mathematics

MDPI

Article

Assessment of Machine Learning Methods for State-to-State Approach in Nonequilibrium Flow Simulations

Lorenzo Campoli *, Elena Kustova and Polina Maltseva

Department of Fluid Mechanics, Saint Petersburg State University, 7/9 Universitetskaya nab., 199034 St. Petersburg, Russia; e.kustova@spbu.ru (E.K.); st079718@student.spbu.ru (P.M.)
* Correspondence: l.kampoli@spbu.ru

Abstract: State-to-state numerical simulations of high-speed reacting flows are the most detailed but also often prohibitively computationally expensive. In this work, we explore the usage of machine learning algorithms to alleviate such a burden. Several tasks have been identified. Firstly, data-driven machine learning regression models were compared for the prediction of the relaxation source terms appearing in the right-hand side of the state-to-state Euler system of equations for a one-dimensional reacting flow of a N_2/N binary mixture behind a plane shock wave. Results show that, by appropriately choosing the regressor and opportunely tuning its hyperparameters, it is possible to achieve accurate predictions compared to the full-scale state-to-state simulation in significantly shorter times. Secondly, several strategies to speed-up our in-house state-to-state solver were investigated by coupling it with the best-performing pre-trained machine learning algorithm. The embedding of machine learning algorithms into ordinary differential equations solvers may offer a speed-up of several orders of magnitude. Nevertheless, performances are found to be strongly dependent on the interfaced codes and the set of variables onto which the coupling is realized. Finally, the solution of the state-to-state Euler system of equations was inferred by means of a deep neural network by-passing the use of the solver while relying only on data. Promising results suggest that deep neural networks appear to be a viable technology also for this task.

Keywords: machine learning; neural network; state-to-state kinetics; vibrational relaxation; chemical reactions

MSC: 76-10

Citation: Campoli, L.; Kustova, E.; Maltseva, P. Assessment of Machine Learning Methods for State-to-State Approach in Nonequilibrium Flow Simulations. *Mathematics* **2022**, *10*, 928. https://doi.org/10.3390/math10060928

Academic Editors: Andrey Gorshenin, Mikhail Posypkin, Vladimir Titarev and Ioannis G. Tsoulos

Received: 19 January 2022
Accepted: 9 March 2022
Published: 14 March 2022

1. Introduction

Numerical simulation of non-equilibrium flows receives much attention in computational fluid dynamics due to their importance for aerospace and low-temperature plasma applications. The state-to-state (STS) approach based on the assumption of fully coupled vibrational-chemical kinetics and fluid dynamics has many advantages with respect to commonly used one-temperature (1T) and multi-temperature (MT) models, since it is capable of describing strong deviations from equilibrium and arbitrary vibrational distributions. As it was shown in [1–4], the state-to-state formulation allows to obtain the best agreement with experimental data. The price for such high level of detail description is the computational cost, which is often prohibitive. This led several researchers to investigate various energy binning approaches, in the framework of traditional CFD (Computational Fluid Dynamics) and DSMC (Direct Simulation Monte Carlo) methods [5–11], as well as MPI-CUDA approaches [12].

In recent years, we have observed a remarkable growth of research interest in machine learning technologies, fostered by the coexistence of several factors, such as the increase of the amount of generated data, the improvements in hardware manufacturing, the reduced cost of computational resources, data storage and transfer, the smarter algorithms' development, the wide-spreading of open source software, and significant and ongoing investment by industry.

Typical machine learning applications regard classification, regression, clustering, or dimensionality reduction tasks. In the early 1990s, neural networks were applied, for example, in [13], to interpolate the rate coefficients of vibrational quantum exchange processes between nitrogen molecules in a state-to-state simulation by means of a multilayer neural network as a function of quantum numbers. In [14], a neural network was used to interpolate the velocity coefficients of a chemical network for the kinetics of an excimer laser, as a function of various parameters, in the state-to-state view for the HCl molecule. In [15] neural networks were trained to emulate the solution of the inverse problem of the Boltzmann transport equation and consequently obtain the excitation cross sections of various molecular states starting from electron transport quantities as functions of the electric field. Updated methods have been successfully applied more recently, for example in [16,17]. Machine learning algorithms have also been used in fluid mechanics problems, for example, in experimental data processing, shape optimization, turbulence closure modeling, control and other traditionally intractable problems [18–21].

In this paper, we explore the possible ways of reducing the computational costs of state-to-state non-equilibrium flow simulations using machine learning methods.

In order to estimate the importance of different terms to the total computational cost, a simple overall profiling analysis was conducted for a typical two-dimensional simulation of a hypersonic non-equilibrium viscous reacting flow across a blunt body geometry. Details of input parameter settings and profile timings are given in Table 1. As expected, the kinetic and transport modules are the most expensive. It is worth noting here that for the computation of the transport properties, the Gupta [22] model has been adopted and this is the reason why its cost is moderate. Nevertheless, as soon as state-to-state models are employed for transport processes as well, the computational cost will be comparable or greater than the kinetic one.

Table 1. Parameter settings and timings of one iteration (0.814 s) for a typical two-dimensional simulation of a hypersonic non-equilibrium viscous reacting flow across a blunt body geometry.

Key	Value
Flow Type	Navier_Stokes
Species	N_2, O_2, NO, N, O
Kinetic	STS
Transport	Gupta
Gas Model	Nonequ_Gas
Simulation_Type	2D_AXI
Pressure	2.0
Temperature	195.0
Velocity	11,360.0
Mass Fractions	N_2: 0.79, O_2: 0.21
Process	**Time**
Exchange	0.00%
Update	0.26%
ComputeConvectionFlux	1.51%
ComputeDissipationFlux	21.93%
ComputeSourceTerms	76.27%
ComputeLocalResidual	0.01%

Consistently, it is possible to individuate several areas, as shown in Figure 1, in which it may be convenient to apply or at least to investigate the use of machine learning in the framework of state-to-state formulations. In the present paper, the Kinetic module will be considered while the Transport one will be addressed in a following publication.

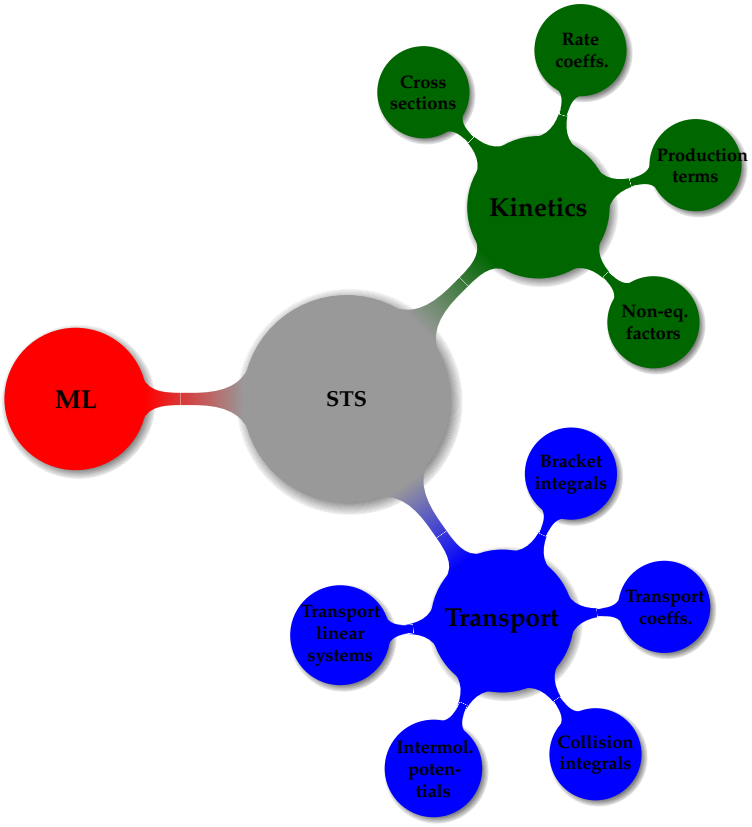


Figure 1. Machine learning for state-to-state: conceptual map.

The organization of the paper is as follows: firstly, the mathematical formulation of the state-to-state formulation is recalled in Section 2. Further details can be found in [23]. Subsequently, in Section 3, several machine learning algorithms are applied to perform regression of the STS relaxation rates. In Section 4, we couple a state-to-state ordinary differential equations (ODE) solver and machine learning in order to speed-up the simulation by relieving the solver from the heavy computation of the kinetics source terms, which are inferred by the machine learning. Section 5 reports the main results from the inference of the set of equations for a one-dimensional inviscid reacting flow behind the plain shock wave in the state-to-state formulation with a deep neural network (DNN). Finally, Section 6 summarizes the conclusions, open issues, and future perspectives.

2. State-to-State Problem Formulation

Consider a chemically reacting gas mixture with translational, rotational, and vibrational degrees of freedom. Under high-temperature conditions, characteristic times of translational and rotational relaxation τ_{tr}, τ_{rot} are much lower compared to those of vibrational relaxation and chemical reactions, τ_{vibr} and τ_{react}; this is confirmed in experiments [24]. This allows one to introduce the following kinetic scaling:

$$\tau_{tr} < \tau_{rot} \ll \tau_{vibr} < \tau_{react} \sim \theta, \tag{1}$$

where θ is the fluid-dynamic time scale. On the basis of Equation (1), a detailed state-to-state model for non-equilibrium gas flows with coupled vibrational relaxation, chemical reactions, and gas dynamics can be developed [23]. Under this condition, translational and rotational degrees of freedom are weakly non-equilibrium and their energy can be expressed in terms of the Maxwell and Boltzmann distributions. At the same time, vibrational relaxation and chemical reactions are strongly non-equilibrium, and therefore, variation of vibrational level populations and chemical composition proceeds at the fluid-dynamic time scale. In this case, conservation equations for mass, momentum, and energy are fully

coupled with master equations for the vibrational level populations and atomic species number densities.

