Enhancing Hall Effect Thruster Simulations with Deep Recurrent Networks

IEPC-2019-501

Presented at the 36th International Electric Propulsion Conference
University of Vienna • Vienna, Austria
September 15-20, 2019

Maximilian Mörtl[1]
Technical University of Munich, Garching b. München, 85748, Germany

Aaron Knoll[2]
Imperial College London, London, SW7 2AZ, United Kingdom

Vinay Williams[3], Peter Shaw[4] and Vasileios Argyriou[5]
Kingston University, London, SW15 3DW, United Kingdom

and

Jack Zamattio[6], Luca Pugliese[7]
Hephaestus Technologies Ltd., London, EC2M 4NS, United Kingdom

Abstract: **Performing fully-kinetic self-consistent simulations of electric plasma propulsion devices is extremely computationally expensive, and the design process relies largely on experience and experiments. In this work, we investigate the possibility of applying deep learning (DL) in particle-in-cell (PIC) simulations in order to predict plasma dynamics in a 1D Hall Effect Thruster (HET) domain, with the aim of achieving better computational performance whilst retaining high convergence accuracy. We train both a recurrent neural network (RNN) and a Wasserstein GAN on a PIC simulation dataset consisting of 200 spatial nodes and 22.719 simulation time steps, which was created with Imperial College's PlasmaSim software. Our goal was to build a model that is able to predict plasma parameters for the whole domain over many time steps until convergence is achieved, while simultaneously capturing the dynamics involved in the HET simulation. We present the results of the predicted plasma potential distribution based on charge densities as input. Additionally, ion velocity prediction results are given for a single-time step prediction as well as for running our models in a recursive mode, where predicted values are re-fed into the model for making further predictions. Our findings suggest that error accumulation is an inherent risk when computing future simulation steps with Neural Networks, but that these risks can be greatly mitigated by learning the appropriate data distributions through repeated observations.**

[1] MSc Student, Department of Mechanical Engineering, maximilian.moertl18@imperial.ac.uk.
[2] Senior Lecturer, Department of Aeronautics, a.knoll@imperial.ac.uk.
[3] MEng Student, Department of Aerospace and Aircraft Engineering, k1811677@kingston.ac.uk.
[4] Senior Lecturer, Department of Aerospace and Aircraft Engineering, P.Shaw@kingston.ac.uk.
[5] Professor, Department of Networks and Digital Sciences, vasileios.argyriou@kingston.ac.uk.
[6] jack@hphtechnologies.com.
[7] luca@hphtechnologies.com.

Nomenclature

ANN	=	Artificial Neural Network
CNN	=	Convolutional Neutral Network
DL	=	Deep Learning
LSTM	=	Long Short-Term Memory
ML	=	Machine Learning
mse	=	Mean Squared Error
NN	=	Neural Network
PIC	=	Particle-in-Cell
ReLU	=	Rectified Linear Unit
RNN	=	Recurrent Neural Network
WGAN	=	Wasserstein Generative Adversarial Network

I.Introduction

A variety of methods exist for simulating Hall Effect Thrusters (HETs), including fully kinetic Particle-in-Cell (PIC) models[1], hybrid PIC models[2] and fully fluid models[3]. While the fully kinetic PIC approach is generally regarded as the highest fidelity method, computational constraints have generally limited the application of such simulations to 1-dimensional domains without incorporating special techniques for circumventing the time step and spatial step constraints[4,5] intrinsic to these methods. Fully fluid solvers, on the other hand, allow for rapid and efficient computation of the plasma properties over much larger timescales and can be applied to multi-dimensional domains with relatively modest computational resource. However, there are some inherent limitations of fully fluid techniques such as difficulties modelling non-Maxwellian behavior typical for ions within the acceleration and near-field region of HETs. Hybrid PIC models, which treat the ions kinetically and the electrons as a background fluid, are the most widely used technique to simulate HETs. The hybrid approach allows for relatively long duration simulations in two-dimensions (typically radial-axial) and is able to correctly capture the performance characteristics of HETs observed experimentally when suitable corrections are applied for the cross-field electron mobility[3]. Despite the numerous successes of prior work, there is nevertheless the opportunity for Machine Learning (ML) to benefit HET simulations. Ultimately, what is needed is a method that can capture the high-fidelity characteristics of a multi-dimensional fully kinetic PIC model, while also achieving the computational performance of a fully fluid simulation.

A number of prior studies have investigated the application of neural networks to modelling fluid flows[6, 7, 8] and solving simple partial differential equations[9,10]. Notably, in a recent paper by Greve et al.[11], a method is presented that uses an optimization algorithm to tailor the anomalous transport coefficient in a hybrid 2D PIC model, HPHall, developed by Fife and Martinez-Sanchez[12]. This approach is data driven, relying on a reference solution from experiment. In this case, machine learning is used as a tool to tune the parameters of an existing simulation.

In this paper we investigate the extent to which Machine Learning can be used as a complete replacement to a fluid simulation. We propose a novel approach where the plasma characteristics are modelled for a short timescale using a high fidelity fully kinetic PIC solver, and these results are used to train Artificial Neural Networks (ANNs), more specifically Deep Neural Networks (DNNs). The DNNs are then used to predict the plasma behavior at the next time step *k+1* taking as inputs the plasma parameters at time *k*, *k-1*, *k-2*, etc. The plasma behavior is then propagated through time recursively, with the output of the DNNs forming the input at the next iteration. The feasibility of this approach is investigated by applying it to a simple E×B plasma discharge described in Section II, where training data is generated using a fully kinetic 1-dimensional PIC model. The DNNs, described in section III, are used to predict two plasma characteristics: plasma potential and ion velocity. These parameters were selected because they exhibit a fundamental difference in behavior. The plasma potential is influenced by the plasma properties across the entire domain, termed here as a "global parameter", and the ion velocity is primarily influenced by the plasma parameters in its immediate vicinity, a so-called "local parameter". The training and propagation technique, described in section IV, therefore differs for these two parameters. The results, presented in section V, compare the output of the DNNs versus the true output from the PIC model when the solution is propagated recursively.

II.Baseline Particle-in-Cell Simulation

Baseline simulations were performed using a conventional PIC approach. These simulations were implemented using an in-house code called PlasmaSim: a fully kinetic model that resolves the electric field in one dimension (x axis) and velocity in three dimensions (x-y-z). The underlying governing equations are the Boltzmann equation for particle transport and the Maxwell equation for the electric field. The model assumes a static imposed magnetic field, leading to a Poisson relationship for determining the electric field

$$\nabla^2\phi = {}^{-\rho}/_{\varepsilon}, \tag{1}$$

where ϕ is the potential, ρ is the space charge and ε is the permittivity of free space. The charged particle species (electrons and singly charged ions) move according to a Lorentz force

$$m\vec{v} = q\left(\vec{E} + \vec{v} \times \vec{B}\right), \tag{2}$$

where m is the particle mass, v is the particle velocity, q is the particle charge, E is the electric field and B is the magnetic field. The particle motion is solved numerically using a multi-step technique known as the Boris method[13].

