The Astrophysical Journal, 940:134 (15pp), 2022 December 1 https://doi.org/10.3847/1538-4357/ac9a4b

Neural Networks for Nuclear Reactions in MAESTROeX

Duoming Fan[1], Donald E. Willcox[1], Christopher DeGrendele[2], Michael Zingale[3], and Andrew Nonaka[1]
[1] Lawrence Berkeley National Laboratory, Center for Computational Sciences and Engineering, One Cyclotron Road, MS 50A-3111, Berkeley, CA 94720, USA
AJNonaka@lbl.gov
[2] University of California Santa Cruz, Applied Mathematics Department, 1156 High Street, Santa Cruz, CA 95060, USA
[3] Stony Brook University, Department of Physics and Astronomy, Stony Brook, NY 11794-3800, USA
Received 2022 July 21; revised 2022 October 10; accepted 2022 October 12; published 2022 November 29

Abstract

We demonstrate the use of neural networks to accelerate the reaction steps in the MAESTROeX stellar hydrodynamics code. A traditional MAESTROeX simulation uses a stiff ODE integrator for the reactions; here, we employ a ResNet architecture and describe details relating to the architecture, training, and validation of our networks. Our customized approach includes options for the form of the loss functions, a demonstration that the use of parallel neural networks leads to increased accuracy, and a description of a perturbational approach in the training step that robustifies the model. We test our approach on millimeter-scale flames using a single-step, 3-isotope network describing the first stages of carbon fusion occurring in Type Ia supernovae. We train the neural networks using simulation data from a standard MAESTROeX simulation, and show that the resulting model can be effectively applied to different flame configurations. This work lays the groundwork for more complex networks, and iterative time-integration strategies that can leverage the efficiency of the neural networks.

Unified Astronomy Thesaurus concepts: Neural networks (1933); Reaction models (2231); Nucleosynthesis (1131); Computational methods (1965); Hydrodynamical simulations (767)

1. Introduction

In stellar astrophysics simulations, nuclear reactions are often the most computationally demanding aspect. Even in moderately complex networks, the timescales of the stiffest reactions can be on the order of pico- or even femtoseconds. In explicit reaction integration schemes, this leads to an overly restrictive (compared to advective, acoustic, or diffusive scale) time step, and for implicit schemes, can require hundreds or thousands of evaluations of the rates per time step. Thus, reactions can be orders of magnitude more expensive than advection and/or implicit global solvers such as self-gravity, momentum, or mass diffusion.

One burgeoning approach to computational fluid dynamics (CFD) is the use of machine-learning techniques to replace various computational kernels such as advection (Papapicco et al. 2022), diffusion (Sirignano & Spiliopoulos 2018), and the Poisson equation (Tang et al. 2017) (for self-gravity, electrostatics, or projection-based decomposition techniques for incompressible flow). In particular, the use of deep neural networks (DNNs) for surrogate modeling in the framework of partial differential equations (PDEs) and ordinary differential equations (ODEs) has been especially popular in recent years. However, while the literature on using DNN models in areas of CFD such as turbulence modeling is increasing rapidly (Echekki & Mirgolbabaei 2015; Duraisamy et al. 2019; Grimberg & Farhat 2020; Lye et al. 2020), there is notably less investigations in using them for chemical kinetics. In terrestrial combustion, this approach has been used for moderate-sized systems of ∼10 species and ∼20 reactions. Astrophysical nuclear reaction networks share a lot in common with terrestrial chemical networks, although the astrophysical rates tend to have much stronger temperature dependence, and hence can be stiffer.

In the context of using a DNN to accelerate or replace computationally expensive PDE or ODE solver, the DNN model is trained to approximate the mapping from input states of the differential equation solver to its solution states. The learning problem then aims at tuning the weights of the DNN model to train it to a high degree of accuracy. Examples of DNN for surrogate modeling include Residual Neural Network (ResNet; He et al. 2016), physics-informed neural network (PINN) (Raissi et al. 2019; Karniadakis et al. 2021), and fractional DNN (Antil et al. 2020). Among them, PINN has achieved success in a wide range of applications by encoding physics constraints into the loss functions of the neural network such that the governing equations are satisfied. However, recent investigations by Ji et al. (2021) and Wang et al. (2021) have shown that the performance of using PINN in stiff chemical kinetic problems with governing equations of stiff ODEs is suboptimal and can often fail due to both numerical and physical stiffness. Hence, our approach to constructing a surrogate model for the reaction computational kernel is more similar to that of Brown et al. (2021) where the goal is to *learn from physics/chemistry*. In addition, we still want to include physics-based constraints in the loss function whenever possible, but they will not be expressed in the form of the governing equations and instead be specific to the problem (for example, conservation of mass).

In this paper we use the astrophysical hydrodynamics code, MAESTROeX (Nonaka et al. 2010; Fan et al. 2019a), to perform millimeter-scale nuclear flame simulations using a neural network to accelerate the reaction steps. We demonstrate that we can train neural networks using data from a traditional MAESTROeX simulation that utilizes stiff ODE integration for the reactions, and run new simulations utilizing this neural network at a reduced computational cost. This particular problem is extremely challenging due to the delicate balance

between advection, thermal diffusion, and reactions. We have implemented a training scheme that is highly accurate and sensitive to this balance, and describe our design decisions below.

2. Model and Prior Numerical Approach

MAESTROeX is a finite-volume code that solves the equations of low Mach number reacting flow in astrophysical environments. Since the low Mach number model does not contain acoustic waves, the time step is limited by an advective Courant–Friedrichs–Lewy (CFL) constraint, which is $\mathcal{O}(1/\mathrm{Ma})$ larger than an acoustic CFL constraint in compressible approaches, where Ma is the characteristic Mach number. The code is suitable for both stratified environments (planar regions or full stars) as well as small-scale simulations without stratification (where it reduces to the system described in Bell et al. 2004c). Here, we focus on the latter case (a millimeter-scale flame); thus, we do not account for gravitational stratification and the model is simpler than the full MAESTROeX equation set,

$$\frac{\partial \rho X_k}{\partial t} = -\nabla \cdot (\rho X_k \boldsymbol{U}) + \rho \dot{\omega}_k, \tag{1}$$

$$\frac{\partial \rho h}{\partial t} = -\nabla \cdot (\rho h \boldsymbol{U}) + \nabla \cdot k_{\mathrm{th}} \nabla T + \rho H_{\mathrm{nuc}}, \tag{2}$$

$$\frac{\partial \boldsymbol{U}}{\partial t} = -\boldsymbol{U} \cdot \nabla \boldsymbol{U} - \frac{1}{\rho} \nabla \pi, \tag{3}$$

$$\nabla \cdot \boldsymbol{U} = S. \tag{4}$$

Here, ρ is the fluid density, X_k is the mass fraction of species k with associated production rate $\dot{\omega}_k$, $\boldsymbol{U}$ is the fluid velocity, h is the enthalpy per unit mass, k_{th} is the thermal conductivity, T is the temperature, H_{nuc} is the nuclear energy generation rate per unit mass, and π is the perturbational (or dynamic) pressure. The divergence constraint is derived directly from the equation of state by taking the Lagrangian derivative and substituting in the equations of mass and energy evolution. The constraint represents a linearized approximation of the velocity field required so that the thermodynamic variables evolve such that $p(\rho, h, \boldsymbol{X}) = p_0$, where p_0 is a constant ambient pressure. The expansion term, S, accounts for local compressibility effects resulting from nuclear burning, compositional changes, and thermal conduction; see Fan et al. (2019a). Millimeter-scale flame instabilities with this algorithm have previously been studied in Bell et al. (2004a, 2004b) and Zingale et al. (2005).

The MAESTROeX algorithm utilizes a projection methodology for the velocity integration, and Strang splitting for the thermodynamic variable integration. The projection algorithm involves an explicit hydrodynamic step followed by a global geometric multigrid Poisson solve to correct the solution so that it satisfies the divergence constraint. The Strang splitting algorithm alternates between a reaction half-step, an advection-diffusion full step, and a reaction half-step to achieve second-order accuracy. Advection is treated explicitly with a second-order accurate Godunov approach based on the corner transport upwind scheme of Colella (1990), and thermal diffusion is treated implicitly using a global Helmholtz linear solver using geometric multigrid. Reactions are integrated using the stiff ODE solver VODE (Brown et al. 1989).

We use a publicly available equation of state (Timmes & Swesty 2000), which includes contributions from electrons, ions, and radiation. We select a simple 3-isotope network describing the first stages of carbon fusion occurring in Type Ia supernovae (SNe Ia). In the simmering and deflagration stages of common SNe Ia models, ^{12}C nuclei fuse with other ^{12}C nuclei to form predominantly either ^{4}He + ^{20}Ne or ^{1}H + ^{23}Na. Because both of these sets of reaction products can be thought to proceed from the decay of a short-lived ^{24}Mg nucleus in an excited state, our network describes these reactions as a single $^{12}\mathrm{C} + {}^{12}\mathrm{C} \rightarrow {}^{24}\mathrm{Mg}$ reaction to reduce the number of equations and isotopes we track. We evolve ^{12}C and ^{24}Mg mass fractions using screening as described in Graboske et al. (1973), Weaver et al. (1978), Alastuey & Jancovici (1978), and Itoh et al. (1979). This particular network contains timescales on the order of 10^{-14} s so implicit integration with VODE is required when using large time steps afforded by the MAESTROeX algorithm. Because common SNe Ia models also generally contain about 50% ^{16}O by mass in the progenitor white dwarf, we include this quantity of ^{16}O in our network, comprising the third species in our network. However, we do not evolve ^{16}O in the network as we are only interested in modeling the early stages of ^{12}C fusion in SNe Ia models and have not extended the current study to account for later ^{16}O burning.