A closed macroscopic flow model can be derived from the Boltzmann equation on the basis of Equation (1) using the Chapman–Enskog method generalized by taking into account different time scales [23]. In the zero-order approximation, we obtain the Euler equations coupled to the master equations for the vibrational state populations including vibrational and chemical production terms; in the first-order, the set of fluid-dynamic equations corresponds to the Navier–Stokes–Fourier equations of viscous reacting flows, and the master equations contain additional state-resolved diffusion fluxes. The general theory is developed in [23].

In the present study, we consider a stationary one-dimensional (1D) flow of a shock heated five-component air mixture including N_2, O_2, NO, N, and O species [25]. The set of equations includes the 1D Euler equations of momentum and total energy

$$\rho v \frac{\partial v}{\partial x} + \frac{\partial p}{\partial x} = 0, \tag{2}$$

$$v \frac{\partial E}{\partial x} + (E + p) \frac{\partial v}{\partial x} = 0, \tag{3}$$

coupled to the master equations for the vibrational level populations n_{ci} of molecules (N_2, O_2, NO)

$$v \frac{\partial n_{ci}}{\partial x} + n_{ci} \frac{\partial v}{\partial x} = R_{ci}^{vibr} + R_{ci}^{react}, \quad i = 0, 1, \ldots l_c, \quad c = 1, 2, \ldots l_m, \tag{4}$$

and equations of chemical kinetics for the number densities n_c of atoms (N, O)

$$v \frac{\partial n_c}{\partial x} + n_c \frac{\partial v}{\partial x} = R_c^{react}, \quad c = 1, 2, \ldots l_a. \tag{5}$$

In Equations (2)–(5), where v is the gas velocity component in the x-direction, ρ and p are density and pressure, l_m, l_a correspond to the numbers of molecular and atomic species in a mixture, and l_c is the number of the last excited vibrational state in the molecular species c. In this study, molecular vibrational energy levels are calculated according to the anharmonic oscillator models. The total number of the excited states is 122 and includes 47 states of N_2, 36 of O_2, and 39 of NO. The total energy per unit volume E includes translational, rotational, vibrational energies, and formation energy [23]. While the translational and rotational energies are calculated on the basis of the local equilibrium Maxwell–Boltzmann distributions and functions of temperature and chemical species number densities, the vibrational energy depends on the non-equilibrium populations of vibrational states. Thus, the calorically-perfect gas model is not applicable in the state-to-state approach.

The vibrational and chemical production rates R_{ci}^{vibr}, R_{ci}^{react}, and R_c^{react} depend on the kinetic scheme and include state-resolved rate coefficients of non-equilibrium vibrational energy transitions and chemical reactions. In the general form, they are given by the expressions [23]

$$R_{ci} = R_{ci}^{vibr} + R_{ci}^{react} = R_{ci}^{VT} + R_{ci}^{VV} + R_{ci}^{2 \rightleftharpoons 2} + R_{ci}^{2 \rightleftharpoons 3}, \tag{6}$$

where VT and VV correspond to the vibration-vibration and vibration-translation energy transitions whereas notations $2 \rightleftharpoons 2$ and $2 \rightleftharpoons 3$ are chosen for the exchange and dissociation-recombination chemical reactions:

$$R_{ci}^{VT} = \sum_M n_M \sum_{i' \neq i} \left(n_{ci'} k_{c,i'i}^M - n_{ci} k_{c,ii'}^M \right), \tag{7}$$

$$R_{ci}^{VV} = \sum_{dki'k'} \left(n_{ci'} n_{dk'} k_{c,i'i}^{d,k'k} - n_{ci} n_{dk} k_{c,ii'}^{d,kk'} \right), \tag{8}$$

$$R_{ci}^{2\rightleftharpoons 2} = \sum_{dc'd'} \sum_{ki'k'} \left(n_{c'i'} n_{d'k'} k_{c'i',ci}^{d'k',dk} - n_{ci} n_{dk} k_{ci,c'i'}^{dk,d'k'} \right), \tag{9}$$

$$R_{ci}^{2\rightleftharpoons 3} = \sum_{M} n_M \left(n_{c'} n_{f'} k_{rec,ci}^{M} - n_{ci} k_{ci,diss}^{M} \right) \tag{10}$$

M stands for the collision partner whose internal state does not vary during the collision; $k_{ii'}^{M}$, $k_{c,ii'}^{d,kk'}$, $k_{ci,c'i'}^{dk,d'k'}$, $k_{ci,diss}^{M}$, $k_{rec,ci}^{M}$ are the state-resolved rate coefficients of vibrational energy transitions, exchange reactions, dissociation, and recombination. The rate coefficients depend on the temperature, chemical species, and vibrational states of all particles involved in the specific reaction; the number of rate coefficients may achieve tens and hundreds of thousands, depending on the mixture composition. Computation and storage of state-specific rate coefficients is one of the most resource consuming parts of state-to-state flow simulations.

For the five-component air mixture considered in the present study, the kinetic scheme includes: VV vibrational energy transitions both within the same chemical species and between molecules of different species; single-quantum VT exchanges; state-specific dissociation reactions of N_2, O_2, NO in collisions with any species; exchange reactions with formation of NO depending on the vibrational states of both reagents and products. For binary mixtures, we keep in the kinetic scheme only VV, VT transitions, and dissociation-recombination reactions. The details of the kinetic scheme can be found in [26].

Thorough analysis and validation of recent models for state-resolved transition and reaction rate coefficients has been carried out in [26]. On the basis of recommendations given in the above paper, we use the following set of rate coefficients in the present study. For binary mixtures, the Schwartz–Slawsky–Herzfeld (SSH) model [27,28] is used for the VV and VT rate coefficients, whereas the state-specific dissociation reactions are described in the frame of the preferential Marrone–Treanor model [29,30] with the parameter $U = D/6k$. For the five-component air mixture, we use the Forced Harmonic Oscillator (FHO) model [31] for the vibrational energy transitions as well as the recent model [32] generalizing the Aliat model for chemical reactions [33] by accounting for the vibrational levels of reaction products; the parameters of the model are derived by fitting the rate coefficients to those obtained in quasi-classical trajectory calculations. Parameters in the Arrhenius law are provided by Park [34]. The rate coefficients of backward reactions are calculated using the detailed balance principle [23]. Note that evaluation of backward reaction rate coefficients is computationally demanding since it requires calculation of vibrational state-specific rotational partition functions.

As shown by the profiling analysis conducted in [26], the computational bottleneck for the system of Equations (2)–(5) is represented by the right-hand side terms. Competing coupled processes are embedded in these terms, which contain time consuming functions (i.e., factorials, exponentials) to be evaluated at each time integration step. In addition, such terms are often stiff and dedicated routines should be preferred.

It is interesting to observe that each of the aforementioned models is characterized by its own computational efficiency, which can significantly affect the overall time-to-simulation. Machine learning methods are agnostic in regards to this aspect, as they provide approximately the same efficiency independently of the model or the processes involved.

3. Regression

Machine learning algorithms are usually grouped into supervised, semi-supervised, and unsupervised [19]. In supervised learning, the algorithm is fed with labeled datasets. Thus, input/output pairs are explicitly provided and regression methods are used to find the best model for the given labeled data via optimization techniques. In unsupervised learning, no labels are provided [35]. The two most popular supervised tasks are certainly classification and regression. The former has discrete output, while the latter is characterized by continuous quantities prediction [36–38].

This section presents the results of the regression of state-to-state relaxation rate terms, Equation (6). Several state-of-the-art machine learning algorithms from the scikit-learn [39] library were compared. Table 2 reports the list of the evaluated algorithms, specifically, Kernel Ridge (KR), Support Vector Machines (SVM), k-Nearest Neighbor (kNN), Gaussian Processes (GP), ensemble methods (Random Forest (RF), Extremely Randomized Trees (ET), Gradient Boosting (GB), Histogram-Based Gradient Boosting (HGB), and Multi-layer Perceptron (MLP). The optimal parameters' combination found by the grid-search algorithm is given in bold font.

Table 2. Hyperparameters settings. The parameters in quotations refer to scikit-learn names. Optimal parameter values with respect to the considered task are indicated in bold font. Reproduced from [40] with permission.