The simulation domain is a cartesian rectangular volume with dimensions $x_{max} = y_{max} = z_{max} = 0.05$ m. The domain is partitioned into 200 evenly divided cells along the x axis and the Poisson equation (Eqn. 1) is discretized using a 2nd order central difference approximation as

$$(\phi_{i-1} - 2\phi_i + \phi_{i+1})/dx^2 = -e\left(n_{ion,i} - n_{electron,i}\right)/\varepsilon, \tag{3}$$

where e is the elementary charge, n_{ion} is the ion number density, $n_{electron}$ is the electron number density and the subscript i denotes the cell number. Equation 3 leads to a tridiagonal system when solving simultaneously for the potential within each cell. This system of equations is solved numerically using the open source tridiagonal matrix solver LAPACK (see http://www.netlib.org/lapack/).

Three species of particles were considered: ions, electrons and neutral particles. The neutral particle type was Xenon and the ion particle type was singly ionized Xenon. These particles were grouped into super-particles, where each super-particle represents many real particles with a common position and velocity. For this simulation, 30000 super-particles were used for each species. This gives an average of 150 super-particles per cell per species. Elastic scattering collisions were modelled between the charged particles and neutral particles. At each time step the collision probability was determined for each super-particle depending on the local properties within its cell. Assuming that the neutral particles have a Maxwellian distribution, the collision frequency of a single particle with the background neutrals was calculated using

$$collision\ frequency = n_{neutral}\sigma\sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2 + {c_n}^2}, \tag{4}$$

where $n_{neutral}$ is the number density of neutral particles within the cell, σ is the collision cross section, v is the particle velocity and c_n is the average speed of neutral particles within the cell. The collision cross section was assumed to be constant with energy, with an average value of 1.466×10^{-19} m^2 for electron neutral collisions and 5.863×10^{-19} m^2 for ion neutral collisions. The collision probability within one time step was determined by multiplying the collision frequency by the time step size. A statistical test was performed to determine if the super-particle would undergo a collision. If so, a collision partner (neutral super-particle) was identified within the cell using the DSMC approach[14]. The direction of the relative velocity vector between the particles after collision was selected from a random unit vector using Marsaglia's recipe[15]. The velocity of charged particle after collision was then determined by conserving the momentum and kinetic energy of the colliding particle pair, leading to

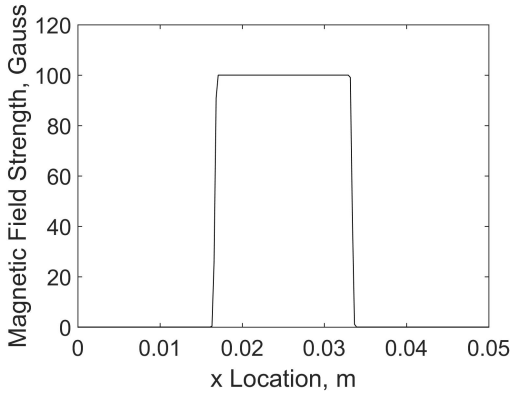

Figure 1. Magnetic field strength (oriented along the y-axis) versus location along the x-axis.

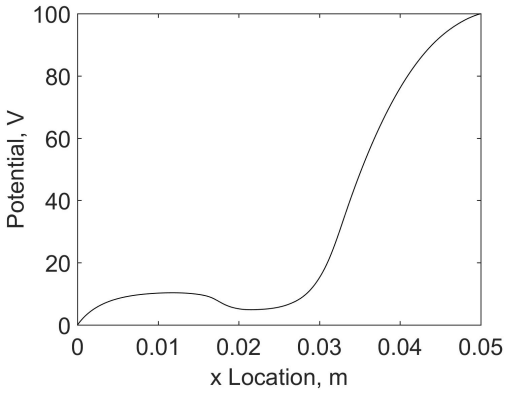

Figure 3. Time averaged plasma potential versus location along the x-axis.

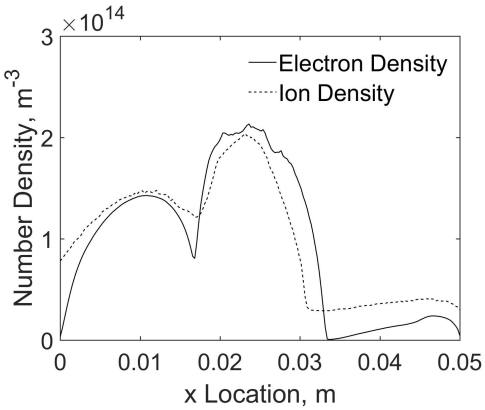

Figure 2. Time averaged electron and ion number density versus location along the x-axis.

$$\vec{v}_{final} = (m\vec{v}_{initial} + m_{neutral}\vec{v}_{neutral} + m_{neutral}\vec{v}_{relative})/(m + m_{neutral}). \quad (5)$$

Boundary conditions were imposed on each face of the rectangular volume. The main axis of the simulation is the x-axis along which the electric field is resolved self consistently at each time step. At the x = 0 and x = 0.05 m faces, the electrons are allowed to freely escape from the simulation domain and are removed from the super-particle set when they cross these boundaries. When the ions cross the x = 0 and x = 0.05 m faces, an ion electron pair is introduced at a random spatial location with velocities sampled from a Maxwellian profile: electrons at 4 eV and ions at 0.1 eV. This boundary condition ensures that the ion density within the simulation domain remains constant with time while maintaining the global charge density consistent with ion production through neutral ionization. The neutral particles undergo specular reflection at the x = 0 and x = 0.05 m faces. The boundary conditions on the y = 0 and y = 0.05 m faces as well as the z = 0 and z = 0.05 m faces are periodic for all particles (i.e. particles crossing these faces are moved to the opposing face and maintain the same velocity). Boundary conditions are imposed for the plasma potential by fixing the voltage of cell 0 to 0 V and cell 200 to 100 V.

A magnetic field orthogonal to the electric field is imposed along the y-axis. The magnitude of the field versus the location along the x-axis is illustrated in **Figure 1**. The field intensity is a rectangular function that is 100 Gauss for x locations between 0.017 m and 0.33 m and 0 Gauss elsewhere.

In the initial state of the simulation, the electrons and ions are uniform with a density of 1×10^{14} m^{-3}. The electrons are initialized with a Maxwellian velocity profile at a temperature of 4 eV and the ions at a temperature of 0.1 eV. The neutral particles are uniform with a density of 1×10^{18} m^{-3} and a temperature of 300 K.

The time step and cell size are constrained by the plasma parameters: plasma frequency, electron gyrofrequency and Debye length. For the conditions in this simulation, the plasma frequency is approximately 90 MHz, the electron gyrofrequency is approximately 280 MHz and the Debye length is approximately 1.5 mm. The simulation timestep is 0.1 ns, which is 35 times smaller than the electron gyroperiod and 111 times smaller than electron oscillation period. The cell size, dx = 0.25 mm, is 6 times smaller than the Debye length.

