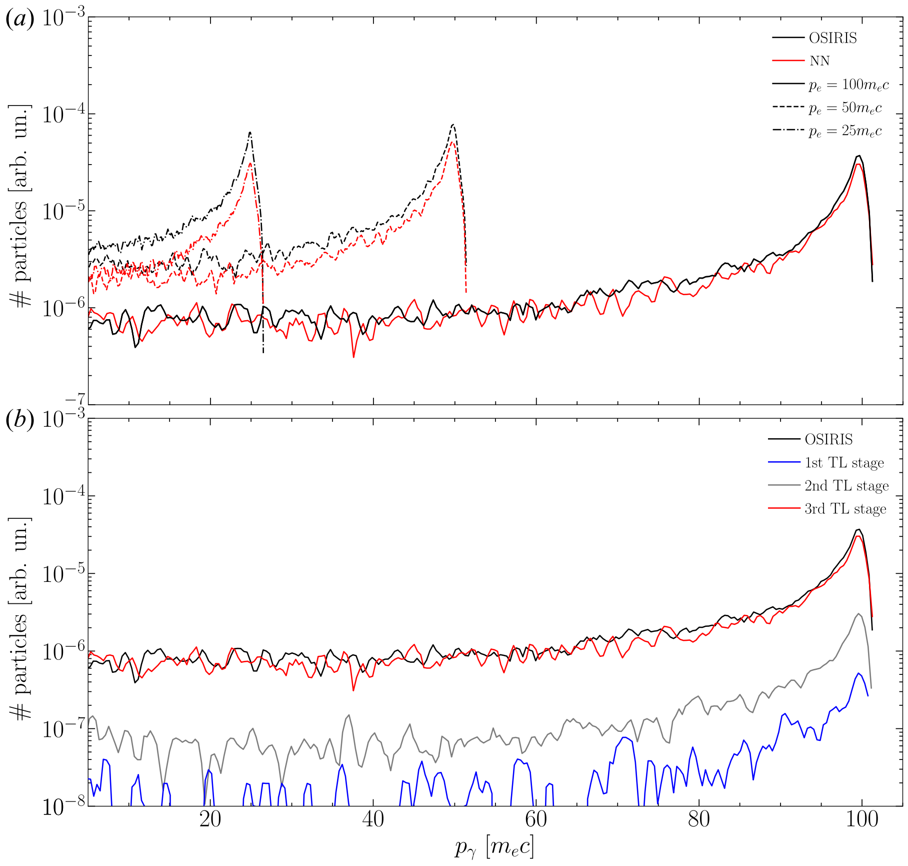

FIGURE 4. (*a*) Scattered photon momentum distribution resulting from inverse Compton scattering with an electron beam with momentum $p_e$. The three peaks correspond to three initial monoenergetic electron beams with $p_e = 25$, 50 and 100 $m_e c$ (identified with dash-dotted, dashed and solid lines, respectively) interacting with a monoenergetic photon gas with energy $2.5 m_e c^2$. For each photon beam energy, the NN module for OSIRIS prediction (red) is plotted against the results from the conventional algorithm (black). (*b*) Scattered photon momentum distribution from inverse Compton scattering between an electron beam with $p_e/m_e c = 100$ and a monoenergetic photon gas with energy $\varepsilon_\gamma/m_e c^2 = p_\gamma/m_e c = 2.5$. Different model predictions at different TL stages are plotted (blue, grey and red) against results obtained with the conventional Compton scattering algorithm in OSIRIS (black).

during the interaction around $p_\gamma \approx p_e$, with an extended tail for $p_\gamma < p_e$. The NN and the analytical result agree within $\sim 10^{-5}$ particle count, i.e. the NN precisely modelled the expected photon spectra for all the three peaks. A small underestimation of the number of collisions is observed for the peaks corresponding to $p_e/m_e c = 25$ and 50, which is a consequence of the underestimation of the probability of interaction for small $p_\gamma$ identified above. Overall, our results show that Compton scattering physics can be accurately described by a NN, and thus that it can be used to replace conventional analytical calculations of the probability of interaction.