3. Deep Neural Network

3.1. Problem Formulation

As stated in Section 2, the MAESTROeX algorithm integrates the reactions using the stiff ODE solver VODE, which evolves the species and enthalpy from $X_k^{\mathrm{in}} \rightarrow X_k^{\mathrm{out}}$ and $(\rho h)^{\mathrm{in}} \rightarrow (\rho h)^{\mathrm{out}}$ by solving the following system of equations over a time interval of Δt,

$$\frac{\partial \rho X_k}{\partial t} = \rho \dot{\omega}_k, \tag{5}$$

$$\frac{\partial (\rho h)}{\partial t} = \rho H_{\mathrm{nuc}}, \tag{6}$$

where in the absence of weak interactions, the nuclear energy generation rate can be expressed in terms of the specific binding energies, q_k, as

$$H_{\mathrm{nuc}} = -\sum_k q_k \dot{\omega}_k. \tag{7}$$

We note that in Fan et al. (2019a) we integrated temperature rather than enthalpy; it was later shown in Zingale et al. (2021) that integrating energy is a more robust approach, which we have subsequently adopted. In the Strang-split time-integration algorithm for MAESTROeX, we evolve only Equations (5) and (6) in our reaction step. Furthermore, we integrate internal energy, e, rather than enthalpy h, since the time derivatives of these quantities from reactions alone are identical for our constant-pressure simulations.

We seek to replace the current reaction solver using DNNs such that the updated mass fractions and enthalpy are determined by the following:

$$\boldsymbol{X}^{\mathrm{out}} = \mathrm{DNN_1}(\boldsymbol{X}^{\mathrm{in}}, \rho^{\mathrm{in}}, T^{\mathrm{in}}, \Delta t), \tag{8}$$

$$(\rho h)^{\mathrm{out}} = (\rho h)^{\mathrm{in}} + \rho^{\mathrm{out}} \mathrm{DNN_2}(\boldsymbol{X}^{\mathrm{in}}, \rho^{\mathrm{in}}, T^{\mathrm{in}}, \Delta t). \tag{9}$$

Note that the inputs to both DNNs are the same, which allows the two models to be combined during the runtime. Also

note that the density remains unchanged during this reaction step, hence the enthalpy update can be expressed simply as

$$h^{\text{out}} = h^{\text{in}} + \text{DNN}_2(\boldsymbol{X}^{\text{in}}, \rho^{\text{in}}, T^{\text{in}}, \Delta t). \tag{10}$$

Combining Equations (8) and (10) into a single neural network gives

$$[\boldsymbol{X}, h]^{\text{out}} = [\boldsymbol{X}, h]^{\text{in}} + \text{DNN}_{\text{full}}(\boldsymbol{X}^{\text{in}}, \rho^{\text{in}}, T^{\text{in}}, \Delta t). \tag{11}$$

3.2. Architecture

To accurately represent the functionality of the ODE solver VODE, the inputs of the DNNs are density, temperature, and mass fractions. Note that in this work, we use a constant time step in each of our simulations, and hence Δt is not used as an input. For more general problems, Δt will also need to be considered. The input dimension is then given by

$$n_0 = \underset{\text{(density)}}{1} + \underset{\text{(temperature)}}{1} + \underset{\text{(\# of species)}}{M} = M + 2, \tag{12}$$

where $M = 2$ for our reaction network as discussed in Section 2. Due to the sensitivity of the nuclear energy generation (expressed in erg per gram), e_{enuc}, to the accuracy of the species mass fractions (refer to the Appendix for a detailed discussion), the output will include both the species mass fractions and nuclear energy generation e_{nuc} for a total output dimension of $M + 1$.

We consider two different types of neural networks for this problem: one single neural network DNN_{full} satisfying Equation (11), and two parallel neural networks DNN_1 and DNN_2 satisfying Equations (8) and (10), respectively, which will be combined during the runtime to approximate the entire solution. It is shown in Section 4 that the single neural network does not seem to perform as well as the parallel network. Additional advantages of training the networks in parallel include decreasing the total training time and avoiding potential memory limitations.

All DNNs are implemented using a ResNet (He et al. 2016) architecture with a differing number of hidden layers and shortcut connections. Although ResNets are most commonly used in classification problems instead of surrogate models, they have been shown to perform well in modeling problems where there are many inputs and few outputs. ResNets differ from fully connected neural networks in that they include additional connections between nonconsecutive hidden layers, usually skipping two or more layers. In very intricate networks, these *shortcut* connections help to alleviate the vanishing gradient problem, which is when the gradients become increasingly small during backpropagation causing suboptimal convergence of the first few layers of the network. After experimenting with varying the number of hidden layers, hidden nodes, and shortcut connections, we chose to use the architectures specified in Figure 1 for the DNNs that we are training and testing. Note that the single and parallel neural networks contain the same number of total hidden nodes.

The activation function used in each hidden layer is the hyperbolic tangent activation function due to its derivatives being continuously differentiable. As a side note, the CELU activation function, which is also continuously differentiable, can be used to obtain faster convergence, but we found that it does not necessarily result in better accuracy in practice. Finally, a ReLU activation function is added to the output layer of DNN_1 to satisfy the physical constraint that mass fractions must be positive values.

3.3. Loss Function and Scaling

Since one of our chosen neural networks forms a parallel ResNet architecture, each parallel DNN would use a separate loss function: one for the mass fractions and one for the nuclear energy generation. In the case of a single DNN, the total loss function would be the sum of the two parallel loss functions. Due to the fact that these neural networks are modeling systems of ODEs describing physical phenomena, there are many opportunities to incorporate physics constraints into the loss functions. In this particular problem, there are two such constraints we consider when constructing the loss function: (1) the sum of the mass fractions must be conserved in DNN_1, and (2) the nuclear energy generation must be of the same sign in DNN_2. Letting $\tilde{X}_k$ and $\tilde{e}_{\text{nuc}}$ be the model-predicted values of the mass fractions and energy generation, and X_k and e_{nuc} the solution states from the ODE solver, the loss functions for the two neural networks are

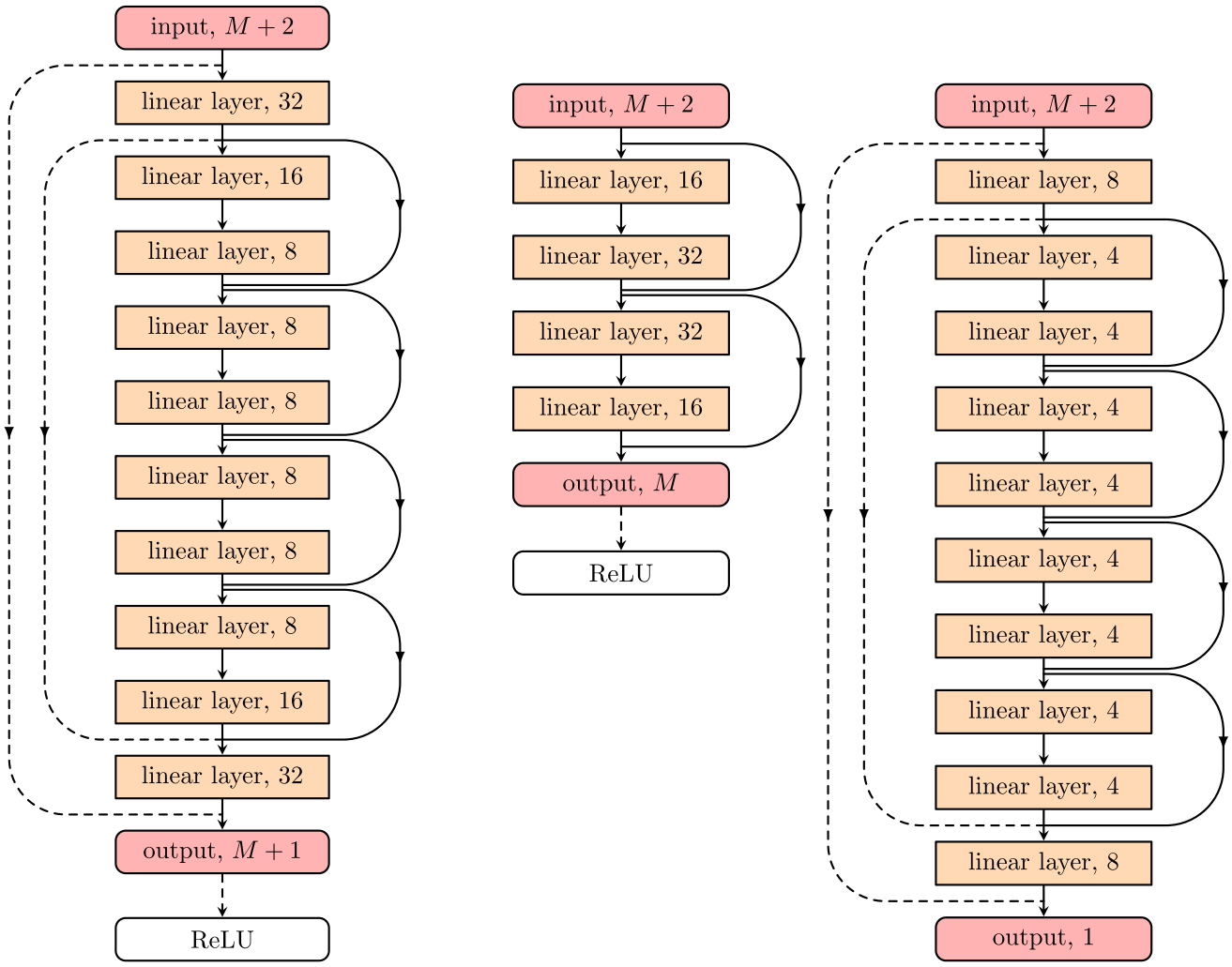
$$\begin{aligned}\text{DNN}_1: \quad \mathcal{L}(\tilde{X}_k) = &\underbrace{\sum_k \frac{1}{N}\sum_{i=1}^{N} \|\tilde{X}_k - X_k\|_2^2}_{\text{MSE}(\tilde{X}_k, X_k)} \\ &+ C_X \underbrace{\frac{1}{N}\sum_{i=1}^{N} \left\| \sum_k \tilde{X}_k - \sum_k X_k \right\|_2^2}_{\text{mass conservation constraint}}\end{aligned} \tag{13}$$