The simulation results were recorded by averaging the particle properties within each of the 200 cells for 1 ns increments of time. Ten properties were recorded within each cell: the average speed of the electrons, ions and neutrals, the number density of electrons, ions and neutrals, the average velocity of the electrons in the x and z directions, the average velocity of the ions in the x direction and finally the plasma potential. The total duration of the simulation was 25 μs. The

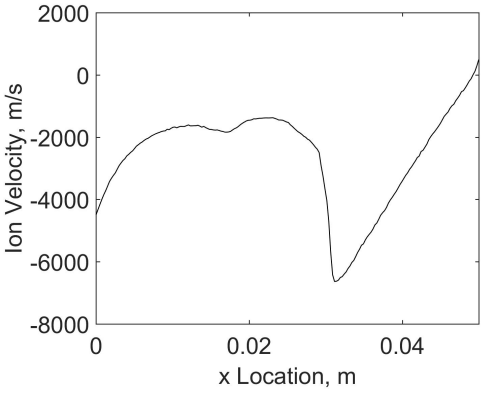

Figure 4. Time averaged ion velocity versus location along the x-axis.

time averaged properties of the plasma potential, electron and ion number density and ion velocity are reported in **Figure 2** through **4** for the full duration of the simulation.

III. Artificial Neural Networks

The field of artificial intelligence was born in the 1950s, when researchers asked the question whether computers can be taught to think in order to perform intellectual tasks that normally can only be done by humans. Machine Learning started to get popular in the 1990s and is now considered the most important subfield of artificial intelligence. ML models are trained by data to perform complex transformations from input data to meaningful outputs. As will be discussed later, ML models may consist of multiple successive layers of increasingly abstract data representations. The number of layers in a ML model is called the depth of the model. A model is considered deep, when more than one or two layers of data representations contribute to the transformation. The field of Deep Learning (DL) is therefore a subcategory of ML suitable for extremely complicated problems. DL models are almost always artificial neural networks, also simply called neural networks (NNs), which are stacks of layers, where each layer consists of a number of neurons[16].

Neural networks have successfully been used in recent years for image classification, weather forecasting, natural language processing or financial data forecasting. The three most common applications are binary classification, multiclass classification and regression tasks. In simple terms, a neural network can be seen as an approximation technology for almost arbitrarily complex functions[17]. The following paragraph will give an overview of the structure of a NN and the components of a NN model.

A. Structure of an Artificial Neural Network Model

As mentioned before, the basic elements of a NN are so-called neurons or nodes, which perform very basic mathematical operations. In a neuron, the input vector X is multiplied by a set of weights and a bias is added. Finally, by multiplying with an activation function the output Y is passed on to the next layer's nodes. The activation function introduces nonlinearity into the system and is a crucial part of the ability to approximate arbitrary nonlinear functions. Popular activation functions that were also used in our models are hyperbolic tangent and sigmoid functions as well as rectified linear units (ReLU).

The typical structure of a fully connected feed forward NN can be seen in **Figure *7***. Fully connected means that each node of one layer is connected to each node of the next layer and feed forward refers to the path of the outputs strictly going just to the next layer but not to other nodes in the same layer. The first layer is called the input layer, the last layer is the output layer and all layers between them are referred to as hidden layers. The number of nodes in the input and output layers are set by the problem that needs to be solved. Number and size of the hidden layers can be chosen depending on the complexity of the learning task.

Besides the neural network itself, a deep learning model requires some more components in order to find a good approximation of the function mapping inputs to outputs. **Figure 5** shows the components of a deep learning model and the algorithmic optimization process of the latter. The input vector X, also called features vector, is fed into the

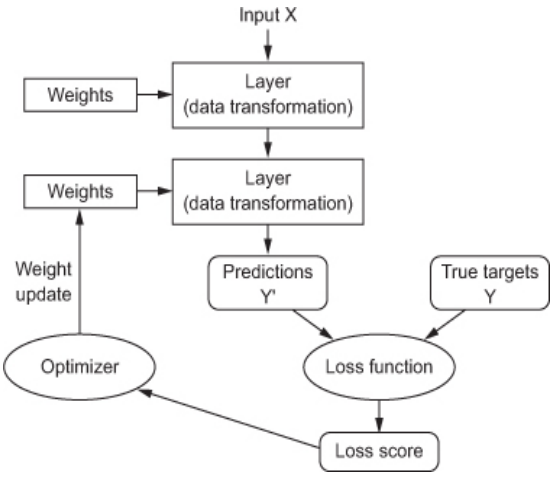

Figure 6. Schematic of the process of building a deep learning model[16].

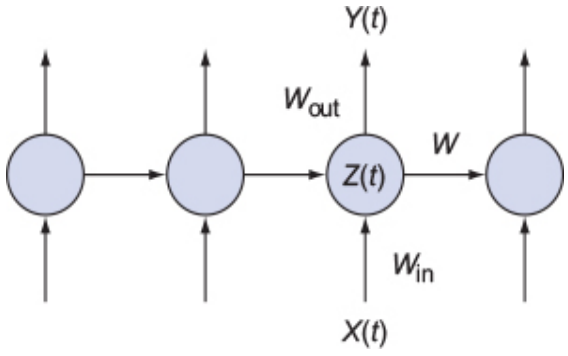

Figure 5: Example of a recurrent neural network that has access to previous states of the network. The two nodes on the left represent the state of the network at previous time steps (t-2 and t-1), which are inputs to the network at respective next time steps[18].

model, in this example a two-layer network, which makes predictions or data transformations depending on the current state of weights and biases. The predictions Y' are compared to the true targets Y within a loss function, for example mean squared error, while the corresponding loss score is passed into an optimizer. The task of the optimizer is to update all weights by calculating the gradient of the loss function with respect to the network's weights. The final goal is to find a converged solution of weights which best represents the data.

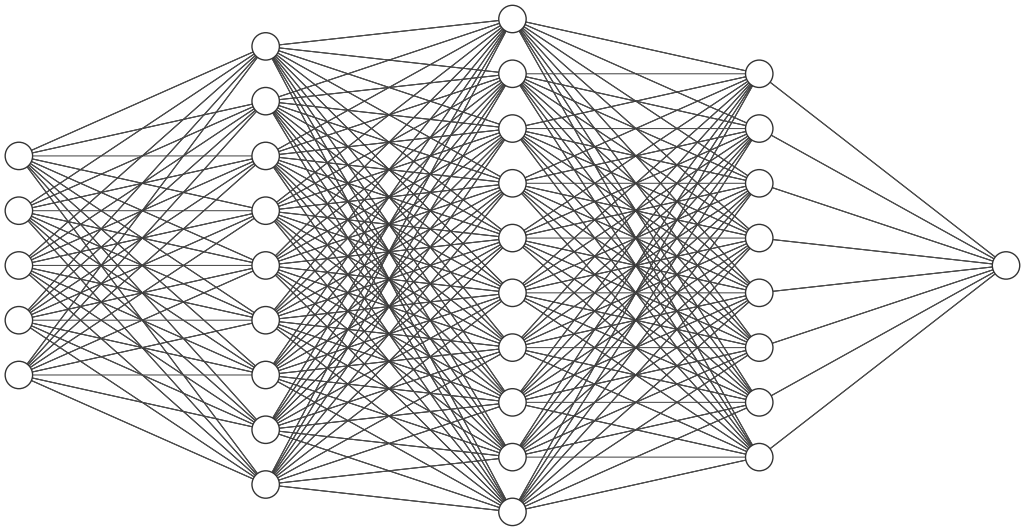

Figure 7: Schematic representation of a fully connected neural network with five inputs, one output and three fully-connected hidden layers.