Algorithm	Parameter	Values
KR	kernel	{poly, **rbf**}
	alpha	{1e-3, 1e-2, 1e-1, **1e0**, 1e1, 1e2, 1e3}
	gamma	{1e-3, 1e-2, 1e-1, 1e0, **1e1**, 1e2, 1e3}
SVM	kernel	{poly, **rbf**}
	gamma	{**scale**, auto}
	C	{1e-2, 1e-1, 1e0, 1e1, **1e2**}
	epsilon	{1e-3, **1e-2**, 1e-1, 1e0, 1e1, 1e2, 1e3}
	coef0	{**1e0**, 1e-1, 2e-1}
kNN	algorithm	{**ball_tree**, kd_tree, brute}
	n_neighbors	{1,2,3,4,5,6,7,8,**9**,10}
	leaf_size	{**1**, 10, 20, 30, 100}
	weights	{uniform, **distance**}
	p	{**1**, 2}
GP	n_restarts_optimizer	{(**0**,1,10,100}
	alpha	{1e-3, 1e-2, **1e-1**, 1e0, 1e1, 1e2, 1e3}
	kernel	{**RBF**, ExpSineSquared, RationalQuadratic, Matern}
DT	criterion	{**mse**, friedman_mse, mae}
	splitter	{**best**, random}
	max_features	{**auto**, sqrt, log2}
RF	n_estimators	{10, **100**, 1000}
	min_weight_fraction_leaf	{0.0, **0.1**, 0.2, 0.3, 0.4, 0.5}
	max_features	{sqrt, log2, **auto**}
	criterion	{**mse**, mae}
	min_samples_leaf	{1, 2 ,3, 4, 5, **10**, 100}
	bootstrap	{**True**, False}
	warm_start	{True, **False**}
	max_impurity_decrease	{0.1, **0.2**, 0.3, 0.4, 0.5}
ET	n_estimators	{10, 100, **1000**}
	min_weight_fraction_leaf	{**0.0**, 0.25, 0.5}
	max_depth	{1, 10, 100, **None**}
	max_leaf_nodes	{2, 10, **100**}
	min_samples_split	{2, **10**, 100}
	min_samples_leaf	{1, 10, **100**}

Table 2. *Cont.*

Algorithm	Parameter	Values
GB	n_estimators	{10, **100**, 1000}
	min_weight_fraction_leaf	{**0.0**, 0.1, 0.2, 0.3, 0.4, 0.5}
	max_features	{sqrt, log2, **auto**, None}
	warm_start	{True, **False**}
	max_depth	{1, 10, 100, **None**}
	criterion	{friedman_mse, mse, **mae**}
	min_samples_split	{**2**, 5, 10}
	min_samples_leaf	{1, **10**, 100}
	loss	{**ls**, lad, huber, quantile}
HGB	loss	{**least_squares**, least_absolute_deviation, poisson}
	min_sample_leaf	{1, 5, 10, 15, 20, **25**, 50, 100}
	warm_start	{**True**, False}
MLP	activation	{**tanh**, relu}
	hidden_layer_sizes	{10, 50, 100, **150**, 200}
	solver	{**lbfgs**, adam, sgd}
	leaning_rate	{constant, invscaling, **adaptive**}
	nesterovs_momentum	{**True**, False}
	warm_start	{True, **False**}
	early_stopping	{**True**, False}
	alpha	{0.00001, **0.0001**, 0.001, 0.01, 0.1, 0.0}

It is worth noticing that no deep neural networks have been investigated for simulating the rates in the present Section. In particular, it is interesting to observe that in [41] 15 hidden layers of size 78 have been used to achieve a significant solution speedup for atmospheric chemistry whereas in [42] 5 hidden layers of sizes 512, 256, 128, 64, and 16 have been used for chemistry reduction applied to the DNS of a syngas turbulent oxy-flame, while in [43] 3 hidden layers of sizes 1600, 400, and 400 have been used to solve stiff ordinary differential equation systems related to chemical kinetics. These examples, highlight the potential of deep neural networks architectures for detailed chemistry integration and reduction. Nevertheless, such type of investigation has been deferred to future publication.

Preliminary evaluation of the aforementioned algorithms was carried out in [40]. In the current analysis, previous results have been checked, updated ,and further used to realize a coupling between an ODE solver and machine learning algorithms, as discussed in the next section.

Considering the solution of the state-to-state one-dimensional flow relaxation behind a shock wave, the dataset was generated by an in-house Matlab code. Further details can be found in [26]. The dataset contains the relaxation rate terms as functions of shock distance, number density (molecular and atomic), velocity, and temperature. A traditional data splitting scheme was adopted: 75% of samples were used for training, while 25% were used for testing. The `train_test_split` built-in scikit-learn function was used. Moreover, the `GridSearchCV` algorithm was used to tune the hyperparameters. Table 2 reports the parameter grid explored by each algorithm. The "optimal" parameters were selected by a 10-fold cross-validation performed on training data only.

The algorithms were fed with scaled data. It is generally regarded as good practice to scale the input data, as it improves training and convergence history. Output targets scaling also reduces the range of the output predictions, possibly making the network train faster and model results better. Data leakage was prevented by using the built-in

scikit-learn `fit_transform` function on the train data while only `transforming` the test data. An assessment of data scaling influence on model performance as well as the choice of the optimal preprocessing method was not conducted, as planned for future publication.

Table 3 reports the comparison of the aforementioned machine learning algorithms for the regression of the relaxation rate terms. In order to evaluate the quality of a model's predictions, the following metrics are used: mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and the regression score (coefficient of determination, R^2). Training and prediction times are also shown. No parallel processing is adopted; simulations are run serially, which yields clean baseline estimates. In other terms, `n_jobs=1`.

The problem that we are currently investigating can be recasted as a single-input, single-output regression problem. Each machine learning algorithm estimates a single vibrational level for each relaxation rate term as a function of temperature. Once the "optimal" set of hyperparameters is found, the re-trained network is used to perform a `Multioutput` regression and predict all the relaxation rate terms at once.

Table 3. Comparison of several machine learning algorithms for regression of relaxation rate terms. Reproduced from [40] with permission.

Algorithm	MAE	MSE	RMSE	R^2	$T_{train}[s]$	$T_{predict}[s]$
KR	7.868505e-08	3.800217e-14	1.949414e-07	0.999999	7.612628	0.075077
SVM	1.236652e-02	2.109761e-04	1.452501e-02	0.999786	5.317098	0.008577
kNN	8.655485e-04	2.659352e-06	1.630752e-03	0.999997	0.002296	0.004962
GP	7.235743e-07	2.436803e-12	1.561026e-06	0.999994	118.3911	0.098444
DT	2.417524e-03	1.623255e-05	4.028964e-03	0.999983	0.003520	0.000317
RF	1.140677e-03	5.016757e-06	2.239812e-03	0.999992	4.362630	0.038143
ET	1.595557e-03	6.005923e-06	2.450698e-03	0.999993	2.279543	0.202767
GB	2.300499e-03	1.478234e-05	3.844782e-03	0.999985	4.823793	0.006213
HGB	6.098571e-03	1.395461e-04	1.181296e-02	0.999859	14.385128	0.042188
MLP	6.023895e-03	7.539429e-05	8.682989e-03	0.999943	11.322764	0.009778

As shown in Table 3, the R^2 score does not discriminate very much, assuming similar values among models. Conversely, the error metrics are more reliable discriminators. Kernel Ridge reports the lowest error levels while the Support Vector Machine reports the highest. Comparable error levels are reported by the remaining algorithms; however, there are noticeable differences in the prediction time. Decision Tree is found to be the fastest algorithm. Relatively small prediction time is attained by k-Nearest Neighbor, which occurs considerably faster than Kernel Ridge, providing at the same time very low error levels (better accuracy is achieved only by Kernel Ridge and Gradient Boosting).

Figure 2 shows the parity plot of the predictions of the Decision Tree algorithm. A similar level of closeness between predictions and actual ground truth values, lying close to the diagonal and evenly distributed above and below the diagonal, have been achieved with all other algorithms, but was omitted for brevity.

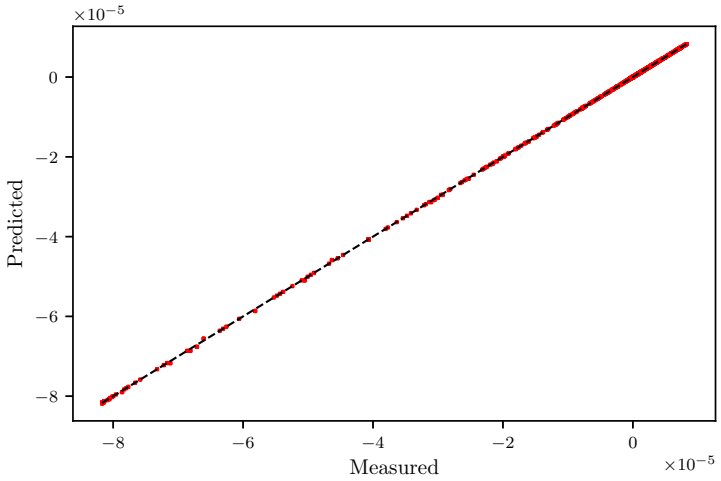
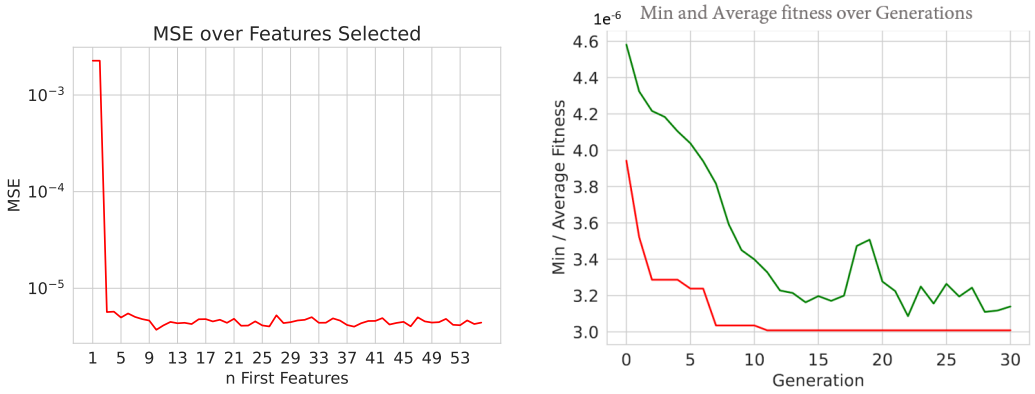


Figure 2. Parity plot of cross-validated predictions against "ground truth" values for the Decision Tree algorithm.

Genetic Algorithms

The family of genetic algorithms deserve a special mention because it can be used for several purposes, among others, to increase the performance by selecting the best subset of features from the input data and by tuning the hyperparameters of the models. An in-depth discussion about genetic algorithms is out of the scope of the paper. The interested reader may refer to [44–48].