$$\begin{aligned}\text{DNN}_2: \quad \mathcal{L}(\tilde{e}_{\text{nuc}}) = &\underbrace{\frac{1}{N}\sum_{i=1}^{N} \|\tilde{e}_{\text{nuc}} - e_{\text{nuc}}\|_2^2}_{\text{MSE}(\tilde{e}_{\text{nuc}}, e_{\text{nuc}})} \\ &+ C_e \underbrace{\frac{1}{N}\sum_{i=1}^{N} \|\text{sign}(\tilde{e}_{\text{nuc}}) - \text{sign}(e_{\text{nuc}})\|_1}_{\text{physical constraint}}\end{aligned} \tag{14}$$

$$\text{DNN}_{\text{full}}: \quad \mathcal{L}([\tilde{X}_k, \tilde{e}_{\text{nuc}}]) = \mathcal{L}(\tilde{X}_k) + \mathcal{L}(\tilde{e}_{\text{nuc}}) \tag{15}$$

for data points $i = 1,...,N$, where C_X and C_e are cost factors for the physical constraints of mass fractions and energy generation, respectively. Here, MSE refers to the mean square error between the ground truth solutions and the model-predicted values.

In order for neural networks to perform optimally, the inputs and outputs should be scaled such that they are close to order $\mathcal{O}(1)$. This is easily achievable for density, temperature, and nuclear energy generation, all of which are normalized with their respective maximum values. However, the mass fractions vary widely from $\mathcal{O}(10^{-30})$ to $\mathcal{O}(10^{-1})$ (see Figure 5) and hence cannot be normalized in a similar manner. Instead, we employ two potential approaches to alleviate this scaling problem. In the first approach, we convert all mass fractions to their inverse log equivalents, which will significantly reduce the range of scale. In the second approach, we use a loss function with exponentially increasing weights that is symmetric around a mass fraction of 0.25, which is motivated by mass conservation. The results of using these two approaches will be discussed further in Section 4. The first approach to addressing the mass fraction scaling problem is to transform both input and output mass fractions (in DNN_1) to their negative inverse log equivalent, $X_k \rightarrow -1/\log(X_k)$. This reduces the mass fractions to a much more narrow range of



(a) $\mathrm{DNN_{full}}$ (b) $\mathrm{DNN_1}$ (left) and $\mathrm{DNN_2}$ (right)

Figure 1. ResNet architectures for (a) single neural network and (b) parallel neural networks, where $\mathrm{DNN_1}$ has output of mass fractions X_k and $\mathrm{DNN_2}$ nuclear energy generation e_{nuc}. The solid lines represent standard ResNet shortcut connections while the dashed lines are additional connections.

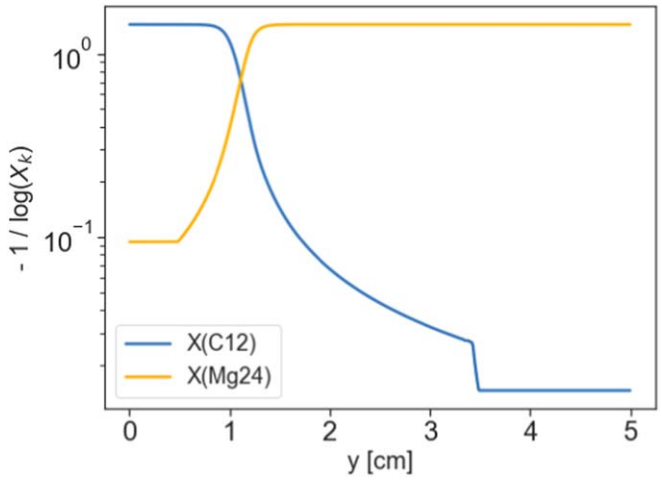


Figure 2. Profile of mass fractions in negative inverse log form at $t = 4.8$ μs.

$\mathcal{O}(10^{-2})$ to $\mathcal{O}(1)$ as shown in Figure 2. This transformation can be applied to the input mass fractions in $\mathrm{DNN_2}$ as well.

The second method is to note that in this particular 3-species network, the sum of X_{C12} and X_{Mg24} is conserved. Therefore, when either mass fraction approaches 0, the other must approach 0.5. In the previous approach where the inverse log form of mass fraction was used, we considered the scaling problem only near 0. However, due to the fact that the sum of the species is conserved, we realized that there is a scaling issue near 0.5 as well. In fact, we need a loss function that penalizes errors near 0 and 0.5 in a symmetric manner as well as takes into account the scaling problem. This can be accomplished by using custom weights $w(X_k)$ in the loss function such that

$$\mathcal{L}_w(\tilde{X}_k) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k} w(X_k)\ ||\tilde{X}_k - X_k||_2^2, \tag{16}$$

where the weight is a function of the mass fraction that is symmetric about $X_k = 0.25$ and exponentially increases toward 0 and 0.5. More specifically, after some initial testing of the accuracy needed for the mass fractions during the real-time simulation, the modified loss function must heavily penalize errors within the mass fraction ranges of (0, 0.1] and [0.4, 0.5). Therefore, the weight function that we selected is similar to a *double* sigmoid function centered at $X_k = 0.05$ and 0.45, and

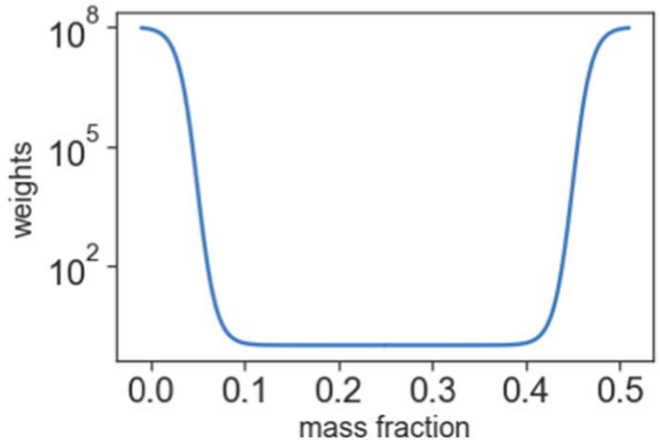


Figure 3. Weights associated with the modified loss function of mass fractions. Note the symmetry around a mass fraction of 0.25.

defined as

$$\log_{10}\{w(X_k)\} = \begin{cases} -\dfrac{p}{\exp(-100X_k + 5) + 1} + p & \text{if } X_k < 0.25 \\ \dfrac{p}{\exp(-100X_k + 5) + 1} & \text{if } X_k \geqslant 0.25 \end{cases}, \quad (17)$$

where p is the order of precision such that 10^p is the maximum value of the weight function. In other words, p dictates the precision where the modified loss function starts to loosen penalties for any values less than order $\mathcal{O}(10^{-p})$. Figure 3 shows an example of the weight function with $p = 8$. During training, we set $p = 10$, but based on some preliminary testing, any value of $p \geqslant 8$ gives stable results. Using this approach to solve the mass fraction scaling problem, the modified loss function for DNN_1 then becomes

$$\mathcal{L}(\tilde{X}_k) = \mathcal{L}_w(\tilde{X}_k) + C_X \frac{1}{N}\sum_{i=1}^{N} \left\| \sum_k \tilde{X}_k - \sum_k X_k \right\|_2^2. \quad (18)$$

3.4. Training and Validation

We define the initial conditions for our simulation as follows. In each simulation, the computational domain is $L_x = 0.625$ cm $\times$ $L_y = 5$ cm with 128×1024 grid cells so the grid spacing is $\Delta x \approx 49$ μm. To initialize the problem, we define the fuel-state density ($\rho_{\mathrm{fuel}} = 5 \times 10^7$ g cm^{-3}), temperature ($T_{\mathrm{fuel}} = 10^8$ K), and composition ($X_{\mathrm{C12,fuel}} = X_{\mathrm{O16,fuel}} = 0.5$, $X_{\mathrm{Mg24,fuel}} = 0$) and use the equation of state to compute the corresponding ambient pressure used throughout the domain, p_0. We define the ash-state temperature ($T_{\mathrm{ash}} = 3 \times 10^9$ K) and composition ($X_{\mathrm{C12,ash}} = 0$). We define profiles for the temperature and species using a smooth hyperbolic tangent,

$$T = T_{\mathrm{fuel}} + \left(\frac{T_{\mathrm{ash}} - T_{\mathrm{fuel}}}{2}\right) \times [1 + \tanh(\alpha(y - \tilde{y}(x)) - 5)], \quad (19)$$

$$X_{\mathrm{C12}} = X_{\mathrm{C12,fuel}} + \left(\frac{X_{\mathrm{C12,ash}} - X_{\mathrm{C12,fuel}}}{2}\right) \times [1 + \tanh(\alpha(y - \tilde{y}(x)) - 5)], \quad (20)$$

$$X_{\mathrm{Mg24}} = 0.5 - X_{\mathrm{C12}}, \quad X_{\mathrm{O16}} = 0.5, \quad (21)$$

where $\alpha = 8$ cm^{-1} is a parameter that controls the steepness of the transition. Then we use the equation of state to compute ρ and h throughout the domain using the temperature, composition, and ambient pressure. The term $\tilde{y}(x)$ represents a spatial perturbation to the initial flame front. In our first tests, $\tilde{y}(x) = 0$ so the flame front is on one plane (one-dimensional). The x-boundary conditions are periodic, and the y-boundary conditions are inflow and outflow. The inflow boundary conditions use a prescribed velocity of $U = (0,\ 10^5)$ cm s^{-1} with the fuel-state condition for the remaining variables. The initial profiles of the species, temperature, and density for the one-dimensional case are shown in the top two panels in Figure 4.