B. Recurrent Neural Networks

A special family of NNs are recurrent neural networks or RNNs[19] that are particularly good at processing sequential data[17]. In contrast to traditional feed forward NNs which do not have connections within one layer, RNNs use special neurons that have the ability to maintain an internal state giving information on what has already been seen to that point. Basically, this translates into having an internal loop that allows the network to access all previously seen time series examples. An example of an RNN which passes information from previous time steps (nodes on the left representing the network at steps t-2 and t-1) to the current time step t can be seen in **Figure 6**. One of the state-of-the-art RNN models is called Long Short-Term Memory (LSTM)[20] and it allows the network to remember short-term events over the long term. This sophisticated RNN is widely used because it prevents the model of having a problem typical for RNNs called the vanishing gradients problem. The latter occurs when RNNs see long sequences of data but are not able to capture short-term patterns in the data because they are overwritten. LSTMs do not suffer from that problem by having implemented several gates in a cell that decide how much information from the old state should be dropped and how much of the new input data is stored.

C. Wasserstein GAN (WGAN)

Generative Adversarial Networks[17], or GANs, are a type of Deep Neural Networks composed of 2 main elements: the Discriminator 'D' and the Generator 'G'. Discriminator and Generator engage in a Min-Max mutual loss optimization game, a form of adversarial training which has been shown to successfully learn meaningful features from data (by 'D'), and to sample said features in the generation step ('G').

The Wasserstein GAN[21] is a valuable extension to the original GAN architecture, in that it allows for the focusing of the learning process around the concept of distance between probability distributions. This distance is known as Wasserstein Distance or EM Distance (Earth Mover's distance), and we have found it to be particularly well suited as a training loss in physical domains where the goal is to learn systems of partial differential equations being described by the physical parameters interacting with one another.

IV. Methodology

In this section we describe our methodology of building and testing 2 different DNN models: a Long Short Term Memory Network (LSTM), and a modified Wasserstein Generative Adversarial Network (WGAN). All experiments are designed to test the ability to learn the physical behaviors of various simulated parameters and to generate accurate

predictions. We used the open-source software TensorFlow and the open-source neural network library Keras for our models while programming was performed in a Jupyter Notebook in the Python language. All steps in our workflow, from the initial dataset to the training process and final evaluation will be covered. The following list gives an overview of the three different prediction tasks on which we focused on:

1. Prediction of the plasma potential distribution in the 1D domain with charge densities as inputs
2. Prediction of the ion velocity at the respective next time step at every node, based on local plasma parameters of the current time step as inputs
3. Prediction of the ion velocity as described in 2. but running the prediction in a so-called recursive mode meaning that the predicted values are reused for making further predictions

A. Data Structure

As was described in section II, the baseline data was generated by a one-dimensional particle-in-cell simulation. The simulation domain consisted of 200 spatial nodes each for which 10 plasma parameters were calculated during each of the 22.719 time steps. Together with the magnetic field, this gives an array of the size [22719, 200, 11] as initial input data structure. A summary of all plasma parameters and the specific ones used for the different prediction tasks can be found in **Table *1***. To get a better idea of how the data looks like, **Figure *8*** gives a colormap for both the ion velocity (uix) and the plasma potential (potential) for all 200 nodes and 22.719 time steps of the simulation. Also, in order to prevent confusion, data coming from the particle simulation is called "original" or "truth", whereas the output of our models is called "predicted".

Plasma Parameter	1. Potential prediction	2. ion velocity prediction
Magnetic field (By)		
Plasma potential (potential)	x	x
Average speed of electrons (Ce)		x
Average speed of ions (Ci)		
Average speed of neutrals (Cn)		
Number density of electrons (ne)	x	x
Number density of ions (ni)	x	x
Number density of neutrals (nn)		
Average electron x-velocity (uex)		
Average electron y-velocity (uey)		
Average ion x-velocity (uix)		x

Table 1: Summary of all plasma parameters and the specific ones used in the prediction models.

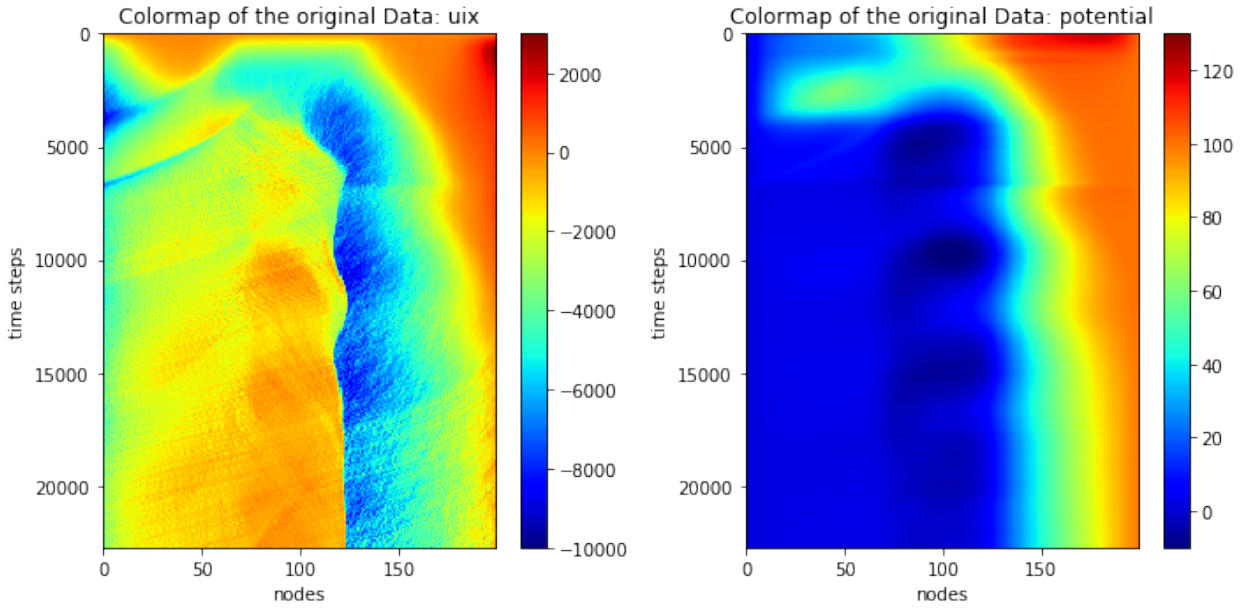

Figure 8: Colormap of the ion velocity (uix) and plasma potential (potential) of the simulation data.