In this study, we explore feature selection [49–54]. Advantages of successful feature selection are: possible increasing of accuracy; reducing the training times; simplifying the trained model. Feature selection [55] has been applied to the same dataset described in the previous Section. AdaBoostRegressor was adopted as the estimator. Figure 3a shows the mean squared error as function of the number of features. It can be noted that more than three or four features do not improve the prediction. Figure 3b shows the minimum/average fitness function used for the evolution of the genetic algorithm as function of generations.



(**a**) (**b**)

Figure 3. (**a**) Plot of the mean squared error values for the relaxation source terms regression problem. (**b**) Min/Average fitness function value over generations for the genetic algorithm.

The algorithm was run for 30 generations with a population size of 30. This yields the following results:

```
-- Best Ever Individual =  [1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0,
1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0,
1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1]
-- Best Ever Fitness =  3.0083253440920882e-06
```

which indicates the features selected to provide the best MSE for the test, which is ∼3e-06, slightly better than the previous result. It is interesting that the genetic algorithm does not use any assumptions about the scanned set of features, but it simply searches for the best possible subset of features based on evolutionary selection principles.

Let us compare the performance of grid and genetic search for the best combinations of hyperparameters. The random grid search supported by the `sklearn` (https://github.com/scikit-learn/scikit-learn) library is conducted over all the possible combinations, which can be very time consuming. A genetic algorithm is capable of searching for the best combinations of hyperparameters within the predefined grid; this option is supported by the `sklearn-deap` (https://github.com/rsteca/sklearn-deap) library, see [56] (https://github.com/DEAP/deap). (All accessed on 1 March 2022).

In the following listing, the scores of default, grid, and genetic grid search approaches are compared for the same aforementioned dataset and estimator. Apart from observing the huge increase of the computational time (and a slight improvement in the score) introduced by the grid search with respect to the default values, we notice the time reduction obtained with the genetic search for a very similar score.

```
*** Default Regressor Hyperparameter values:
{learning_rate: 1.0, loss: linear, n_estimators: 50, random_state: 42}
Score with default values = 0.9980733620530691
Time Elapsed = 0.00024271011352539062~s

*** Performing Grid Search...
Best parameters: {learning_rate: 1.0, loss: square, n_estimators: 30}
Best score: 0.9987849200477441
Time Elapsed = 1080.802133321762~s

*** Performing Genetic Grid Search...
--- Evolve in 300 possible combinations ---
gen nevals avg      min      max      std
0   20     0.997418 0.996355 0.998481 0.000614776
1   15     0.997736 0.996508 0.998611 0.000582864
2   11     0.997967 0.997001 0.998614 0.000521943
3   14     0.998238 0.997351 0.998682 0.000403602
4   12     0.998421 0.997624 0.998682 0.00032174
5   9      0.998547 0.997658 0.998682 0.000216111
Best individual is: {n_estimators: 80, learning_rate: 0.215, loss: square}
with fitness: 0.9986822271396701
Time Elapsed = 352.01691937446594 s
```

Moreover, genetic algorithms can be used to directly search the entire parameter space. In particular, differently from the previous result where the genetic algorithm was used to scan a pre-defined hyperparameter space, we can represent each hyperparameter as a variable participating in the search. The results are shown in Figure 4. It is worth observing the opposite concavity of the curves in Figures 3b and 4. This is due to the different definition of the fitness function, which, consequently, in one case is minimized and in the other is maximized. By running a direct genetic search we obtained the following results, which achieved the best score for the considered dataset and estimator.

```
- Best solution is:
params = n_estimators = 25, learning_rate = 0.929, loss = square
Accuracy = 0.99900
```

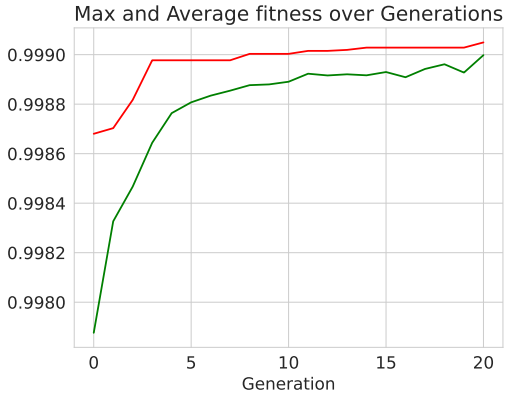


Figure 4. Min/Average fitness function value over generations of the hyperparameters direct genetic search approach.

As a final remark, it is interesting to note that genetic algorithms provide the additional capability of symbolic regression [57–59]. This could be a significant advantage with respect to other techniques, especially when considering coupling of machine learning and ODE/CFD solvers, since it allows the immediate implementation of an algebraic expression. Such promising direction will be further investigated in future work.

4. Machine Learning Coupled with ODE Solver

In the previous section, several machine learning methods for the regression of the relaxation terms, Equation (6), defined in the framework of the state-to-state formulation, have been compared. It was found that by an appropriate selection of hyperparameters, for example, through a cross-validation technique, satisfactorily accurate predictions can be achieved in shorter times with respect to traditional methods. A further continuation of the previous task may then consist in exploiting the potential of the machine learning to alleviate the computational cost of kinetic processes.

In the present section, an interface between the best-performing machine learning algorithm and an ODE solver is explored. Specifically, the same code has been implemented in Matlab and Fortran. In [26] a further comparison between the two implementations was provided. The baseline solution and dataset were generated by running the Matlab version, as described in the previous section, until the equilibrium was reached. The best-performing machine learning algorithm, previously trained, tested, and validated was deployed as a `pickle` module to be fed with input data from the solver.

4.1. Matlab-Python Interface

On the application side, direct Python call functionality from Matlab is used. It is possible, in fact, to access Python libraries, functions or classes from Matlab by adding the py prefix to the Python name, which calls the machine learning regressor model, which simply loads scalers, reshapes, and transforms the input variable array and performs the prediction.

A one-dimensional reactive shock flow relaxation in the framework of state-to-state formulation is considered. Further details about this test case and results can be found in [26]. From the computational point of view, the problem reduces to the integration of an ODE system up to the equilibrium state within selected relative and absolute error tolerances, as shown in Appendix A Listing A1. Due to the stiff nature of the test case, `ode15s` is used which, in turn, calls the `rpart` function, which is responsible for the computation of the right-hand side of the system of equations, whose algorithm is reported in Appendix A Algorithm A1. It is worth observing that Algorithm A1 presents the solver code for the binary mixture. Nevertheless, it can be easily modified to consider air mixtures.

The first aspect to consider is where to actually apply the machine learning. To the best of the authors' knowledge, in fact, this aspect has not been fully detailed in the literature.

Considering Algorithm A1, at least four places appear to be possible candidates, and correspondingly the regression of different targets can be performed:

1. Regression of chemical reaction rate coefficients, k_{ci}, (lines 5–8 of Algorithm A1);
2. Regression of chemical reaction relaxation terms, R_{ci}, Equation (6) (lines 9–10 of Algorithm A1, before matrix inversion at line 12);
3. Regression of the right-hand side inside ODE function call, *dy* (after matrix inversion at line 12 of (lines 5–8 of Algorithm A1);
4. Regression of the ODE solver function call output, [X,Y] at line 2 of Listing A1).

Option (1) would certainly be possible due to the simple temperature dependence of the rate coefficients, which would make their regression quite straightforward. Nevertheless, it would provide a minimal speed-up as we should still perform the expensive main loop to compute the relaxation terms, R_{ci}, (lines 9–10 of Algorithm A1 and a non-negligible communication time would be required for the Python function calls within the loop itself. This option, then, was not further investigated but it may be a reasonable choice depending on the problem's features.

Option (4), to learn and predict the output of the ODE solver, would allow us to circumvent the call to the integrator, *tout court*, providing the greatest speed-up with respect to the baseline solution. In this case, the distance from the shock wave (or equivalently, the relaxation time) was employed as input feature while the species number density (molecule vibrational levels), velocity, and temperature were predicted. Figures 5 and 6 report the profile of temperature and number density for the selected vibrational levels obtained with this approach by Matlab and machine learning for binary and air mixture. Satisfactory agreement was obtained in both cases.

Another relevant aspect to note is the computational cost. Table 4 shows the time-to-solution obtained with Matlab and machine learning for binary and air mixture. As expected, there is no appreciable gain in using machine learning for simple binary mixtures for whom traditional methods perform well. Nevertheless, it is also worth mentioning here that for N_2/N, a computationally simple SSH model is used, whereas for air5, a much more expensive FHO model was adopted. Indeed, by considering air5 mixtures with FHO model a remarkable speed-up ($\sim$300$\times$) was observed. Moreover, the computational cost (CPU time) of the machine learning is almost independent of the number of kinetic processes taking place, which means that the more complex the mixtures, the more significant of a gain will be obtainable with machine learning with respect to the Matlab baseline. Furthermore, the CPU time is independent of the timestep and local stiffness of the ODE. The most advantageous characteristic is that we expect only a moderate increase in the storage requirements with rising number of input scalars [60]. These characteristics appear to be quite appealing in the framework of state-to-state approaches, which tend to saturate the computational resources with bottlenecks associated to chemical (and transport) processes.