Using the MAESTROeX algorithm with VODE, we model 12 μs of evolution over 500 time steps with $\Delta t = 24$ ns, which corresponds to an advective CFL condition of $\sim$0.5. The peak velocity in this simulation is very close to the inflow velocity, 10^5 cm s^{-1}, which corresponds to a Mach number of $\mathcal{O}(10^{-4})$. The final configuration of the species is shown in the bottom two panels in Figure 4. Over this time, the burning front speed is much smaller than the inflow velocity, so the front travels $\sim$1 cm to the right, which is roughly 20% of the length of the domain.

The data we use to train and validate the DNN models is based on reaction data from this simulation. The inputs and outputs of the ODE solver VODE are saved to plotfiles that are then used to train the neural networks offline. These plotfiles are generated every 10 time steps starting from the time $t = 4.8$ μs to the final time of $t = 12$ μs. Figure 5 shows the profiles of the relevant variables at $t = 4.8$ μs.

We found in preliminary testing that simulations using neural networks that are trained using data directly from the MAESTROeX simulation tend to perform poorly for inputs that are slightly perturbed from the training data. This indicates that the model is most likely overfitting since it only sees a small set of the possible solution space of the ODE solver. To improve the robustness, stability, and overall performance of the DNNs, we present an approach to diversify the training data by slightly perturbing the fluid state just before using VODE to solve the system. The procedure to perturb the states is as follows.

1. Apply a random perturbation within ± 0.005 to X_{C12}.
2. Subtract the same perturbation from X_{Mg24} to ensure mass conservation. If any of the resulting mass fractions is negative, set the perturbation to zero (i.e., do not perturb this cell).
3. Randomly perturb both density and temperature within $\pm 0.02\%$ and $\pm 0.06\%$, respectively.
4. Repeat steps 1−3 in 75% of the cells in the computational domain.

The overall process to generate the training data is shown by the gray arrows in Figure 6. Note that we still run the actual simulation using unperturbed data; the perturbed data is passed in through separate, additional calls to VODE that do not feed back into the main simulation. Also note that we complete

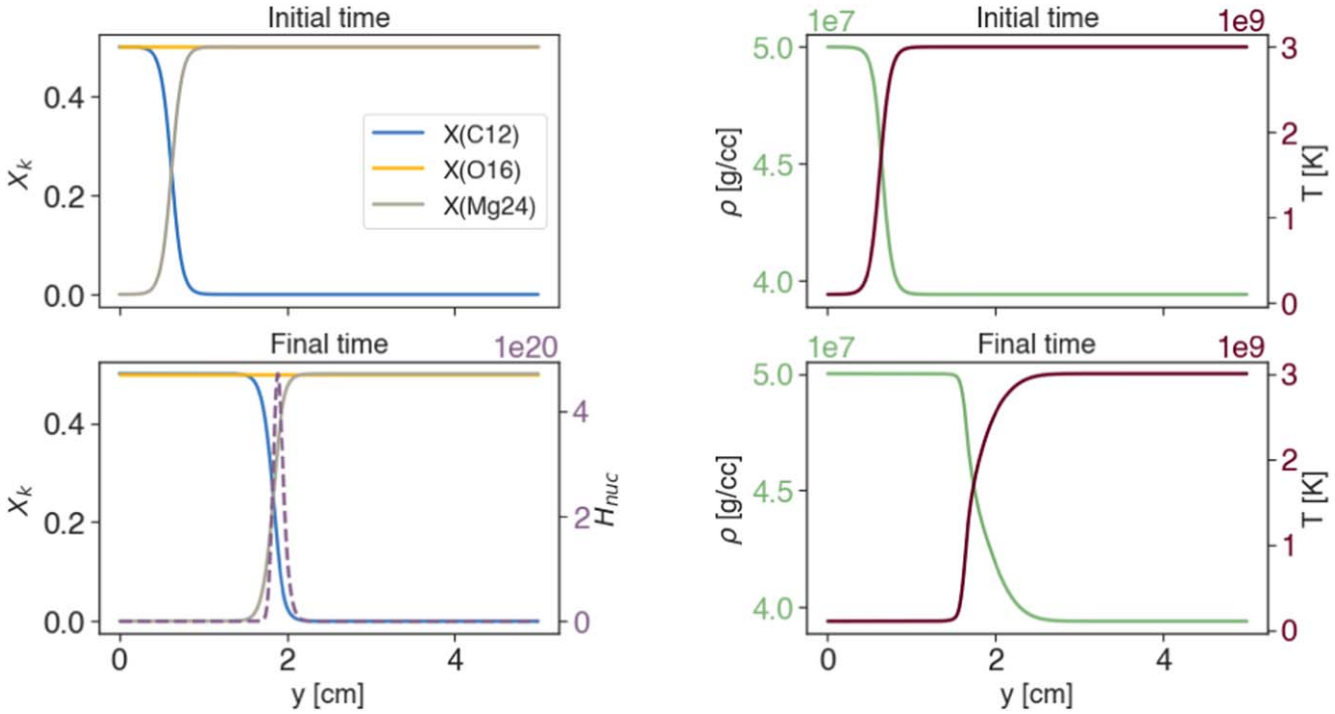


Figure 4. Profile of species mass fractions, density, and temperature at initial time $t = 0$ (top row) and final time $t = 12$ μs (bottom row) for the planar case.

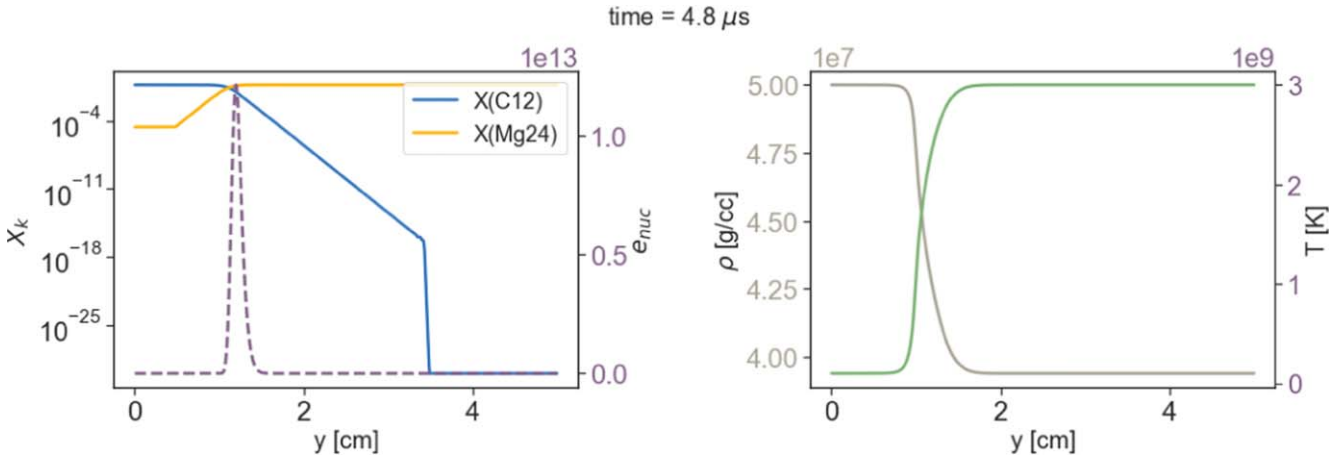


Figure 5. Profiles of species mass fractions, density, and temperature at $t = 4.8$ μs, the time starting from when the training data is generated.

training over a representative set of data over all time steps before using the trained model to start a DNN-based simulation. In future work, we may explore training *on the fly* to further improve the efficiency of the workflow. In the one-dimensional case, the data set we chose is the solution along x-slices located at $x = [0.1, 0.2, 0.3]$ cm, and in the two-dimensional case, the training data is taken from half of the domain ($x \leqslant 0.3125$ cm).

The validation data is a subset of the training data that is not used to update the neural network but is instead used to measure the performance of the neural network during training time. The neural network makes predictions on this data to give an indicator of the error of the predictions to the ground truth, but does not actually backpropagate gradients to optimize for this data. Usually, a different loss function is used for validation. For simplicity, we chose to use the MSE as the validation loss function. We then split the total training data set into 10% for validation and 90% to be used to actually train the DNNs.