B. Pre-processing

The preprocessing steps for the raw input data from the particle simulation which initially is represented by a three-dimensional array of size [time steps, nodes, plasma parameters] are described in this section. Since the data is the output of a self-consistent simulation there were no missing data points or erroneous values that should have been corrected. The plasma parameters span over a huge range of magnitudes, for example the electron number density reaches values of up to 10^{14}, whereas the ion velocity gets as low as -10^{5}. Neural networks perform best when the input values are in somewhat comparable ranges because the gradient descent optimization converges much faster. Therefore, all values were scaled between 0 and 1 based on their respective minimum and maximum values.

The whole process of building a DL model involves two separate phases. First, the training phase in which the NN weights are adjusted by the optimization algorithm based on the loss score between predicted and true output values. Secondly, the testing phase after the model is readily trained, the network's weights are fixed, and predictions can be made on formerly unseen data. A common problem of NNs is called overfitting and describes the phenomenon when the model is trained to perfectly fit the training data but is not able to generate accurate predictions on unseen input data, which is also known as the ability to generalize.

The solution to this problem is to split the dataset into three separate parts, namely the training, the validation and the test dataset. As the names suggests, the training set is used by the model during the training phase to update its weights, whereas the validation set serves the purpose of having a measure of the ability to generalize, the validation loss, which is calculated after each training epoch. One training epoch is defined as one complete presentation of every training data sample to the NN. In order to converge, the NN needs to be trained over many epochs. At the point where the validation loss is not decreasing anymore but the model is still getting better loss scores on the training set, the training phase ends due to overfitting. Typical data splits for training/validation/test of 70/15/15 or 60/20/20 were chosen in most of our models.

Another pre-processing step that was performed for the LSTM model running single-time step predictions was to randomize the data points before splitting into the different datasets was done.

C. LSTM Model - Input, Output and Architecture

For the case where we tried to predict the plasma potential distribution, the input values comprised electron and ion number densities for every node in the domain. Since the potential is a "global" field function, all number densities at one time step were taken into account. The desired output was then defined as the plasma potential in every node for the next time step. The structure of the LSTM model consisted of an input layer with 400 nodes, the first hidden LSTM layer with 500 nodes, a second hidden LSTM layer with 500 nodes and a dense layer with 200 nodes. The sigmoid activation function was used.

Input and output data configuration for the ion velocity prediction was set up differently. We chose to train the model on a small spatial subset of the full dataset because of two reasons. First, as the plasma parameters are not dependent on the physical state of the plasma at locations far away from the node of interest, not all 200 nodes and respective parameters were taken as inputs. Also, in the traditional finite difference solving algorithms only the parameters of adjacent nodes are used for the calculation. Secondly, it was computationally extremely expensive to feed the network an input vector of size 1000 and making predictions on 200 ion velocity values. Hence, we experimented with feeding information of 3 to 51 adjacent nodes as input but chose a number of 5. Additionally, since LSTM networks expect time series data, we trained the model on randomly selected sequences of 100 time steps. In summary, the input array consisted of 100 time steps for 5 nodes with 5 plasma parameters each, leading to the input dimension of [100, 25]. For each sample of 100 time steps, the prediction task was to accurately forecast the ion velocity at the middle node for the next time step. The aim of this process was that the model was able to learn the underlying physics on a small scale by showing it changes in plasma parameters over time and space. The NN model's structure was set up as follows: after the 25 input nodes, three stacked layers containing 800, 256 and 128 LSTM nodes respectively, and a dense layer of size 1 were created. The activation functions of the layers were again sigmoid functions.

D. LSTM Model - Supervised Training Process

In the plasma potential prediction model's training process, the RMSprop optimizer algorithm[22] was used with an initial learning rate of 0.001 and an automatic decay during training to 0.0001. The model was trained for 20 epochs with 200 steps each. Validation loss scores were monitored for executing early stopping to prevent the model from overfitting. For calculating the training and validation loss, the mean squared error (mse) function was used.

The Adam optimizer[23] was used in all ion velocity prediction models and the latter were trained for 100 epochs and 300 steps per epoch. The same mse loss was monitored and the validation loss was again taken as a measure when overfitting occurred in the model.

E. LSTM Model - Performance Evaluation Metrics

Each LSTM model's performance on the test dataset was calculated by the previously mentioned mse loss function. The plasma potential model achieved a loss value of 7.9×10^{-5}, while the ion velocity prediction achieved 3.19×10^{-5}. However, a better indicator of how well the model performs on unseen data is to look at the predicted values for all nodes and many time steps, which will be presented in the next section alongside the true simulated data.

Additionally, when the ion velocity LSTM model is used in a recursive mode the performance evaluation is also done by comparing the predicted to the simulated data.

F. WGAN Model - Input, Output, and Architecture

After experimenting with LSTMs, we have formulated an approach using a modified WGAN to create a 1-D Plasma Simulation Acceleration model, focused on recursive prediction. The dataset consists of 22.719 timesteps and 200 nodes (cells) with each node having 5 parameters being considered. This data was first divided into a train set consisting of 18000 timesteps and a test set consisting of 5000 timesteps. Corresponding ground truth values of +1, +5 and +10 timesteps lookahead were also created.

The input ground truth to the model can be controlled by setting a flag value. The model used is a modified Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP). This GAN architecture helps stabilize the training of a normal GAN with faster convergence and handles the problems of exploding/vanishing gradients and mode collapse robustly.

As GANs have been widely successful in solving Image Synthesis problems, we first converted the time series problem into an Image Synthesis problem by converting each timestep into a greyscale image of 20x50x1 dimensions. This approach was led by notions of computational efficiencies to be gained in more advantageous data representations[24]. These images are fed as input to the model in batches with each batch containing random timesteps and are shown in **Figure *9*.**

Figure 9: Greyscale Image Batch of Plasma Parameters, seeking more Efficient Data Representations.

G. WGAN Model - Adversarial Training Process

At training time, the WGAN Generator ('G') is fed real PIC simulation data from which it will generate the predictive lookahead, and the WGAN Discriminator ('D') is fed the Generator's ('G') output along with the ground truth value; the training loss is the Wasserstein Distance between probability distributions between plasma parameters inherently observed in the data. Then, at inference time, an accelerated simulation can be run by providing the real value of the 1st desired timestep, at which point the Generator ('G') will produce the first lookahead prediction which will recursively be taken again as input for Generator ('G') in the next step, so on. After the WGAN model has been trained on the train set, the inference is performed on the test set which are unseen values for the model. The whole dataset is tested from timesteps 1 to 22.719 by utilizing a K-Fold Cross Validation, thereby ensuring the Plasma-GAN is never asked to infer (test set) any of the data it has been trained on (train set).