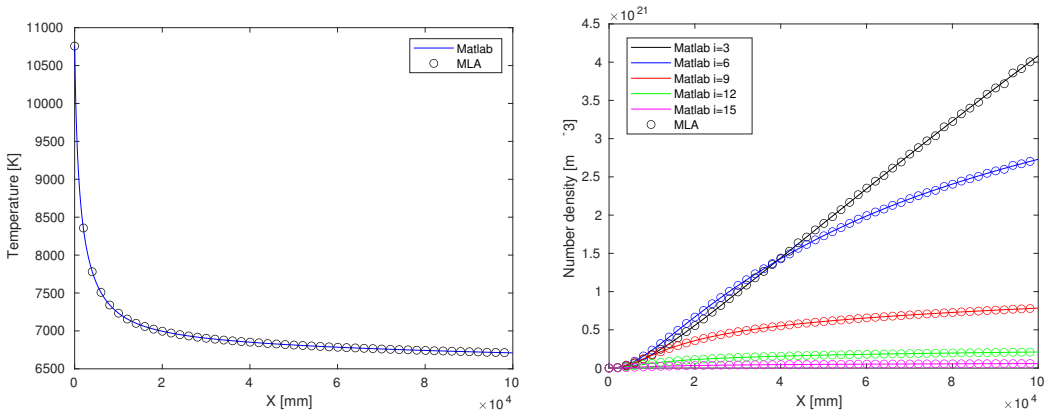


Figure 5. Comparison of Matlab and machine learning solution for the 1D reacting flow behind the shock wave in state-to-state approach for binary N_2/N mixture.

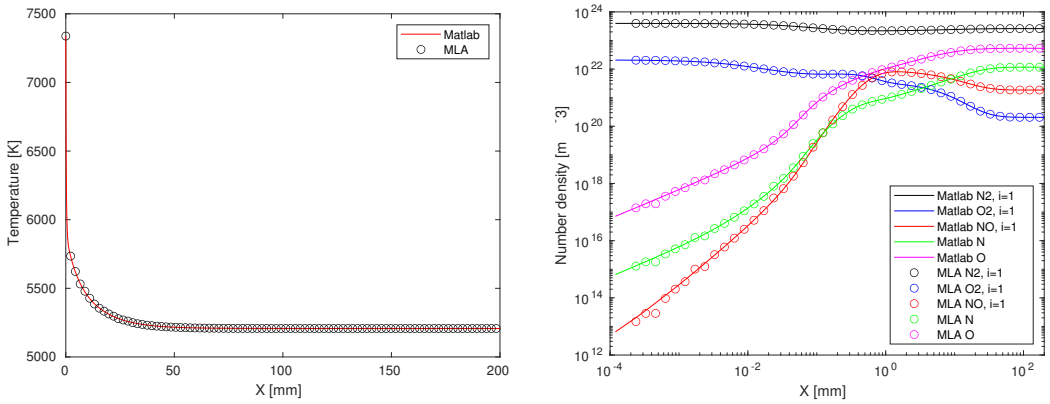


Figure 6. Comparison of Matlab and machine learning solution for the 1D reacting flow behind the shock wave in state-to-state approach for air5 mixture.

Table 4. Comparison of time-to-simulation for Matlab and machine learning solutions for the same number of integration points.

	N_2/N			air5		
	Matlab	**ML**	**FANN**	**Matlab**	**ML**	**FANN**
Time [s]	7.3541	6.8475	0.09	1874.7	6.8974	0.11

Options (2) and (3), to learn the relaxation terms before or after the A matrix inversion, would both permit to avoid the main loop (lines 9–10 of Algorithm A1) but evidently, option (3) will be faster by-passing the A matrix computation and inversion at each step. Nevertheless, when trying to apply this option (2 or 3, equivalently), using the same solver settings as (1) and (4), the results shown in Figure 7 were obtained. The machine learning solution time tends to diverge as soon as the tolerance is decreased. As already observed in [41,61], this behavior is connected to the nature of the solution methods for initial value problems (IVPs). Stiff chemistry solvers fall into this class, in which the usage of machine learning applied to the primary state variables is difficult but still suitable for secondary property prediction. For boundary value problems (BVPs), to which the majority of CFD problems belong to, the field can be more easily predicted by machine learning and corrected to a defined tolerance.

This distinction turned out to be relevant in the effort to couple Matlab ODE solver with machine learning. The problem with using machine learning to predict the integration of the relaxation rate terms is that the accuracy of the predicted values is not sufficient when such values are repeatedly fed into the machine learning model. Even with relative prediction errors reaching as low as 10^{-5}, the solver solution slowly diverges. Consequently, even if machine learning predictions can be very fast, the nature of the IVP does not easily allow for an effective correction.

The complementary nature between machine learning models and differential equations has been recently noted in [62,63] where a possible solution was proposed from the perspective of neural ODE. Moreover, various investigations of stiff systems have been presented in the literature. For example, combustion applications also involve stiff chemistry and deep neural networks have been implemented notably in [42,43]. Such alternatives, nevertheless have been not investigated in the present paper and were deferred to further publication.

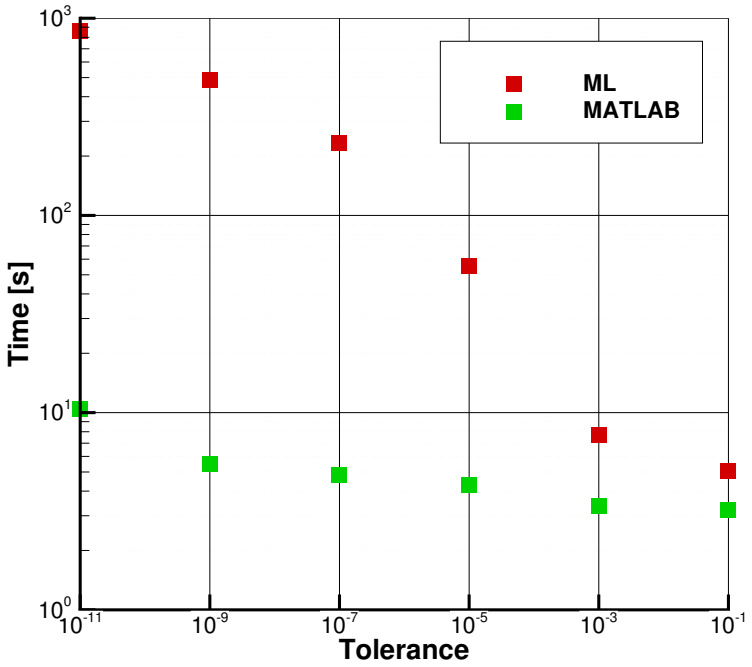


Figure 7. Comparison of Matlab and machine learning time-to-solution for the 1D reacting flow behind the shock wave in state-to-state approach for binary N_2/N mixture. The machine learning call is performed within the ODE system integration before the matrix inversion.

4.2. Fortran-Python Interface

Python has de facto become the lingua-franca of the machine learning community. Numerous packages like Scikit-Learn [39], TensorFlow [64], PyTorch [65], Caffe [66], MXNet [67], and Theano [68], even if written in C/C++/CUDA, always rely on a Python interface. On the other hand, C/C++ and Fortran still represent the mainstream languages for HPC applications. While the realization of a Matlab-Python interface is straightforward, integrating trained machine learning components back into Fortran-based codes is not trivial [69] and is well documented. In this regards, as we plan to undertake such a task, we investigated the existing options.

There are, in fact, several ways to interface machine learning frameworks into a Fortran code:

- Re-coding specific model architectures into Fortran;
- Calling Python from within Fortran (e.g., using wrapper libraries such as Python's C API [70], Cython [71], CFFI (https://cffi.readthedocs.io), SWIG [72] (http://www.swig.org/projects.html), Babel [73], SIP (https://riverbankcomputing.com/software/sip/intro) or Boost.python library (https://wiki.python.org/moin/boost.python);
- Use a pure Fortran NN library, or bridging library (there are several existing solutions, e.g., FANN (https://github.com/libfann), neural-fortran (https://github.com/modern-fortran/neural-fortran), FKB (https://github.com/scientific-computing/FKB), frugally-deep (https://github.com/Dobiasd/frugally-deep), Ro-boDNN [74], TensorflowLite [64] C/C++ API and tiny-dnn (https://github.com/tiny-dnn/tiny-dnn), (all accessed on 1 March 2022);
- Intrinsic Fortran procedures, such as `get_command_argument`, `get_command` to invoke Python scripts and exchange data through I/O files.

The first option would most likely provide efficient solutions and good compatibility with existing Fortran codes. Nevertheless, it would be time-consuming and inflexible as changing the machine learning model architecture means recoding in Fortran. The last option is the easiest and the less efficient and certainly not adequate for HPC frameworks.

Pure Fortran or bridging libraries can provide fast solutions and good compatibility with Fortran codes. At the present moment, the existing solutions only have limited architectures and algorithms available, since they are either Fortran re-implementations of methods or APIs that mirror common machine learning frameworks. Translations from native machine learning model format are often required. The case of bridging libraries in C/C++ would introduce again the necessity of having an additional interface layer.

Using wrappers is by far one of the most frequent solutions [75–78]. In this case, no re-coding is required as the machine learning model remains in Python but various compatibility issues and technicalities may arise depending on the wrapping approach.

Finally, the direct interface to C/C++ machine learning frameworks (i.e., TensorFlow) may be an interesting option. The primary benefit to this approach is flexibility: the implementation can be changed, extended, or optimized without affecting the code integration, the library can be integrated into new codes without requiring a complicated extraction, and the code runs on multiple types of hardware and performs lightweight inference without requiring the full TensorFlow ecosystem.

Considering, for example, the `frugally-deep` package, it first converts DNN models to json files; then it creates C++ header classes, allowing one to load json files as object graphs; the latter can be evaluated on the input data.

After installation, a typical workflow would be as follows:

- Create, train and save deep learning model from Python:
  `model.save("keras_model.h5", include_optimizer=False);`
- Convert the saved model into the required format:
  `python3 convert_model.py keras_model.h5 fdeep_model.json;`
- Load model in C++ using `frugally-deep`:
  `const auto model = fdeep::load_model("fdeep_model.json");`
- Load data from Fortran:
  - pass it to function in C++;
  - make inference;
  - pass inference result back to Fortran.