In Figure 7 we show the training and validation losses during the first 500 epochs of training the full DNN model using negative inverse log scaling of the inputs. There are two important things to point out here. First, the validation data loss is slightly higher, but follows a downward trend similar to the training data during the training time. This is because the neural network has not been optimized for this validation data, so it will never achieve the same loss as that of the training data; however, this gives us confidence that the model is learning. Second, the validation loss of this neural network has not diverged from the training loss. If the validation loss were to increase while the training loss continues to decrease, this would be evidence of overfitting. On the other hand, if the validation loss was simply not decreasing in the same manner as the training loss (i.e., the validation loss plateaus at a higher loss cost), this would be evidence that our model is not complex enough to capture the intricacies of our data such that it generalizes well. This is a well-known phenomena called the

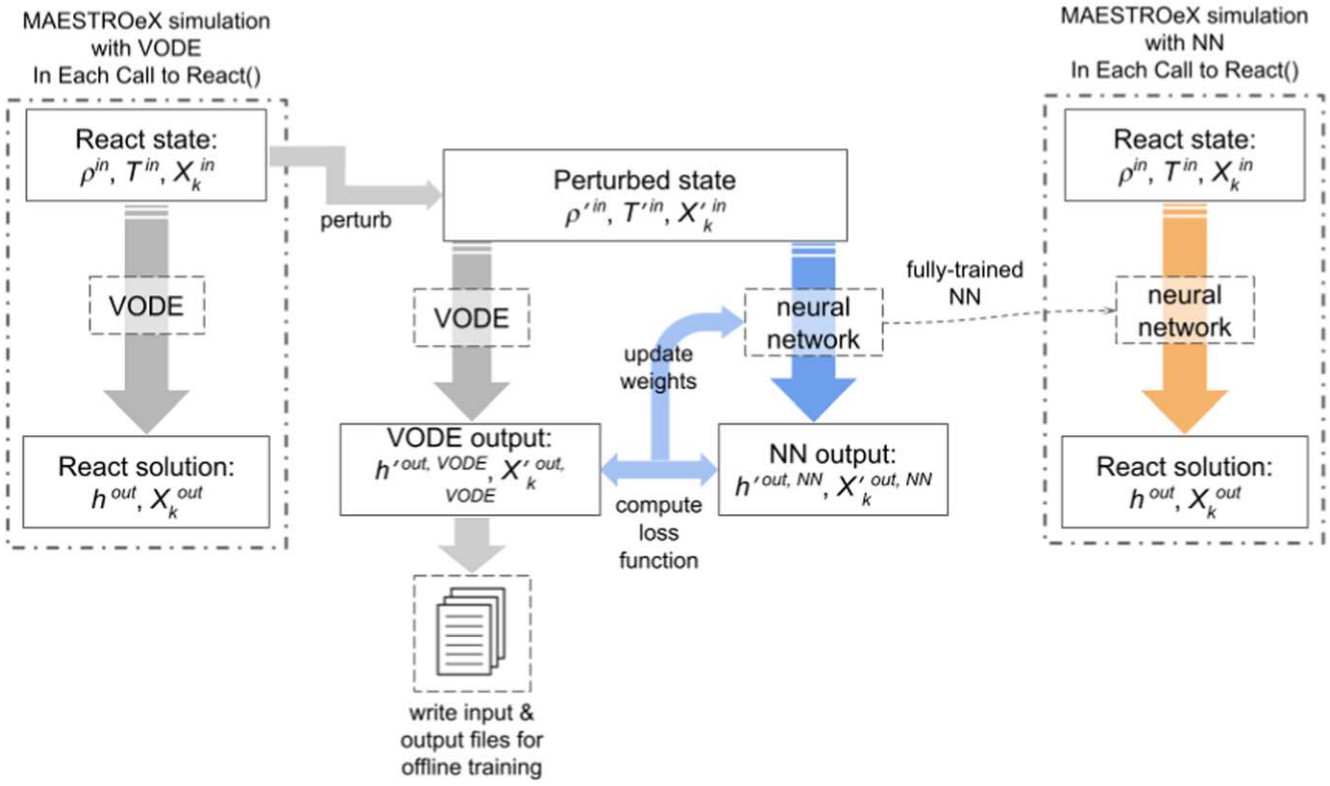


Figure 6. Illustrated overview of our machine-learning approach. The gray arrows indicate the steps involved in generating the training data, the blue arrows the training and validation process, and the orange arrows the testing phase.

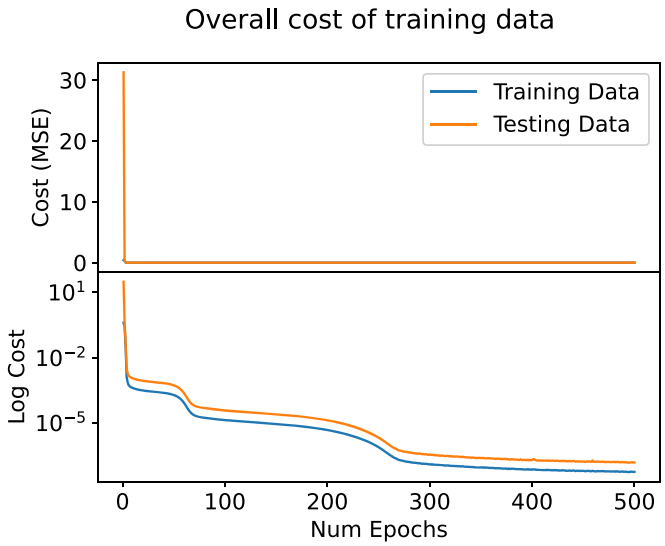


Figure 7. Total loss of training and validation data for 500 epochs during the validation phase.

bias-variance trade-off. Here, we show our model is simple enough to generalize well in addition to being sophisticated enough to accurately capture the dynamics of the problem.

3.5. Testing

The testing phase comes after the training and validation of the neural networks. In order to determine the best scaling approach and loss functions to use, when running with trained DNN in place of VODE, we rerun the same flame diffusion simulation in Section 3.4 starting with the configuration at $t = 4.8\ \mu$s to the final time of $t = 12\ \mu$s. After determining the best training approach, we train a new DNN with a perturbed flame with a V-shaped front, and test this model on three different configurations: the same V-shaped front, a one-dimensional flame, and a sinusoidally varying flame front. The V-shaped front is defined using

$$\tilde{y}(x) = \begin{cases} -x & \text{if } x \leqslant L_x/2 \\ -L_x + x & \text{else} \end{cases}, \tag{22}$$

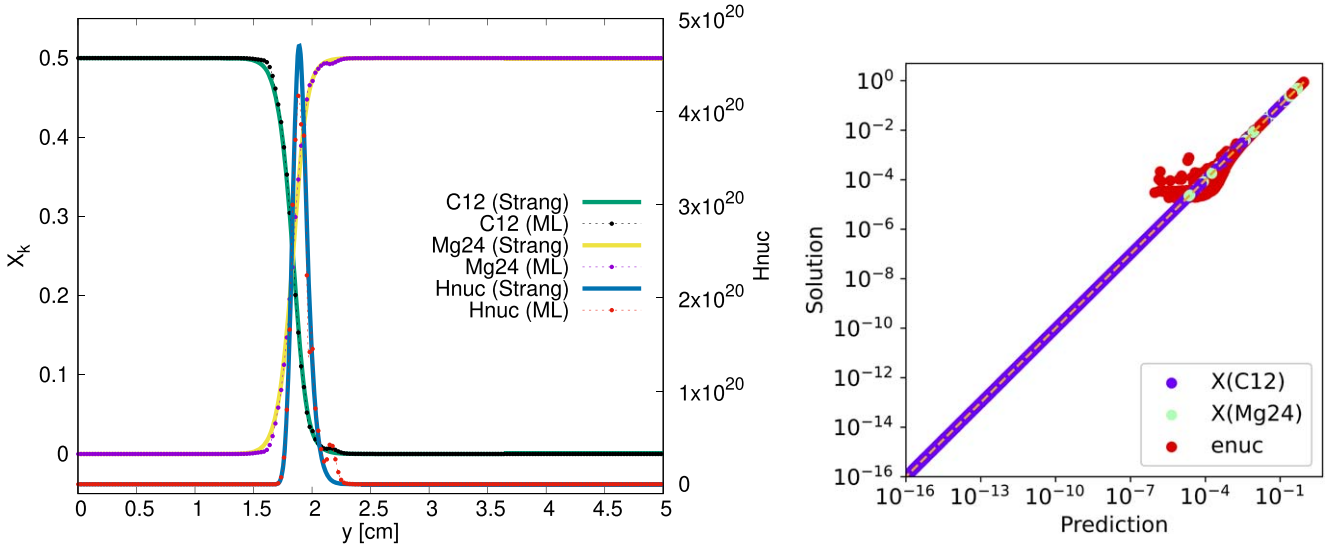
where $\tilde{y}(x)$ is used in Equations (19) and (20) and again, $L_x = 0.625$ cm is the width of the domain. For our simulations involving a sinusoidally varying flame front, we use a domain that is twice as wide (with twice as many grid cells in x so that Δx remains the same) and define the flame front using

$$\tilde{y}(x) = \frac{2L_x}{4} \sin\left(\frac{2\pi}{2L_x} x\right). \tag{23}$$

The results of these simulations are discussed in detail in Section 4.