In order to illustrate the relevance of TL in the training process, we show in figure 4 the scattered photon momentum distributions for a simulation with $p_e/m_e c = 100$ obtained with NNs trained with a different number of TL stages. We can see a clear convergence

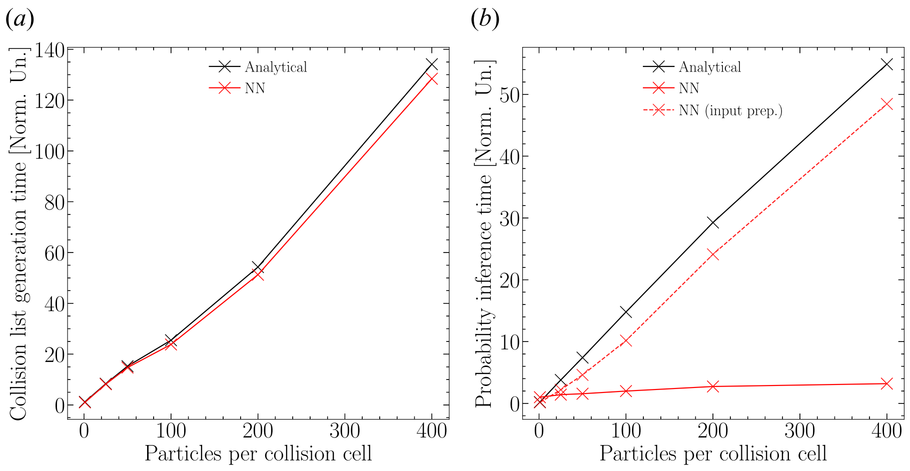

FIGURE 5. (*a*) Comparison of the elapsed time during the complete collision list generation algorithm. (*b*) Comparison of the elapsed time for the inference of the collision probability and for the NN case the input preparation is also plotted. The time is normalized with respect to the time taken for the conventional approach with particles per collision cell = 1.

towards the results obtained with the analytical approach with increasing number of TL stages. These results suggest that simulations can be performed with NNs trained in an already expected electron and/or photon energy range, and extra TL stages can be applied if at some point particle energies exceed the training range. Besides the improvement in accuracy, applying extra TL stages has the advantage of requiring a small number of training samples and thus does not significantly affect the cost of training.

We have also benchmarked the computational performance of our ML-based method. Several simulations with varying number of particles per collision cell were performed and timed. We chose to benchmark our ML-based method against the original algorithm in OSIRIS, which is heavily optimized. The direct comparison between the two methods is plotted in figure 5. In order to guarantee that both methods were compared on the same footing, we timed the computation of the probability of interaction with the NN and with the analytical approach within the same simulation. In each time iteration of these simulations, the analytical probability of interaction was used to determine if each pair of macroparticles should be collided. This ensured that both methods compute the probability for the same number of collisions in each time step, and that they are provided with exactly the same inputs in each computation.

The performance comparison in figure 5(*a*) displays the total collision list generation time for the analytical and NN approaches. It demonstrates that both methods scale linearly with respect to the number of particles per collision cell. The NN-based approach outperforms the optimized Compton module, which becomes more evident with higher number of particles. The whole NN algorithm achieves only slightly faster times than the analytical-based collision list algorithm. The reason behind this is that, even if the NN inference calculation is significantly faster than its analytical counterpart, as seen in figure 5(*b*), the NN algorithm requires that the input of the NN is prepared. This part of the algorithm slows down the whole collision list generation.

## 4. Conclusions

In this work, we have proposed an interface to include ML-based methods in production PIC simulations. This interface allows NNs to be trained using open-source Python packages and deployed in PIC codes written in Fortran. We have presented a proof-of-concept implementation of this interface in the OSIRIS PIC code, and used it to perform simulations where an involved analytical calculation is replaced with a NN of parameters determined in a Python environment. The NN, which was trained with analytical data, computes the probability for two macroparticles to interact via Compton scattering. In order to ensure that the NN accurately captures the dynamic range of the target probability, we have (i) employed multiple stages of TL corresponding to different particle energy ranges and (ii) used a balanced dataset within each stage. We have tested the performance of our approach by studying the photon spectra produced in the interaction between an electron beam and a monoenergetic photon distribution. The ML-based method accurately reproduced the results obtained with a conventional algorithm using the analytical approach. We have also demonstrated that the ML-based method was able to outperform the analytical model in computational efficiency.

In the simulations presented in this work, the NN was trained in a Python environment and its parameters were then saved for later runtime use. However, our implementation including *neural-fortran* also allows training NNs within the PIC simulation and exporting them for use in other environments or other PIC simulations. This flexible architecture offers a promising avenue for future applications of ML-based methods in PIC, not limited only to collisional processes (e.g. Compton scattering, Coulomb collisions), but also applicable to a vast range of physical processes (e.g. high-order quantum electrodynamic processes) and numerical techniques, such as field and particle equation solvers, dynamical load balancing and advanced diagnostics.