In order to obtain an estimate of the speed-up attainable by using a direct interface to C/C++ machine learning frameworks, in the present paper, a simple experiment was conducted. Specifically, the C Fast Artificial Neural Network library (FANN) was binded to the Fortran version of the state-to-state 1D Euler shock relaxation solver. With this configuration, the simulation reported in Table 4 for the air mixture was repeated. It was found that without any appreciable difference in the qualitative agreement of the results, the time-to-simulation was about 0.1 s, as reported in Table 4, that is, 70 times faster that the Matlab/Python interface and about four orders of magnitude faster that the original Matlab solution. This kind of speed-up is in agreement with results found in [41].

5. Deep Neural Network for 1D STS Euler Shock Flow Relaxation

In this section, a deep neural network (DNN) is used to infer the 1D Euler system of the equation's solution for high-speed non-equilibrium reacting flows according to a state-to-state description [23]. We consider the relaxation of a flow across a normal shock wave for a binary N_2/N mixture and we shall infer the number density of pseudo-species (47 vibrational levels of N_2) and atomic N, n_{ci}, density ρ, velocity v, pressure p, specific internal energy E as well as relaxation source terms R_{ci} for all the considered processes by using the DNN and adopting the distance from the shock front x as input feature descriptor. The dataset is the same as the one described in Section 3. Hence, the output vector is made up of 100 variables:

$$\mathbf{y} = [n_{ci}, \rho, v, p, E, R_{ci}] \tag{11}$$

The dataset was divided by using the built-in `train_test_split` scikit-learn function, where 75% of samples were used for the training phase, while 25% were used for testing and successively normalized with `MinMaxScaler`. For this task, we used a multi-layer

perceptron (MLP) architecture, a type of ANN that is well suited for nonlinear regression problems. We train the neural network by minimizing average mean-squared-error (MSE) using stochastic gradient descent with two optimizers, an external one by Scipy (L-BFGS-B), a quasi-Newton, full-batch gradient-based optimization algorithm, and an internal one by TensorFlow (Adam). Such strategy was found to be beneficial for convergence. The learning rate was kept constant as equal to the default value. We use a limited number of experiments to select the network architecture, batch size, and epochs based on performance of the evaluation dataset. It found that this task was not particularly sensible to such parameters as soon as a shallow network was not employed.

In early attempts at training the DNN, the network was trained to simultaneously fit all targets. This was the favored approach because (1) the softmax function could be used as the output activation function and (2) previous researchers have succeeded with this approach. However, it was realized that, due to the non-convex and stiff nature of the optimization problem, the learning algorithm would often get trapped in local minima. It was therefore opted to train individual networks for each target output variable independently. This made the DNN both faster to train and much more accurate, as well as easier to modify. This approach is in line with the work of other authors [41,61,79–81].

Figure 8 reports the profiles of number density, relaxation rates for few selected vibrational levels, pressure, and velocity, while Table 5 summarizes the mean relative errors. Values inferred by the DNN are compared with the "ground truth". Satisfactory agreement was obtained for all targets. Nevertheless, several open questions remain, for example, regarding the generalization skills and interpretability of such an approach. We would like to have a robust DNN able to generalize with respect to variations of the full set of initial conditions and able to distinguish the contribution of the different physical processes.

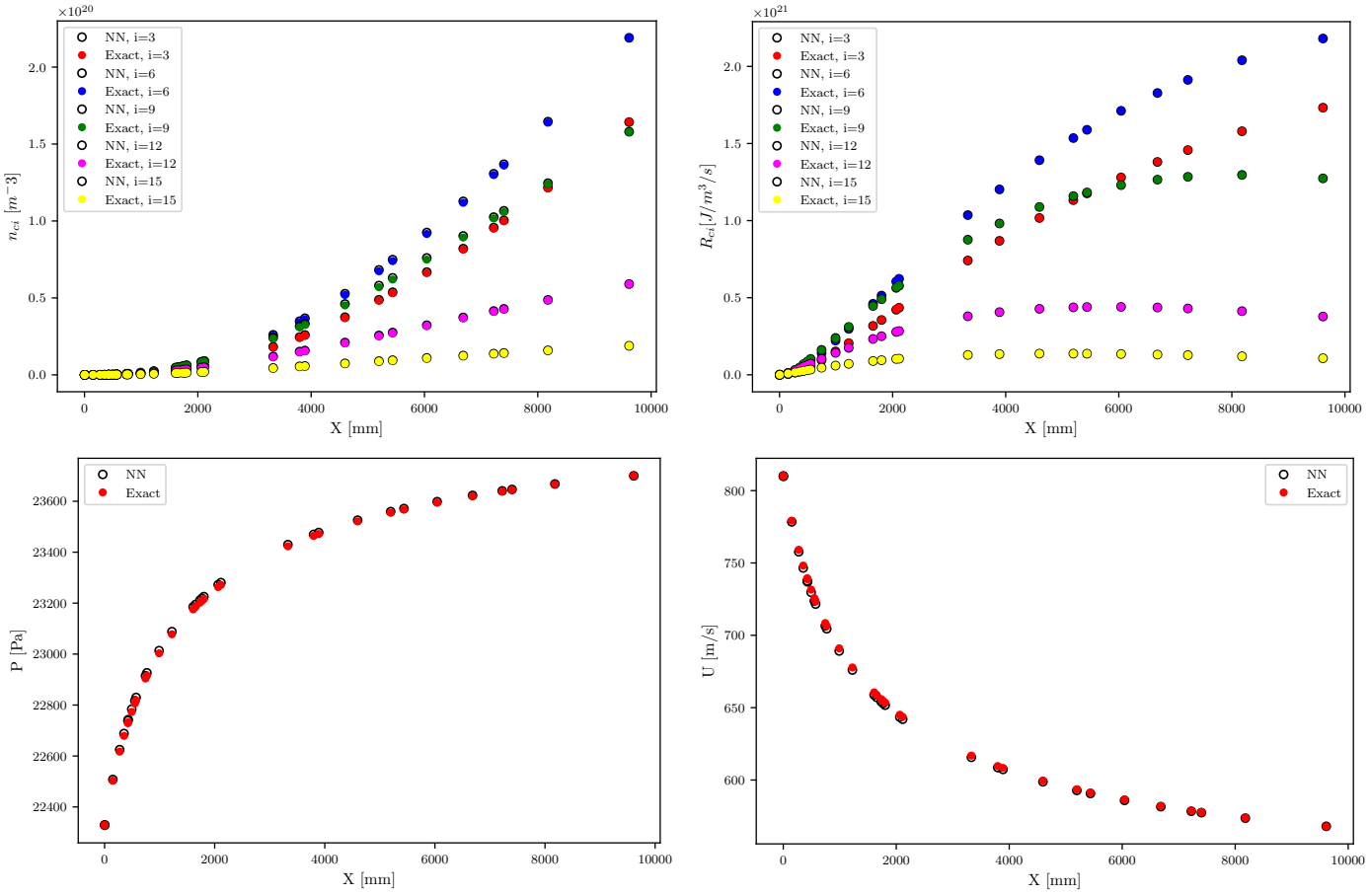


Figure 8. Solution of the state-to-state 1D shock flow relaxation inferred by DNN.

Table 5. Mean relative error associated to inferred variables.

Variable	Mean Relative Error
n_{ci} [m^{-3}]	4.890×10^{-4}
R_{ci} [$J/m^3/s$]	4.039×10^{-4}
ρ [kg/m^3]	3.793×10^{-4}
u [m/s]	3.673×10^{-4}
p [Pa]	1.083×10^{-4}
E [eV]	1.248×10^{-4}

6. Conclusions

In this work, we presented an assessment of a subset of machine learning methods to state-to-state formulations applied to a one-dimensional post-shock flow relaxation.

First, several state-of-the-art machine learning algorithms were compared for the regression of the relaxation rate terms. Decision Tree was found to be the fastest (shortest prediction time) while Kernel Ridge to be the most accurate (smallest errors). Good trade-off solution between prediction time and accuracy was provided by k-Nearest Neighbor. The machine learning-based prediction accuracy could be further improved by training a bigger dataset and refined hyperparameters tuning/optimization.

Secondly, due to the very small prediction time of the best-performing regressor, a coupling between an ODE solver and machine learning was attempted. In this case, the aim was to investigate possible speed-up, obtainable by relieving the solver from the heavy computation of the stiff kinetic terms. Several strategies have been discussed and few issues reported.

The third task consisted in directly inferring the solution of the Euler system of equations for the one-dimensional state-to-state reacting shock flow by exploiting a deep neural network (DNN). In this regards, satisfactory agreement was obtained for all variables of interest. Nevertheless, further research is going on in order to improve generalizability and interpretability.

In future work, we plan to explore the possibility of the regression of state-specific transport coefficients; this will allow to speed-up the computation of the transport module for two-dimensional problems. Moreover, feature extraction and selection techniques are worth being further investigated as another promising way to face the curse of dimensionality imposed by the use of bigger datasets associated to more complex mixtures such as air or CO_2. In this regard, genetic algorithms proved to be a valuable instrument not only for feature selection and hyperparameters tuning but also to perform symbolic regression, whose output can be directly implemented into ODE/CFD solvers.

Author Contributions: Conceptualization, L.C.; methodology, L.C.; software, L.C.; validation, L.C., P.M.; formal analysis, L.C.; investigation, L.C.; resources, L.C.; data curation, L.C.; writing—original draft preparation, L.C.; writing—review and editing, L.C. and E.K.; visualization, L.C.; supervision, E.K.; project administration, E.K.; funding acquisition, E.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Saint Petersburg State University, project ID 84912260.