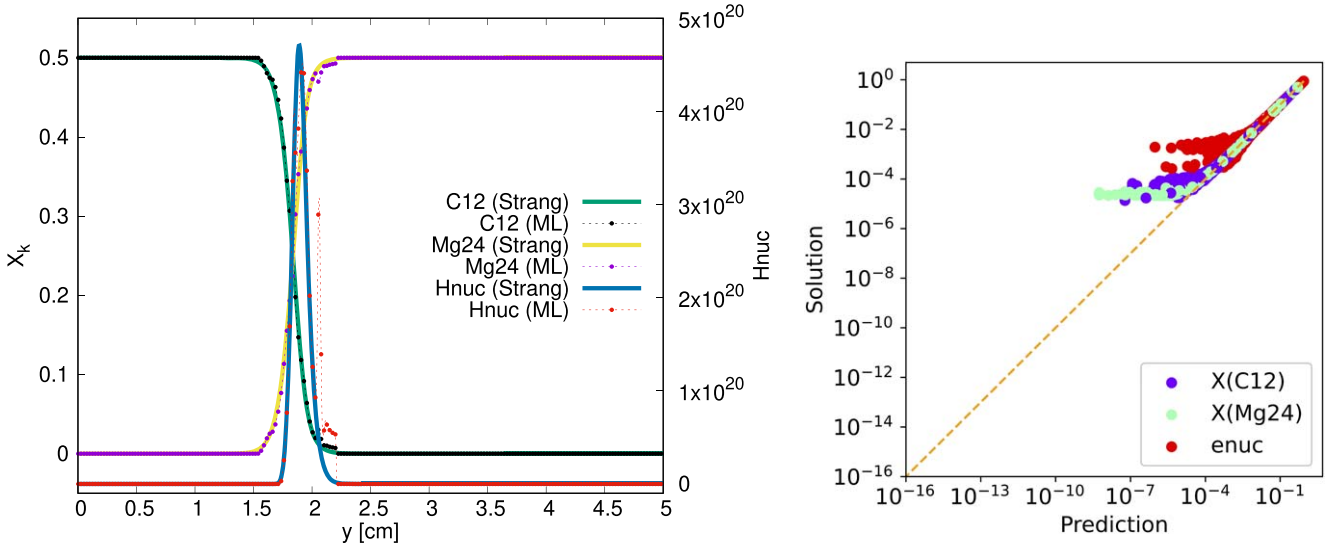
3.6. Software Implementation

The initial simulations used to generate the plotfiles that contain the training data is run using MAESTROeX (Fan et al. 2019a, 2019b, https://github.com/AMReX-Astro/MAESTROeX hash `717e4bc`), which uses Starkiller Microphysics libraries (AMReX-Astro Microphysics Development Team et al. 2022, https://github.com/AMReX-Astro/Microphysics) tag `22.07`), and the AMReX framework (Zhang et al. 2019, 2021; AMReX Development Team et al. 2022, https://github.com/AMReX-Codes/amrex) tag `22.07`).



(a) (b)

Figure 8. (a) Profile of species mass fractions and nuclear energy generation rate at final time $t = 12$ μs, and (b) validation results using the negative inverse log form of X_k as input for full DNN. Note that each point in the profile plot is every five data points on the grid in the y-direction.



(a) (b)

Figure 9. (a) Profile of species mass fractions and nuclear energy generation rate at final time $t = 12$ μs, and (b) validation results using symmetric mass fraction loss function for full DNN.

The training and validation of the neural networks are implemented in Python using the PyTorch library (Paszke et al. 2019) and yt (Turk et al. 2011) is used to extract the training data from the plotfiles. The main reason we chose to use PyTorch over other machine-learning libraries is due to the simplicity of its C++ API LibTorch for integration into our existing C++ codes. By using PyTorch's tracing JIT and LibTorch library, we are able to export the trained neural networks to a model file that can then be imported and used in C++. The integration of LibTorch into the AMReX framework allows the user to take advantage of AMReX's parallelism capability on both host (MPI+OpenMP) and MPI+GPU. We have implemented the use of the LibTorch library in AMReX and, by extension, MAESTROeX in order to test the trained neural networks. The problem setup is found in `/Exec/science/flame_ml/` directory of MAESTROeX.

4. Results

As outlined in Section 3.5, we will first analyze the performance of the full DNN and parallel DNNs by training and testing with the same one-dimensional planar problem. Then after determining the best model to use, we will train a new model using data from a flame with a V-shaped front and test it on three different configurations: the same V-shaped front, a one-dimensional flame, and a sinusoidally varying flame front. For each of the test cases, the final solution at $t = 12$ μs is compared between using VODE and the trained DNN. In addition, plots of the model prediction versus the ground truth solution of the output variables are included to show how well the model performs on the testing data at the end of the training phase. The closer the points are to the $y = x$ line, the closer the prediction is to the ground truth.

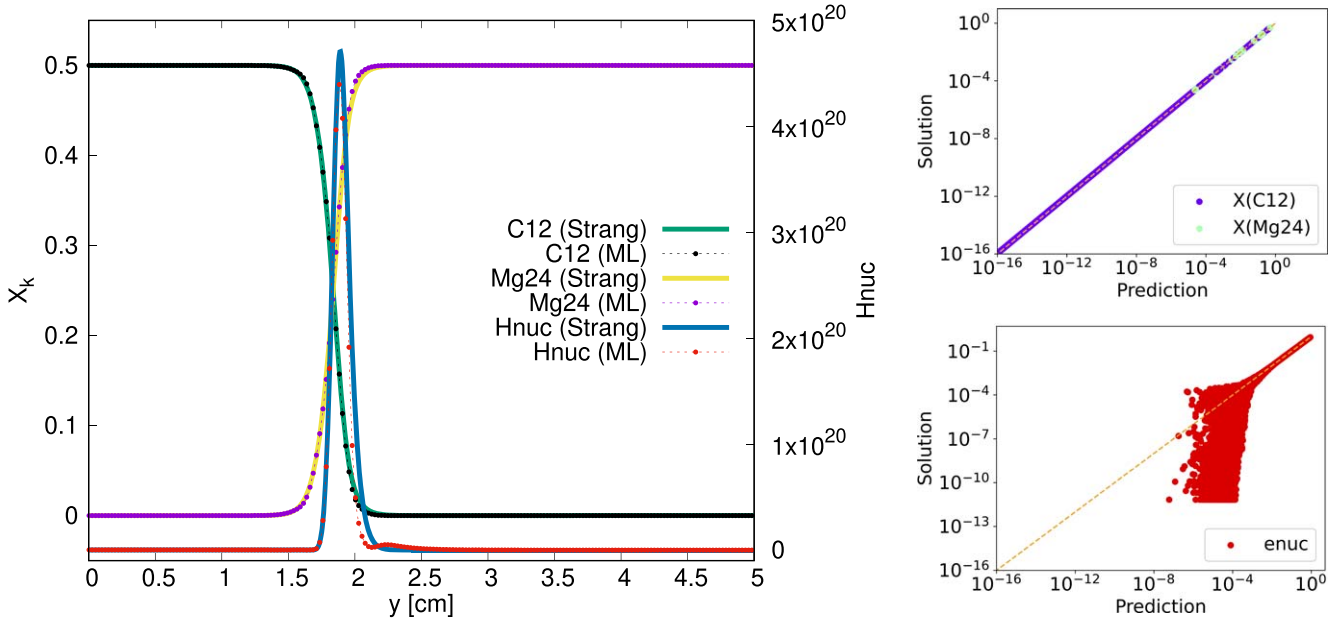


Figure 10. (Left) Final profiles of mass fractions and nuclear energy generation rate at $t = 12$ μs, and (right) parallel validation results using the negative inverse log form of X_k as input for both DNNs. Note that each point in the profile plot is every five data points on the grid in the y-direction.