## Acknowledgements

We would like to thank E.P. Alves and R.A. Fonseca for fruitful discussions, and for their comments on the manuscript. We are grateful for the comments from the anonymous referee for detailed and insightful review of our work, including suggestions that significantly improved the quality of the results as well as of the manuscript.

*Editor William Dorland thanks the referees for their advice in evaluating this article.*

## Funding

This work was supported by the European Research Council (ERC-2015-AdG grant 695088). C.B. was also supported by FCT (Portugal) (grant PRT/BD/152270/2021). P.J.B. was also supported by FCT (Portugal) (grant UI/BD/151559/2021). F.C. was also supported by FCT (Portugal) (grant PD/BD/114307/2016) in the framework of the Advanced Program in Plasma Science and Engineering (APPLAuSE, FCT grant PD/00505/2012). All simulations and tests were performed in the IST cluster, Lisbon.

## Declaration of interests

The authors report no conflict of interest.

## REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., *et al.* 2015 TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

AGUILAR, X. & MARKIDIS, S. 2021 A deep learning-based particle-in-cell method for plasma simulations. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 692–697. IEEE.

ARBER, T.D., BENNETT, K., BRADY, C.S., LAWRENCE-DOUGLAS, A., RAMSAY, M.G., SIRCOMBE, N.J., GILLIES, P., EVANS, R.G., SCHMITZ, H., BELL, A.R., *et al.* 2015 Contemporary particle-in-cell approach to laser-plasma modelling. *Plasma Phys. Control. Fusion* **57** (11), 113001.

ARROWSMITH, C.D., SHUKLA, N., CHARITONIDIS, N., BONI, R., CHEN, H., DAVENNE, T., DYSON, A., FROULA, D.H., GUDMUNDSSON, J.T., HUFFMAN, B.T., *et al.* 2021 Generating ultradense pair beams using 400 GeV/c protons. *Phys. Rev. Res.* **3** (2), 023103.

BACCHINI, F., RIPPERDA, B., CHEN, A.Y. & SIRONI, L. 2018 Generalized, energy-conserving numerical simulations of particles in general relativity. I. Time-like and null geodesics. *Astrophys. J. Suppl. Ser.* **237** (1), 6.

BELLO, F.A.D., SHLOMI, J., BADIALI, C., FRATTARI, G., GROSS, E., IPPOLITO, V. & KADO, M. 2021 Efficiency parameterization with neural networks. *Comput. Softw. Big Sci.* **5** (1), 1–12.

BIRD, G.A. 1989 Perception of numerical methods in rarefied gasdynamics. *Prog. Astronaut. Aeronaut.* **117**, 211–226.

BLUMENTHAL, G.R. & GOULD, R.J. 1970 Bremsstrahlung, synchrotron radiation, and compton scattering of high-energy electrons traversing dilute gases. *Rev. Mod. Phys.* **42** (2), 237.

BUNEMAN, O. 1993 Tristan. Computer Space Plasma Physics: Simulation Techniques and Softwares.

CHOLLET, F. 2017 *Deep Learning with Python*. Simon and Schuster.

CHOLLET, F. 2018 *Keras: The Python Deep Learning Library*. Astrophysics Source Code Library. ascl–1806.

COMPTON, A.H. 1923 A quantum theory of the scattering of x-rays by light elements. *Phys. Rev.* **21** (5), 483.

CURCIC, M. 2019 A parallel Fortran framework for neural networks and deep learning. In Acm sigplan fortran forum (Vol. 38, No. 1, pp. 421). New York, NY, USA: ACM.

DAWSON, J.M. 1983 Particle simulation of plasmas. *Rev. Mod. Phys.* **55** (2), 403.

DECYK, V.K. 2007 Upic: a framework for massively parallel particle-in-cell codes. *Comput. Phys. Commun.* **177** (1–2), 95–97.

DEL GAUDIO, F., GRISMAYER, T., FONSECA, R.A. & SILVA, L.O. 2020 Compton scattering in particle-in-cell codes. *J. Plasma Phys.* **86** (5).