Acknowledgments: The authors would like to thank Olga Kunova for kindly providing the Matlab code used in Section 4.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Machine Learning Coupled with ODE Solver—Matlab-Python Interface Code

Listing A1. Matlab function call to the stiff ODE solver.

```
options = odeset('RelTol', 1e-12, 'AbsTol', 1e-12);
[X,Y] = ode15s(@rpart, xspan, Y0_bar,options);
```

Algorithm A1 Source term calculation.

function RPART(t, y, dy)
  compute A ▷ A x = B
  compute K_{dr} ▷ diss/rec eq. constant
  compute K_{vt} ▷ vibr/trans. eq. constant
  compute k_d ▷ diss. rates
  compute k_r ▷ rec. rates
  compute k_{vt} ▷ vibr/trans. rates
  compute k_{vv} ▷ vibr/vibr. rates
  for $i \leftarrow 1 : l$ **do** ▷ l, vibrational levels
    $R_D \leftarrow y, k_r, k_d$ ▷ dis./rec. source terms
    $R_{VT} \leftarrow y, k_{vt}$ ▷ vibr./trans. source terms
    $R_{VV} \leftarrow y, k_{vv}$ ▷ vibr./vibr. source terms
  end for
  $B \leftarrow R_D + R_{VT} + R_{VV}$ ▷ full source terms
  $dy \leftarrow A^{-1} \cdot B$
  return dy
end function

References

1. Armenise, I.; Reynier, P.; Kustova, E. Advanced models for vibrational and chemical kinetics applied to Mars entry aerothermodynamics. *J. Thermophys. Heat Transf.* **2016**, *30*, 705–720. [CrossRef]
2. Kunova, O.; Kustova, E.; Mekhonoshina, M.; Nagnibeda, E. Non-equilibrium kinetics, diffusion and heat transfer in shock heated flows of N_2/N and O_2/O mixtures. *Chem. Phys.* **2015**, *463*, 70–81. [CrossRef]
3. Kunova, O.; Kustova, E.; Mekhonoshina, M.; Shoev, G. Numerical simulation of coupled state-to-state kinetics and heat transfer in viscous non-equilibrium flows. In *AIP Conference Proceedings*; AIP Publishing LLC: Melville, NY, USA, 2016; Volume 1786, p. 070012.
4. Kunova, O.; Kosareva, A.; Kustova, E.; Nagnibeda, E. Vibrational relaxation of carbon dioxide in state-to-state and multi-temperature approaches. *Phys. Rev. Fluids* **2020**, *5*, 123401. [CrossRef]
5. Magin, T.E.; Panesi, M.; Bourdon, A.; Jaffe, R.L.; Schwenke, D.W. Coarse-grain model for internal energy excitation and dissociation of molecular nitrogen. *Chem. Phys.* **2012**, *398*, 90–95. [CrossRef]
6. Munafo, A.; Panesi, M.; Magin, T. Boltzmann rovibrational collisional coarse-grained model for internal energy excitation and dissociation in hypersonic flows. *Phys. Rev. E* **2014**, *89*, 023001. [CrossRef]
7. Parsons, N.; Levin, D.A.; van Duin, A.C.; Zhu, T. Modeling of molecular nitrogen collisions and dissociation processes for direct simulation Monte Carlo. *J. Chem. Phys.* **2014**, *141*, 234307. [CrossRef]
8. Torres, E.; Bondar, Y.A.; Magin, T. Uniform rovibrational collisional N2 bin model for DSMC, with application to atmospheric entry flows. In *AIP Conference Proceedings*; AIP Publishing LLC: Melville, NY, USA, 2016; Volume 1786, p. 050010.
9. Berthelot, A.; Bogaerts, A. Modeling of plasma-based CO_2 conversion: Lumping of the vibrational levels. *Plasma Sources Sci. Technol.* **2016**, *25*, 045022. [CrossRef]
10. Sahai, A.; Lopez, B.E.; Johnston, C.O.; Panesi, M. A reduced order maximum entropy model for chemical and thermal non-equilibrium in high temperature CO_2 gas. In Proceedings of the 46th AIAA Thermophysics Conference, Washington, DC, USA, 13–17 June 2016; p. 3695.
11. Diomede, P.; van den Sanden, M.C.; Longo, S. Insight into CO_2 dissociation in plasma from numerical solution of a vibrational diffusion equation. *J. Phys. Chem. C* **2017**, *121*, 19568–19576. [CrossRef]
12. Bonelli, F.; Tuttafesta, M.; Colonna, G.; Cutrone, L.; Pascazio, G. An MPI-CUDA approach for hypersonic flows with detailed state-to-state air kinetics using a GPU cluster. *Comput. Phys. Commun.* **2017**, *219*, 178–195. [CrossRef]
13. Armenise, I.; Capitelli, M.; Garcia, E.; Gorse, C.; Lagana, A.; Longo, S. Deactivation dynamics of vibrationally excited nitrogen molecules by nitrogen atoms. Effects on non-equilibrium vibrational distribution and dissociation rates of nitrogen under electrical discharges. *Chem. Phys. Lett.* **1992**, *200*, 597–604. [CrossRef]

14. Longo, S.; Comunale, G.; Gorse, C.; Capitelli, M. Simplified and complex modeling of self-sustained discharge-pumped, Ne-buffered XeCl laser kinetics. *Plasma Chem. Plasma Process.* **1993**, *13*, 685–700. [CrossRef]
15. Morgan, W.L. The feasibility of using neural networks to obtain cross sections from electron swarm data. *IEEE Trans. Plasma Sci.* **1991**, *19*, 250–255. [CrossRef]
16. Tezcan, S.; Akcayol, M.; Ozerdem, O.C.; Dincer, M. Calculation of Electron Energy Distribution Functions From Electron Swarm Parameters Using Artificial Neural Network in SF6 and Argon. *IEEE Trans. Plasma Sci.* **2010**, *38*, 2332–2339. [CrossRef]
17. Stokes, P.W.; Cocks, D.G.; Brunger, M.J.; White, R.D. Determining cross sections from transport coefficients using deep neural networks. *Plasma Sources Sci. Technol.* **2020**, *29*, 055009. [CrossRef]
18. Schmidt, J.; Marques, M.R.; Botti, S.; Marques, M.A. Recent advances and applications of machine learning in solid-state materials science. *Npj Comput. Mater.* **2019**, *5*, 1–36.
19. Brunton, S.L.; Noack, B.R.; Koumoutsakos, P. Machine learning for fluid mechanics. *Annu. Rev. Fluid Mech.* **2020**, *52*, 477–508. [CrossRef]
20. Bruno, D.; Capitelli, M.; Catalfamo, C.; Celiberto, R.; Colonna, G.; Diomede, P.; Giordano, D.; Gorse, C.; Laricchiuta, A.; Longo, S.; et al. Transport properties of high-temperature Mars-atmosphere components. *ESA Sci. Tech. Rev.* **2008**, *256*. [CrossRef]
21. Brunton, S.L.; Hemati, M.S.; Taira, K. Special issue on machine learning and data-driven methods in fluid dynamics. *Theor. Comput. Fluid Dyn.* **2020**, *34*, 333–337. [CrossRef]
22. Gupta, R.N.; Yos, J.M.; Thompson, R.A.; Lee, K.P. *A Review of Reaction Rates and Thermodynamic and Transport Properties for an 570 11-Species Air Model for Chemical and Thermal Nonequilibrium Calculations to 30000 K*; NASA Technical Report NASA-RP-1232; National Aeronautics and Space Administration, Langley Research Center: Hampton, VA, USA, 1990; Volume 90.
23. Nagnibeda, E.; Kustova, E. *Nonequilibrium Reacting Gas Flows. Kinetic Theory of Transport and Relaxation Processes*; Springer: Berlin/Heidelberg, Germany, 2009.
24. Stupochenko, Y.; Losev, S.; Osipov, A. *Relaxation Processes in Shock Waves*; Springer: Berlin/Heidelberg, Germany, 1967.
25. Kunova, O.; Nagnibeda, E. State-to-state description of reacting air flows behind shock waves. *Chem. Phys.* **2014**, *441*, 66–76. [CrossRef]
26. Campoli, L.; Kunova, O.; Kustova, E.; Melnik, M. Models validation and code profiling in state-to-state simulations of shock heated air flows. *Acta Astronaut.* **2020**, *175*, 493–509. [CrossRef]
27. Schwartz, R.; Slawsky, Z.; Herzfeld, K. Calculation of Vibrational Relaxation Times in Gases. *J. Chem. Phys.* **1952**, *20*, 1591–1599. [CrossRef]
28. Herzfeld, K.; Litovitz, T. *Absorption and Dispersion of Ultrasonic Waves*; Academic Press: Cambridge, MA, USA, 2013; Volume 7.
29. Marrone, P.; Treanor, C. Chemical Relaxation with Preferential Dissociation from Excited Vibrational Levels. *Phys. Fluids* **1963**, *6*, 1215–1221. [CrossRef]
30. Kunova, O.; Kustova, E.; Savelev, A. Generalized Treanor–Marrone model for state-specific dissociation rate coefficients. *Chem. Phys. Lett.* **2016**, *659*, 80–87. [CrossRef]
31. Adamovich, I.; Macheret, S.; Rich, J.; Treanor, C. Vibrational energy transfer rates using a forced harmonic oscillator model. *J. Thermophys. Heat Transfer.* **1998**, *12*, 57–65. [CrossRef]
32. Kustova, E.; Savelev, A.; Kunova, O. Rate coefficients of exchange reactions accounting for vibrational excitation of reagents and products. *AIP Conf. Proc.* **2018**, *1959*, 060010.
33. Aliat, A. State-to-state dissociation-recombination and chemical exchange rate coefficients in excited diatomic gas flows. *Phys. A: Stat. Mech. Its Appl.* **2008**, *387*, 4163–4182. [CrossRef]
34. Park, C. Review of chemical-kinetic problems of future NASA missions. I-Earth entries. *J. Thermophys. Heat Transf.* **1993**, *7*, 385–398. [CrossRef]
35. Brunton, S.L.; Kutz, J.N. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*; Cambridge University Press: Cambridge, UK, 2019.
36. Rasmussen, C.E. *Gaussian Processes in Machine Learning*; Summer School on Machine Learning; Springer: Berlin/Heidelberg, Germany, 2003; pp. 63–71.
37. Stulp, F.; Sigaud, O. Many regression algorithms, one unified model: A review. *Neural Netw.* **2015**, *69*, 60–79. [CrossRef] [PubMed]
38. Kostopoulos, G.; Karlos, S.; Kotsiantis, S.; Ragos, O. Semi-supervised regression: A recent review. *J. Intell. Fuzzy Syst.* **2018**, *35*, 1483–1500. [CrossRef]
39. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
40. Campoli, L. Machine learning methods for state-to-state approach. In *AIP Conference Proceedings*; AIP Publishing LLC: Melville, NY, USA, 2021; Volume 2351, p. 030041.
41. Kelp, M.M.; Tessum, C.W.; Marshall, J.D. Orders-of-magnitude speedup in atmospheric chemistry modeling through neural network-based emulation. *arXiv* **2018**, arXiv:1808.03874.
42. Wan, K.; Barnaud, C.; Vervisch, L.; Domingo, P. Chemistry reduction using machine learning trained from non-premixed micro-mixing modeling: Application to DNS of a syngas turbulent oxy-flame with side-wall effects. *Combust. Flame* **2020**, *220*, 119–129. [CrossRef]
43. Zhang, T.; Zhang, Y.; E, W.; Ju, Y. DLODE: A deep learning-based ODE solver for chemistry kinetics. In Proceedings of the AIAA Scitech 2021 Forum, virtual event, 11–15 & 19–21 January 2021; p. 1139.