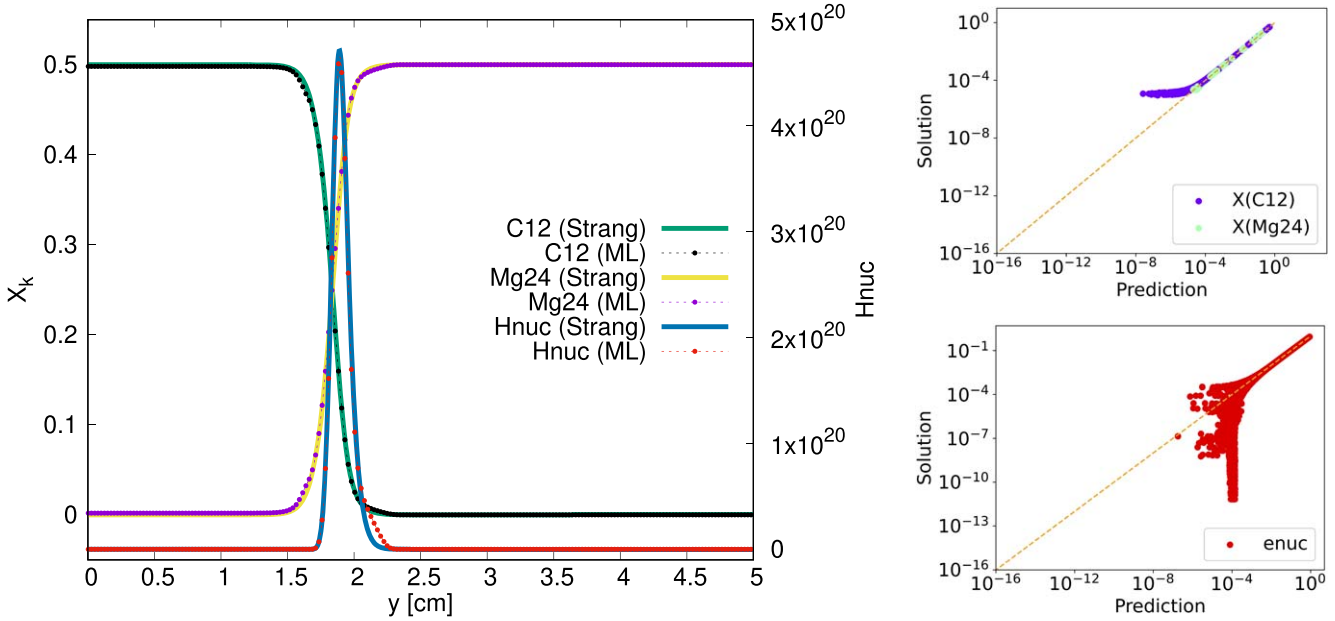


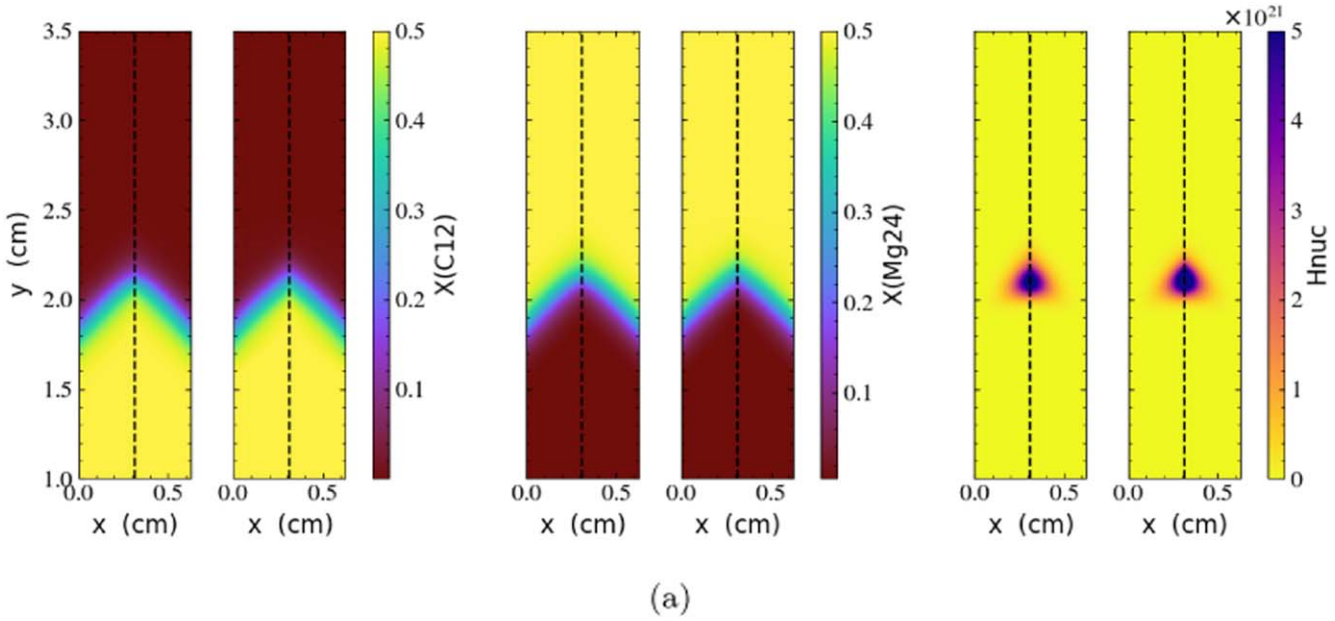
Figure 11. Final profiles of mass fractions and nuclear energy generation rate at $t = 12$ μs (left), and parallel validation results (right) using symmetric mass fraction loss function for DNN_1.

4.1. Full DNN

The full DNN was trained using data from the one-dimensional planar flame problem for 3000 epochs, and the results are shown in Figures 8 and 9. Of the two scaling strategies discussed in Section 3.3, the negative inverse log transformation of X_k inputs seems to be the better choice, although both neural networks show similar shortcomings in performance. The first issue is the emergence of oscillations at the front of the flame, which suggests that the solution has become unstable. The oscillations are clearly more pronounced when using the symmetric mass fraction loss function. The second issue is that both full DNN models have underestimated the peak nuclear energy generation rate by a noticeable margin. In this case, the log transformation of X_k inputs underestimated the peak by $\sim$11% while the symmetric loss function underestimated by $\sim$4%.

4.2. Parallel DNN

Similar to the full DNN, each parallel DNN was trained using data from the one-dimensional planar flame problem for 3000 epochs, and the results are shown in Figures 10 and 11. From these results, we can easily conclude that the parallel DNN performed much better than the full DNN. There is no longer any discernible oscillations at the front of the flame and



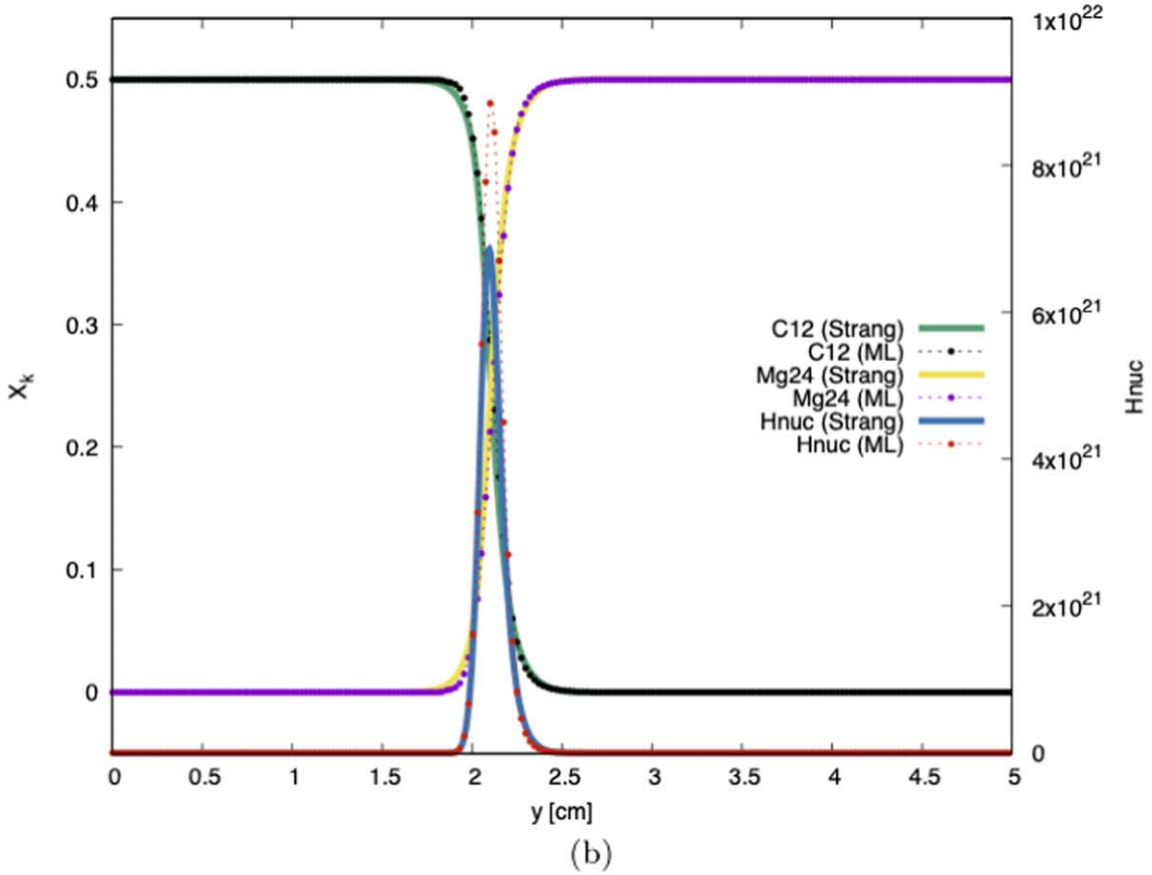


Figure 12. Solutions to the two-dimensional V-shape problem at final time $t = 12$ μs. Note that each point in the profile plot is every five data points on the grid in the y-direction. (a) Contour plots of species mass fractions and nuclear energy generation rate. Each pair of contour plots shows the Strang solution using VODE on the left and the trained DNN model on the right along the vertical dashed line. (b) Profiles along the vertical dashed line.

the profiles of all three neural network outputs (dotted lines) are very similar to the original Strang solution using VODE (solid lines). However, the scaling strategy using the negative inverse log transformation of X_k inputs underestimated the peak nuclear energy generation rate by $\sim$6% while the symmetric mass fraction loss function only underestimated the peak by $<$1%.

The improvement in the performance of the parallel DNNs over the full DNN is likely due to the fact that the loss function of the mass fractions can be derived independently from that of the nuclear energy generation. By defining completely separate neural networks for the mass fractions and $e_{\rm nuc}$, the resulting parallel neural networks are able to model each output state with