The predictions obtained from the Generator in the form of greyscale images are then converted back into form of the original timesteps by reshaping them back into 200x5 values. These are then plotted alongside the ground truth values to check the result quality of the simulation, to easily compare outputs from other models studied in this work (LSTM).

H. WGAN Model - Performance Evaluation and Computational Speedup

In evaluating the effectiveness of the WGAN model, we focused on dynamical evolution of parameter states to a realistic physical representation compared to the plasma PIC simulation, as well as computational speedup over the original PIC simulation.

V.Results

A. LSTM Model - Field Function Prediction – Plasma Potential

The results of the LSTM model's prediction of plasma potential for all of the 22,719 time steps can be seen in **Figure *10***. All values were rescaled to the original potential units. It can be seen that when the LSTM model is trained on randomly selected 60% of the whole dataset, the prediction accuracy is good. Note that the model takes as input only the charge densities of electrons and ions in every cell and gives as output the corresponding plasma potential. Therefore, the full step of calculating the plasma potential which is traditionally done with a finite difference method in the PIC simulations can be performed by a neural network. However, more simulation datasets should be tested with this model and especially cross-case testing between multiple simulation runs with different boundary conditions and thruster configurations should be performed in a next step.

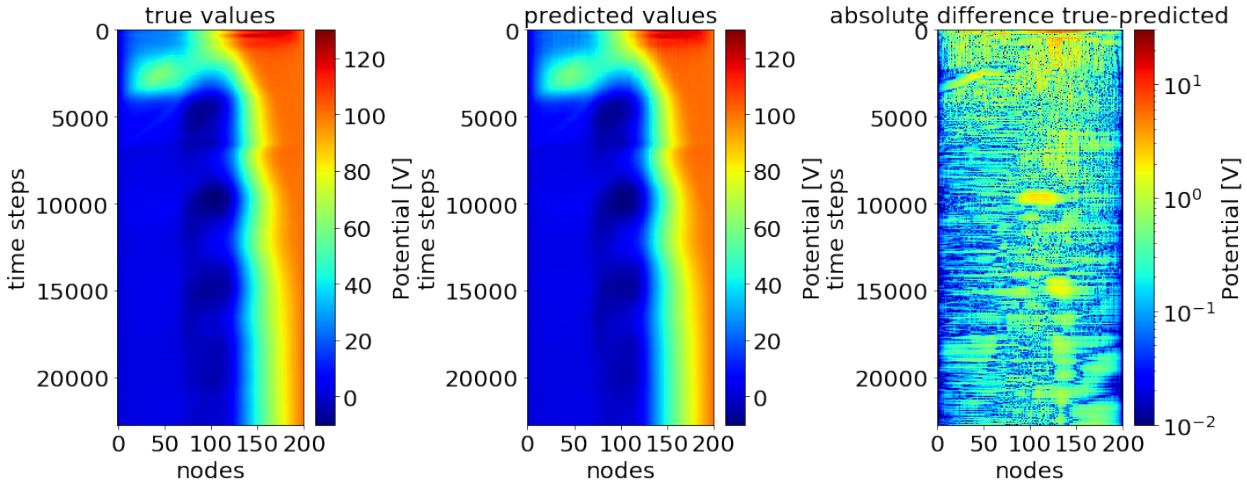

Figure 10: LSTM Model - True and predicted plasma potential values for all time steps as well as the absolute difference.

B. LSTM Model - Local Plasma Parameter Prediction – Ion Velocity

The next plasma parameter that we attempted to predict with the LSTM model was the ion velocity, uix. The results for predicting only the next time step's uix can be seen in **Figure *11***. We call the latter prediction process "normal mode" because for every time step's prediction the input data comes from true simulated data only. On the other hand, we say the prediction is running in "recursive mode" when the predicted uix value is reused for predicting the next one, i.e. the model is operating in a self-feeding loop. The predicted results for the normal mode are very encouraging and suggest that the LSTM model is able to accurately predict the next time step's uix value at a specific node. However, since for every prediction the input values are taken from the ground truth data and considering that the differences in plasma parameter values between two time steps are not really big, the small errors are not really proof that the model has picked up the fundamental physics involved in the simulation. Because of that, we moved on to operating the LSTM model in recursive mode and see for how many time steps the prediction tracks the simulation well. It is important to note that for now only the ion velocity values are predicted and reused for further predictions, whereas all other four plasma parameters involved in the prediction are still taken from the true simulation data. The idea behind this approach is to first check the ability of the LSTM model to predict one parameter in recursive mode and if successful, proceeding by predicting all parameters in recursive mode. **Figure *12*** shows the results of uix prediction when the model is used in a recursive mode. It gets clear from this figure that after just a few prediction steps the uix values start to diverge with this approach. The error accumulation over time suggests that the LSTM model has no mechanism of bringing the predicted values back to true values after an error occurred. It can therefore be concluded that this model was not able to learn the underlying physical phenomena which govern the plasma state over time.

Two questions arise from our findings. First, is it possible to find a better training process and NN architecture which does not suffer from error propagation by having a lot more training data so that the model will learn the physics behind the simulation and will therefore only make predictions consistent with it? Or, secondly, is the error accumulation over time an unavoidable problem which can be optimized so that more recursive time steps into the future may be performed but after which some traditional simulation techniques need to be used in order to get to a fully converged physical solution?

In seeking to answer these questions, we later deployed a modified Wasserstein GAN model (WGAN) as a comparative NN architecture to the LSTM model.