44. Holland, J.H. Genetic algorithms. *Sci. Am.* **1992**, *267*, 66–73. [CrossRef]
45. Koza, J.R. Survey of genetic algorithms and genetic programming. In *Wescon Conference Record*; Western Periodicals Company: Racine, WI, USA, 1995; pp. 589–594.
46. Mitchell, M. *An Introduction to Genetic Algorithms*; MIT Press: Cambridge, MA, USA, 1998.
47. Goldberg, D.E. *Genetic Algorithms*; Pearson Education India: London, UK, 2006.
48. Koza, J.R. *Genetic Programming II: Automatic Discovery of Reusable Programs*; MIT Press: Cambridge, MA, USA, 1994.
49. Guyon, I.; Gunn, S.; Nikravesh, M.; Zadeh, L.A. *Feature Extraction: Foundations and Applications*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 207.
50. Chandrashekar, G.; Sahin, F. A survey on feature selection methods. *Comput. Electr. Eng.* **2014**, *40*, 16–28. [CrossRef]
51. Kumar, V.; Minz, S. Feature selection: A literature review. *SmartCR* **2014**, *4*, 211–229. [CrossRef]
52. Li, J.; Cheng, K.; Wang, S.; Morstatter, F.; Trevino, R.P.; Tang, J.; Liu, H. Feature selection: A data perspective. *ACM Comput. Surv. (CSUR)* **2017**, *50*, 1–45. [CrossRef]
53. Cunningham, P.; Kathirgamanathan, B.; Delany, S.J. Feature Selection Tutorial with Python Examples. *arXiv* **2021**, arXiv:2106.06437.
54. Bolón-Canedo, V.; Sánchez-Maroño, N.; Alonso-Betanzos, A. A review of feature selection methods on synthetic data. *Knowl. Inf. Syst.* **2013**, *34*, 483–519. [CrossRef]
55. Vafaie, H.; De Jong, K.A. Genetic Algorithms as a Tool for Feature Selection in Machine Learning. In Proceedings of the International Conference on Tools with Artificial Intelligence—ICTAI, Arlington, VA, USA, 10–13 November 1992; pp. 200–203.
56. Fortin, F.A.; De Rainville, F.M.; Gardner, M.A.G.; Parizeau, M.; Gagné, C. DEAP: Evolutionary algorithms made easy. *J. Mach. Learn. Res.* **2012**, *13*, 2171–2175.
57. Zhong, J.; Feng, L.; Ong, Y.S. Gene expression programming: A survey. *IEEE Comput. Intell. Mag.* **2017**, *12*, 54–72. [CrossRef]
58. Vaddireddy, H.; San, O. Equation discovery using fast function extraction: A deterministic symbolic regression approach. *Fluids* **2019**, *4*, 111. [CrossRef]
59. Vaddireddy, H.; Rasheed, A.; Staples, A.E.; San, O. Feature engineering and symbolic regression methods for detecting hidden physics from sparse sensor observation data. *Phys. Fluids* **2020**, *32*, 015113. [CrossRef]
60. Blasco, J.A.; Fueyo, N.; Larroya, J.; Dopazo, C.; Chen, Y.J. A single-step time-integrator of a methane–air chemical system using artificial neural networks. *Comput. Chem. Eng.* **1999**, *23*, 1127–1133. [CrossRef]
61. Buchheit, K.; Owoyele, O.; Jordan, T.; Van Essendelft, D. The Stabilized Explicit Variable-Load Solver with Machine Learning Acceleration for the Rapid Solution of Stiff Chemical Kinetics. *arXiv* **2019**, arXiv:1905.09395.
62. Chen, R.T.; Rubanova, Y.; Bettencourt, J.; Duvenaud, D.K. Neural ordinary differential equations. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 6571–6583.
63. Rackauckas, C.; Innes, M.; Ma, Y.; Bettencourt, J.; White, L.; Dixit, V. Diffeqflux. jl-A julia library for neural differential equations. *arXiv* **2019**, arXiv:1902.02376.
64. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv* **2016**, arXiv:1603.04467.
65. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 8026–8037.
66. Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; Darrell, T. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM International Conference on Multimedia, Orlando, FL, USA, 3–7 November 2014; pp. 675–678.
67. Chen, T.; Li, M.; Li, Y.; Lin, M.; Wang, N.; Wang, M.; Xiao, T.; Xu, B.; Zhang, C.; Zhang, Z. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv* **2015**, arXiv:1512.01274.
68. Al-Rfou, R.; Alain, G.; Almahairi, A.; Angermueller, C.; Bahdanau, D.; Ballas, N.; Bastien, F.; Bayer, J.; Belikov, A.; Belopolsky, A.; et al. Theano: A Python framework for fast computation of mathematical expressions. *arXiv* **2016**, arXiv:1605.02688.
69. Wang, Y.; Reddy, R.; Gomez, R.; Lim, J.; Sanielevici, S.; Ray, J.; Sutherland, J.; Chen, J. A General Approach to Creating Fortran Interface for C++ Application Libraries. In *Current Trends in High Performance Computing and Its Applications*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 145–154.
70. Van Rossum, G.; Drake, F.L., Jr. *Python/C API Reference Manual*; Python Software Foundation: Wilmington, DE, USA, 2002.
71. Behnel, S.; Bradshaw, R.; Citro, C.; Dalcin, L.; Seljebotn, D.S.; Smith, K. Cython: The best of both worlds. *Comput. Sci. Eng.* **2011**, *13*, 31–39. [CrossRef]
72. Beazley, D.M. SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++. In Proceedings of the Tcl/Tk Workshop, Monterey, CA, USA, 10–13 July 1996; Volume 43, p. 74.
73. Prantl, A.; Epperly, T.; Imam, S.; Sarkar, V. *Interfacing Chapel with Traditional HPC Programming Languages*; Technical Report; Lawrence Livermore National Lab. (LLNL): Livermore, CA, USA, 2011.
74. Szemenyei, M.; Estivill-Castro, V. Real-time scene understanding using deep neural networks for RoboCup SPL. In *Robot World Cup*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 96–108.
75. Johnson, S.R.; Prokopenko, A.; Evans, K.J. Automated Fortran–C++ Bindings for Large-Scale Scientific Applications. *Comput. Sci. Eng.* **2019**, *22*, 84–94. [CrossRef]
76. Prokopenko, A.V.; Johnson, S.R.; Bement, M.T. *Documenting Automated Fortran-C++ Bindings with SWIG*; Technical Report; Oak Ridge National Lab. (ORNL): Oak Ridge, TN, USA, 2019.

77. Evans, K.; Young, M.; Collins, B.; Johnson, S.; Prokopenko, A.; Heroux, M. *Existing Fortran Interfaces to Trilinos in Preparation for Exascale ForTrilinos Development*; Technical Report; Oak Ridge National Lab. (ORNL): Oak Ridge, TN, USA, 2017.
78. Young, M.T.; Johnson, S.R.; Prokopenko, A.V.; Evans, K.J.; Heroux, M.A. *ForTrilinos Design Document*; Technical Report; Oak Ridge National Lab. (ORNL): Oak Ridge, TN, USA, 2017.
79. Mao, Z.; Lu, L.; Marxen, O.; Zaki, T.A.; Karniadakis, G.E. DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators. *arXiv* **2020**, arXiv:2011.03349.
80. Cai, S.; Wang, Z.; Lu, L.; Zaki, T.A.; Karniadakis, G.E. DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *arXiv* **2020**, arXiv:2009.12935.
81. Sharma, A.J.; Johnson, R.F.; Kessler, D.A.; Moses, A. Deep Learning for Scalable Chemical Kinetics. In Proceedings of the AIAA Scitech 2020 Forum, Orlando, FL, USA, 6–10 January 2020; p. 0181. [CrossRef]