FONSECA, R.A., MARTINS, S.F., SILVA, L.O., TONGE, J.W., TSUNG, F.S. & MORI, W.B. 2008 One-to-one direct modeling of experiments and astrophysical scenarios: pushing the envelope on kinetic plasma simulations. *Plasma Phys. Control. Fusion* **50** (12), 124034.

FONSECA, R.A., SILVA, L.O., TSUNG, F.S., DECYK, V.K., LU, W., REN, C., MORI, W.B., DENG, S., LEE, S., KATSOULEAS, T., *et al.* 2002 OSIRIS: a three-dimensional, fully relativistic particle in cell code for modeling plasma based accelerators. In *International Conference on Computational Science*, pp. 342–351. Springer.

FUKUSHIMA, K. 1969 Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Trans. Syst. Sci. Cybern.* **5** (4), 322–333.

GRISMAYER, T., TORRES, R., CARNEIRO, P., CRUZ, F., FONSECA, R.A. & SILVA, L.O. 2021 Quantum electrodynamics vacuum polarization solver. *New J. Phys.* **23** (9), 095005.

HAN, J. & MORAGA, C. 1995 The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pp. 195–201. Springer.

HIGGINSON, D.P. 2017 A full-angle Monte-Carlo scattering technique including cumulative and single-event Rutherford scattering in plasmas. *J. Comput. Phys.* **349**, 589–603.

HOCKNEY, R.W. & EASTWOOD, J.W. 1988 *Computer Simulation Using Particles*. CRC Press.

KEMP, A.J, PFUND, R.E.W. & MEYER-TER VEHN, J. 2004 Modeling ultrafast laser-driven ionization dynamics with monte carlo collisional particle-in-cell simulations. *Phys. Plasmas* **11** (12), 5648–5657.

KINGMA, D.P. & BA, J. 2014 Adam: a method for stochastic optimization. arXiv:1412.6980.

KIRKPATRICK, J., PASCANU, R., RABINOWITZ, N., VENESS, J., DESJARDINS, G., RUSU, A.A., MILAN, K., QUAN, J., RAMALHO, T., GRABSKA-BARWINSKA, A., *et al.* 2017 Overcoming catastrophic forgetting in neural networks. *Proc. Natl Acad. Sci.* **114** (13), 3521–3526.

KUBE, R., CHURCHILL, R.M. & STURDEVANT, B. 2021 Machine learning accelerated particle-in-cell plasma simulations. arXiv preprint arXiv:2110.12444.

MILLER, R.H. & COMBI, M.R. 1994 A Coulomb collision algorithm for weighted particle simulations. *Geophys. Res. Lett.* **21** (16), 1735–1738.

NANBU, K. 1997 Theory of cumulative small-angle collisions in plasmas. *Phys. Rev.* E **55** (4), 4642.

OTT, J., PRITCHARD, M., BEST, N., LINSTEAD, E., CURCIC, M. & BALDI, P. 2020 A fortran-keras deep learning bridge for scientific computing. Scientific Programming, 2020.

PARFREY, K., PHILIPPOV, A. & CERUTTI, B. 2019 First-principles plasma simulations of black-hole jet launching. *Phys. Rev. Lett.* **122** (3), 035101.

PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L., *et al.* 2019 Pytorch: an imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates.

SHERLOCK, M. 2008 A Monte-Carlo method for Coulomb collisions in hybrid plasma models. *J. Comput. Phys.* **227** (4), 2286–2292.

SHUKLA, N., SCHOEFFLER, K., BOELLA, E., VIEIRA, J., FONSECA, R. & SILVA, L.O. 2020 Interplay between the Weibel instability and the Biermann battery in realistic laser-solid interactions. *Phys. Rev. Res.* **2** (2), 023129.

TAKIZUKA, T. & ABE, H. 1977 A binary collision model for plasma simulation with a particle code. *J. Comput. Phys.* **25** (3), 205–219.

TAN, C., SUN, F., KONG, T., ZHANG, W., YANG, C. & LIU, C. 2018 A survey on deep transfer learning. In *International Conference on Artificial Neural Networks*, pp. 270–279. Springer.

VRANIC, M., GRISMAYER, T., FONSECA, R.A. & SILVA, L.O. 2016*a* Quantum radiation reaction in head-on laser-electron beam interaction. *New J. Phys.* **18** (7), 073035.

VRANIC, M., MARTINS, J.L., FONSECA, R.A. & SILVA, L.O. 2016*b* Classical radiation reaction in particle-in-cell simulations. *Comput. Phys. Commun.* **204**, 141–151.