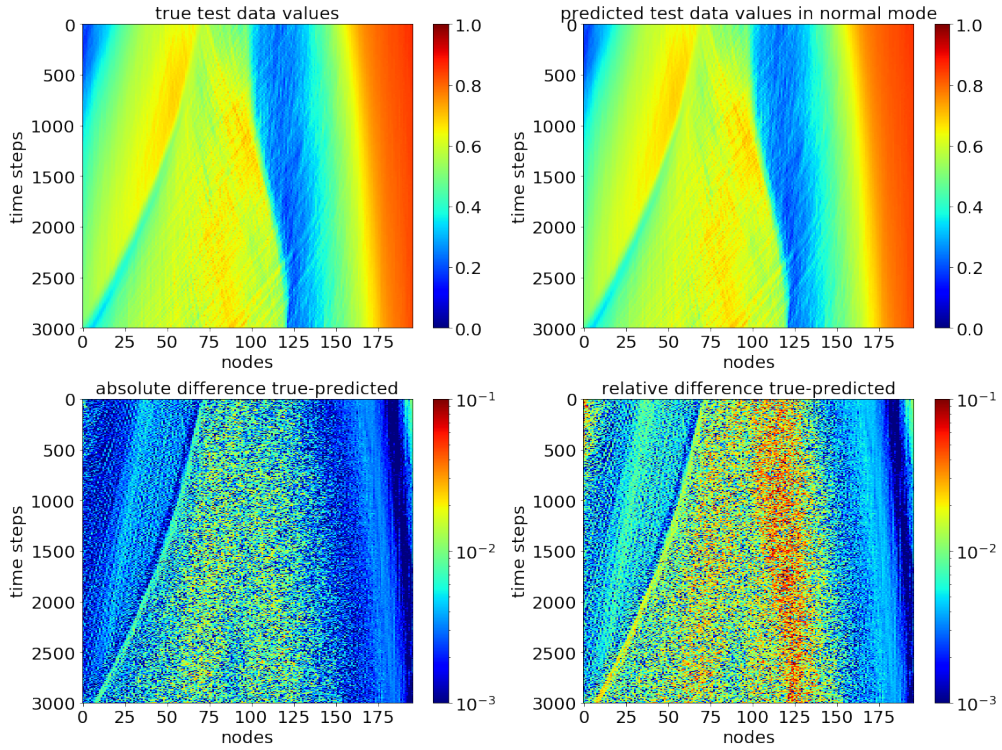

Figure 11: LSTM Model - Results of the ion velocity prediction (top right) compared to the true simulation data (top left) when the LSTM model is used in normal mode. Absolute (bottom left) and relative (bottom right) difference between true and predicted values are also given.

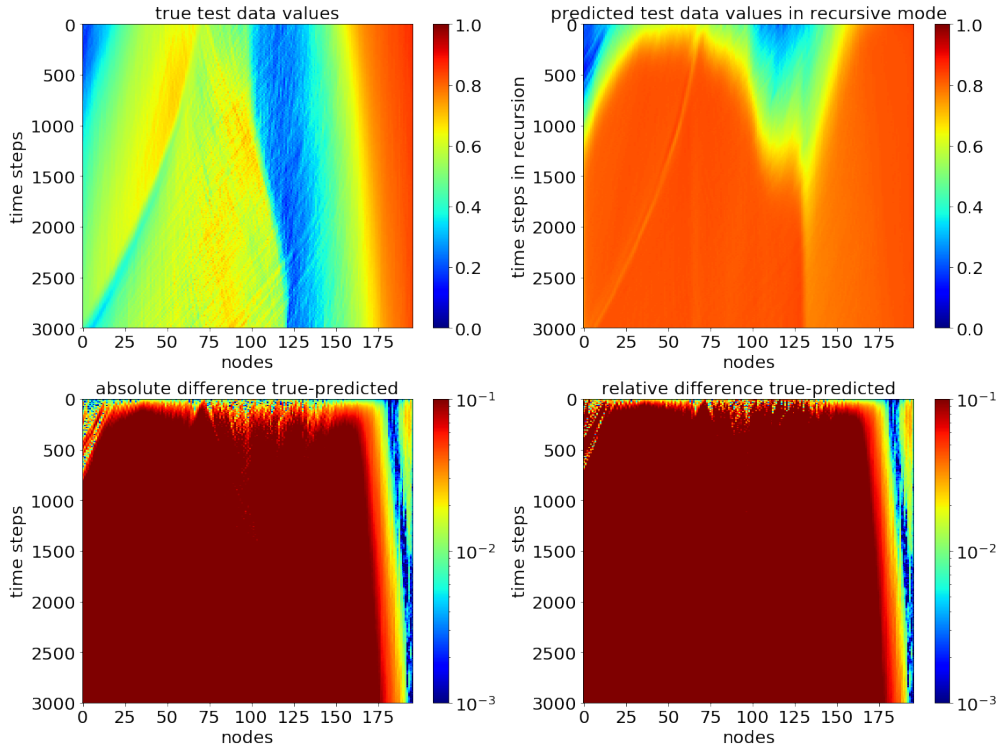

Figure 12: LSTM Model - Results of the ion velocity prediction (top right) compared to the true simulation data (top left) when the LSTM model is used in recursive mode. Absolute (bottom left) and relative (bottom right) difference between true and predicted values are also given.

C. WGAN Model - Local Plasma Parameter Prediction - Ion Velocity

Finally, we focus our attention on recursive prediction of the uix parameter and experiment with a modified WGAN model, or PlasmaGAN. We tuned this model to learn the latent probability distributions displayed in the plasma simulation data and utilizing the Wasserstein distance to discriminate between plasma parameters. This approach has shown great promise in learning the general physical properties for the uix (ion velocity) parameter, as we observed very realistic evolution of ion velocity distributions. It appears that at least the major physical descriptors of the system of equations described by the plasma PIC simulation have been absorbed by the PlasmaGAN, as can be seen by the realistic trend in the generated samples being outputted by the model

It should also be noted that only 1 ground truth simulation datapoint has been fed to this model, which then generates all future timesteps towards a convergence or equilibrium which is similarly displayed by the plasma PIC simulations over the whole of its 22.719 timesteps as can be seen in **Figure *13*.**

These results give us confidence in the method, as an incorrect apprehension of physical descriptors could not have generated a conversion of parameter values, in this case ion velocities, over as many timesteps as seen below.

Lastly, considerations around the computational advantages which can be gained with this approach must also be made. We have noted a computational speedup of about 4 orders of magnitude compared at inference time with our method compared to the PIC numerical approximation of these PDEs, while retaining all basic plasma parameter distributions after several thousands of timesteps, if at a loss of finer parameter resolution.

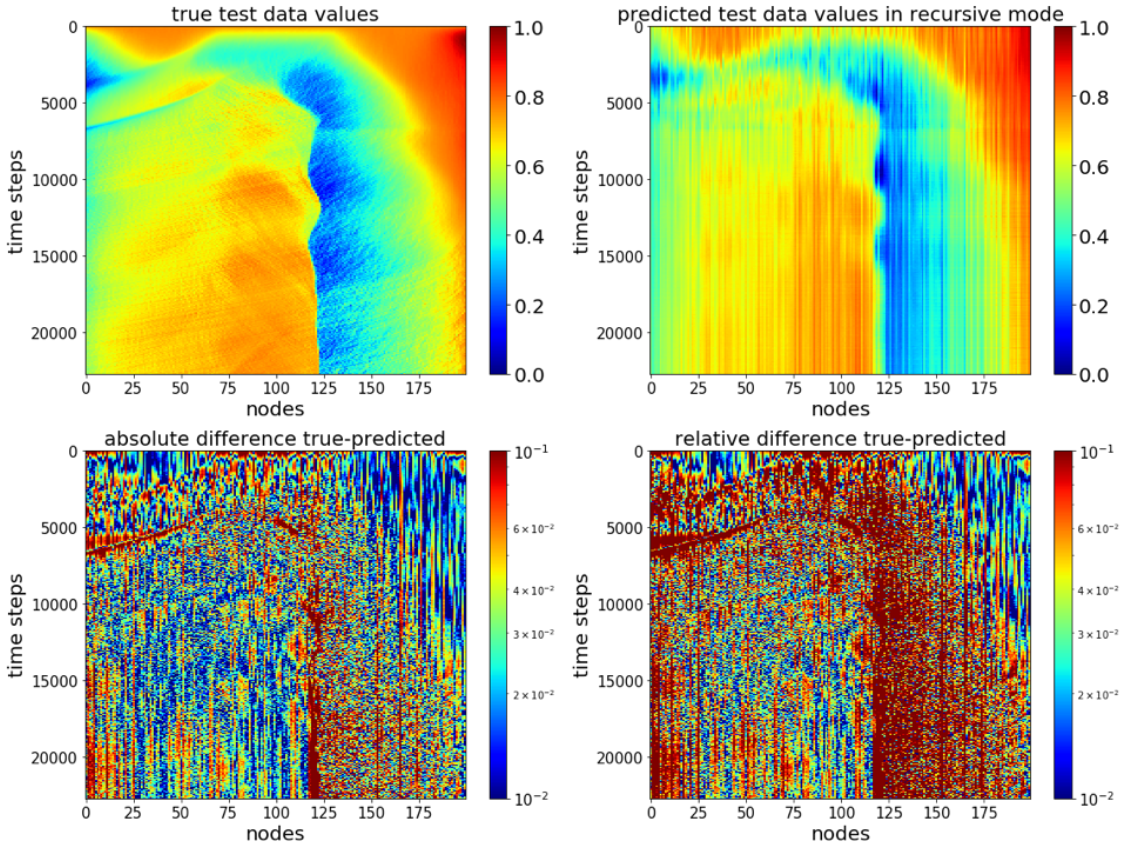

Figure 13: WGAN Model - Results of the ion velocity prediction (top right) compared to the true simulation data (top left) when the model is used in recursive mode by the PlasmaGAN (WGAN). Absolute (bottom left) and relative (bottom right) difference between true and predicted values are also given.

VI.Conclusion

This study has shown early promise in the capabilities of artificial neural networks to predict complex plasma behaviors at a future point in time based on a limited time history of the plasma parameters. The neural network was trained using data from a fully kinetic PIC model and, when supplied data directly from this model, was able to predict the plasma potential and ion velocity to very good precision. However, a significant limitation in the approach was encountered when using the LSTM model recursively: with the output of the LSTM network being supplied as an

input for the next iteration. In this scenario, the error was found to accumulate rapidly leading to a divergence between the true and predicted results within less than 100 iterations. It should be noted that this work is still preliminary, and it is still too early to say whether this is a fundamental limitation or whether workarounds can be found to mitigate the issue. On the other hand, utilizing a modified WGAN architecture we have created PlasmaGAN, a system capable of learning the general physical dynamics for ions described by the Boltzmann equation in conjunction with Maxwell's equations. The system is capable of acting as a deep learning-based simulation accelerator, whereby a single timestep of real data is fed and a valid, physically consistent approximation of the full PIC plasma simulation is produced at a computational speedup of 4 orders of magnitude. We believe that this data-driven method will be useful in allowing greatly accelerated first principles validation of various plasma devices such as Hall Effects Thrusters, validation which so far has been constrained by the computational scaling limitations of classical numerical methods.

References

[1]M. Hirakawa and Y.Arakawa, "Particle simulation of plasma phenomena in Hall thrusters," IEPC-95-164, 1995.

[2]Fife, John Michael, "Two-dimensional hybrid particle-in-cell modeling of Hall thrusters," Diss. Massachusetts Institute of Technology, 1995.

[3]Lam, Cheryl M., Aaron K. Knoll, Mark A. Cappelli, and Eduardo Fernandez, "Two-dimensional (z-θ) simulations of Hall thruster anomalous transport," *In Proceedings of the 31st International Electric Propulsion Conference,* 2009.

[4]Szabo, James, Noah Warner, Manuel Martinez-Sanchez, and Oleg Batishchev. "Full particle-in-cell simulation methodology for axisymmetric Hall effect thrusters," *Journal of Propulsion and Power* 30, no. 1, 2013: 197-208.

[5]Cho, Shinatora, Kimiya Komurasaki, and Yoshihiro Arakawa, "Kinetic particle simulation of discharge and wall erosion of a Hall thruster," *Physics of Plasmas* 20, no. 6, 2013, 063501.

[6]Thuerey, Nils, Konstantin Weissenow, Harshit Mehrotra, Nischal Mainali, Lukas Prantl, and Xiangyu Hu, "Well, how accurate is it? A Study of Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations," arXiv preprint arXiv:1810.08217, 2018.

[7]Tompson, Jonathan, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin, "Accelerating eulerian fluid simulation with convolutional networks," *In Proceedings of the 34th International Conference on Machine Learning*-Volume 70, pp. 3424-3433. JMLR. org, 2017.

[8]Farimani, Amir Barati, Joseph Gomes, and Vijay S. Pande, "Deep learning the physics of transport phenomena," arXiv preprint arXiv:1709.02432, 2017.

[9]Chiaramonte, M., and M. Kiener, "Solving differential equations using neural networks," *Machine Learning Project*, 2013: 1.

[10]Dockhorn, Tim, "A Discussion on Solving Partial Differential Equations using Neural Networks," arXiv preprint arXiv:1904.07200, 2019.

[11]Greve, C. M., K. Hara, R. S. Martin, D. Q. Eckhardt, and J. W. Koo, "A data-driven approach to model calibration for nonlinear dynamical systems," *Journal of Applied Physics* 125, no. 24, 2019: 244901.

[12]Fife, John Michael, "Hybrid-PIC modeling and electrostatic probe survey of Hall thrusters," PhD diss., Massachusetts Institute of Technology, 1998.

[13]Boris, J. P., "Relativistic plasma simulation-optimization of a hybrid code," *In Proceeding of Fourth Conference on Numerical Simulations of Plasmas*, 1970.

[14]Bird, Graeme A., and J. M. Brady, "Molecular gas dynamics and the direct simulation of gas flows," Vol. 5. Oxford: Clarendon press, 1994.

[15]Marsaglia, George.,"Choosing a Point from the Surface of a Sphere," *The Annals of Mathematical Statistics* 43 (2): 645–46, 1972.

[16]Chollet, Francois, "Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek," MITP-Verlags GmbH & Co. KG, 2018.

[17]Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[18]Shukla, Nishant. *Machine learning with TensorFlow*. Manning Publications Co., 2018.

[19]Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." *Cognitive modeling* 5, no. 3, 1988: 1.

[20]Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9, no. 8 (1997): 1735-1780.

[21]Arjovsky, Martin, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks." In *International conference on machine learning*, pp. 214-223. 2017.

[22]Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky. "Lecture 6a overview of mini-batch gradient descent (2012)." *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture*, 2012.

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

[24]Worrall, Daniel E., Stephan J. Garbin, Daniyar Turmukhambetov and Gabriel J. Brostow, "Harmonic Networks: Deep Translation and Rotation Equivariance," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016: 7168-7177.