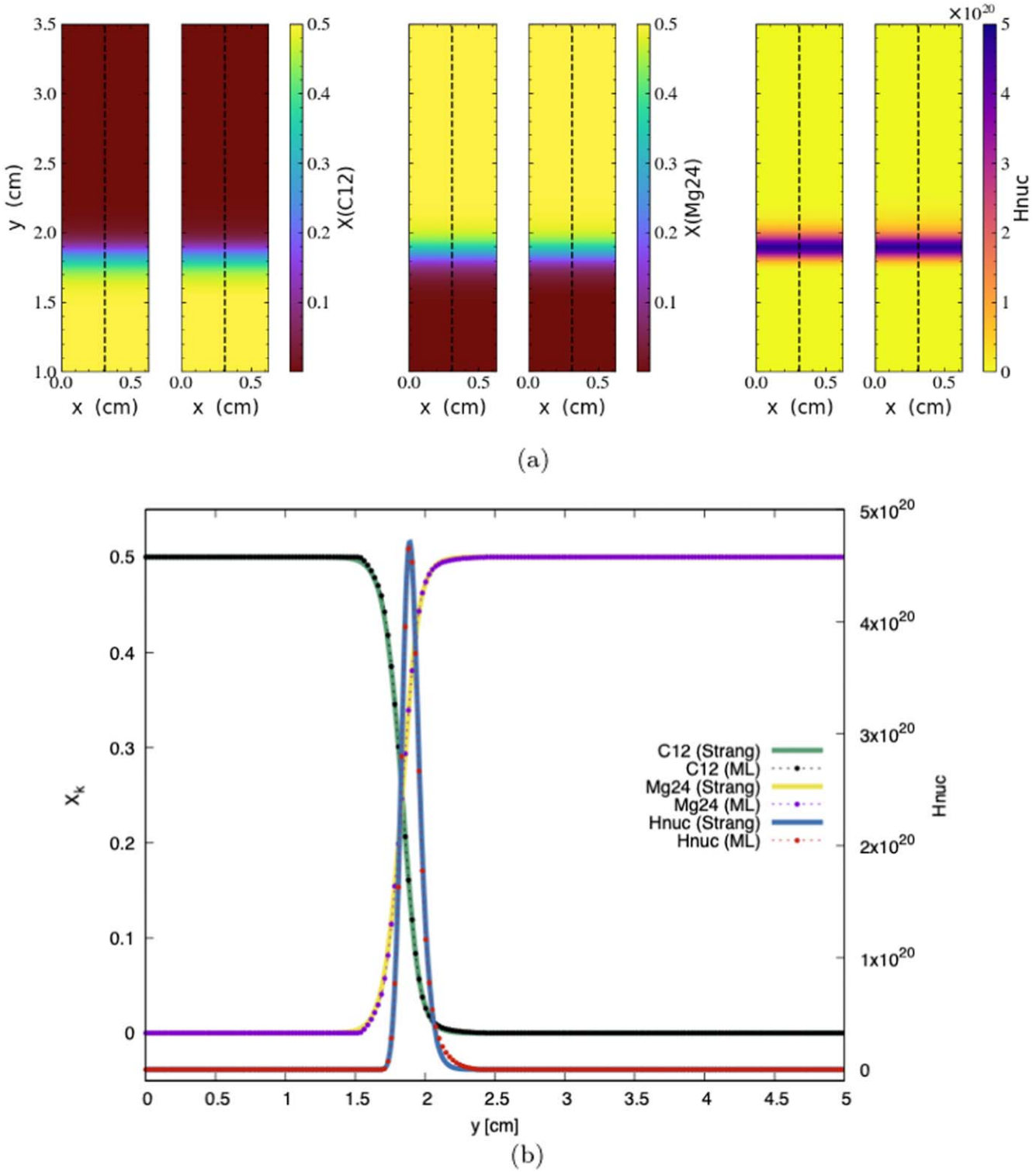


Figure 13. Solutions to the planar problem at final time $t = 12$ μs using an machine-learning model trained using data from the V-shape problem. (a) Contour plots of species mass fractions and nuclear energy generation rate. Each pair of contour plots shows the Strang solution using VODE on the left and the trained DNN model on the right along the vertical dashed line. (b) Profiles along the vertical dashed line.

higher accuracy. Interestingly, the time it takes to train one epoch of data on the parallel neural network is only 4% faster than the full neural network, which is much less of a speedup than expected. This is most likely caused by the uneven distribution of nodes and hidden layers in DNN_1 and DNN_2 with the former containing two-thirds of the total number of hidden nodes and much wider hidden layers than the latter. See the Appendix for a more detailed discussion on the use of the parallel DNN for separating the mass fraction and e_{nuc} neural networks.

4.3. Two-dimensional Flame Problems

Based on the results in Sections 4.1 and 4.2, we chose to train the parallel neural network using the symmetric mass

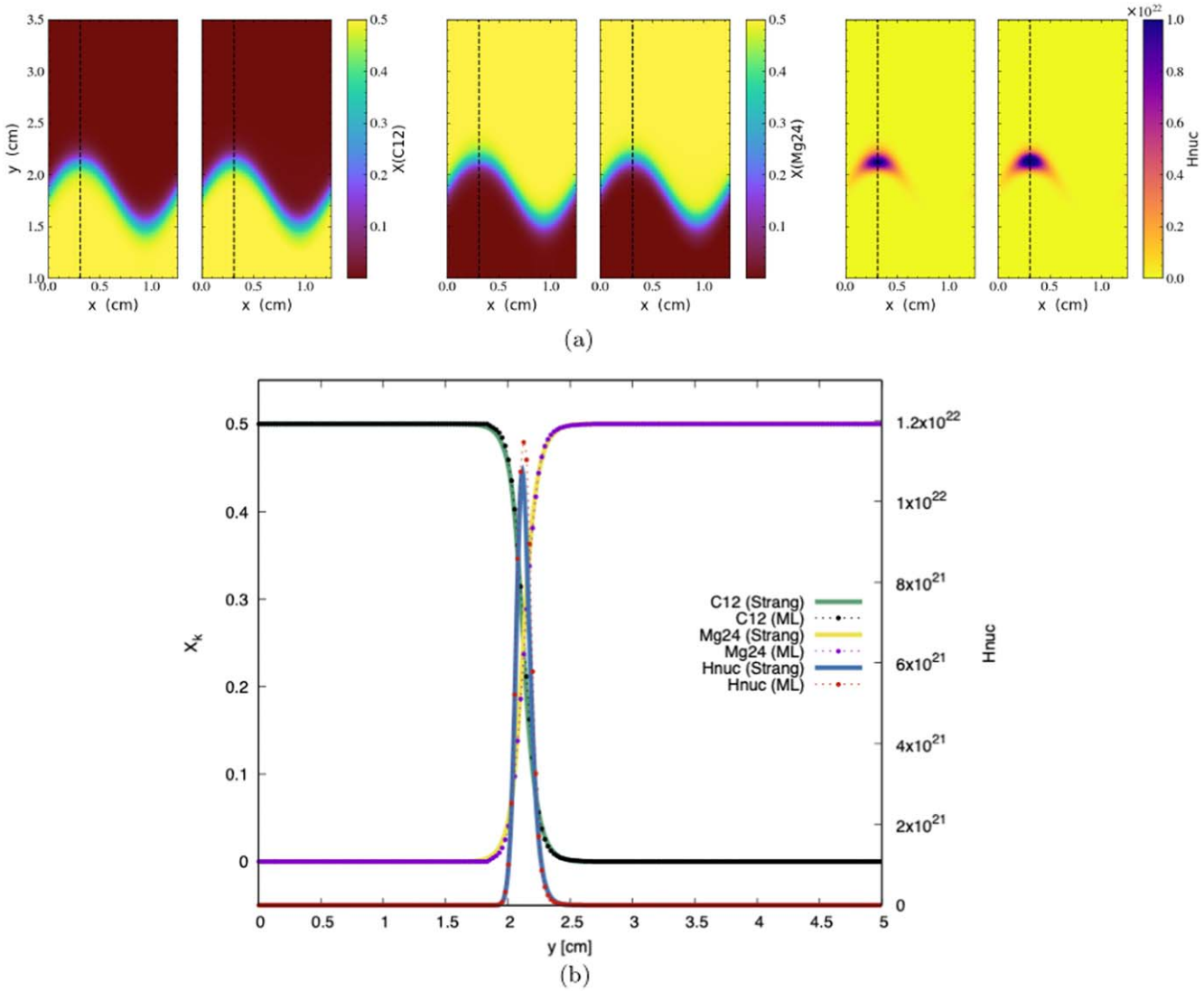


Figure 14. Solutions to the sine wave problem at final time $t = 12$ μs using a machine-learning model trained using data from the V-shape problem. (a) Contour plots of species mass fractions and nuclear energy generation rate. Each pair of contour plots shows the Strang solution using VODE on the left and the trained DNN model on the right along the vertical dashed line. (b) Profiles along the vertical line.

fraction loss function on the two-dimensional flame with a V-shaped front problem. The DNNs were trained using data from half of the domain ($x \leqslant 0.3125$ cm) for 500 epochs, and the results at the final time are shown in Figure 12. The mass fraction solutions and location of the flame front using the parallel DNN are comparable to those using VODE. However, the peak $H_{\rm nuc}$ prediction has an overshoot of approximately 28%. One possible explanation for this subpar performance is that the solution space of the ODE near the peak of the V-shaped flame front makes up only a small portion of the training data set. This is especially true near the end of the simulation as the peak $H_{\rm nuc}$ increases over time and would only exist in the last couple of plotfiles used to generate the training data set. Hence, because the neural network does not see as many instances of the solution near the peak $H_{\rm nuc}$, it is possible that the model does not have enough information to make good predictions there.

By contrast, when we tested this trained DNN on the one-dimensional planar problem, the solution at peak $H_{\rm nuc}$ is almost indistinguishable from the VODE solution as shown in Figure 13. Again, this could be attributed to the fact that there are many more instances of solutions closer to those encountered in the planar problem in the training data set and as a result the neural network is able to produce much more accurate predictions. Given these results, a potential solution to improve the performance of the DNN for the V-shaped flame problem is to include more instances of the flame near the peak $H_{\rm nuc}$ either by expanding the training data set to include data from two-thirds of the domain ($x \leqslant 0.4167$ cm) or to output multiple plotfiles with varying solution perturbations (as described in Section 3.4) near the end of the simulation.

As the final test, we test the trained parallel DNN on another two-dimensional problem that differs from the training set data, which is a flame with a sinusoidally varying flame front. In this problem, the computational domain is 1.25×5 cm with 256×1024 grid cells so that the grid spacing stays the same as in the previous problems. Figure 14 shows that the neural network solutions of both mass fractions and nuclear energy generation rate are close to the VODE solutions, with a slight overshoot in the peak $H_{\rm nuc}$ of approximately 7%.

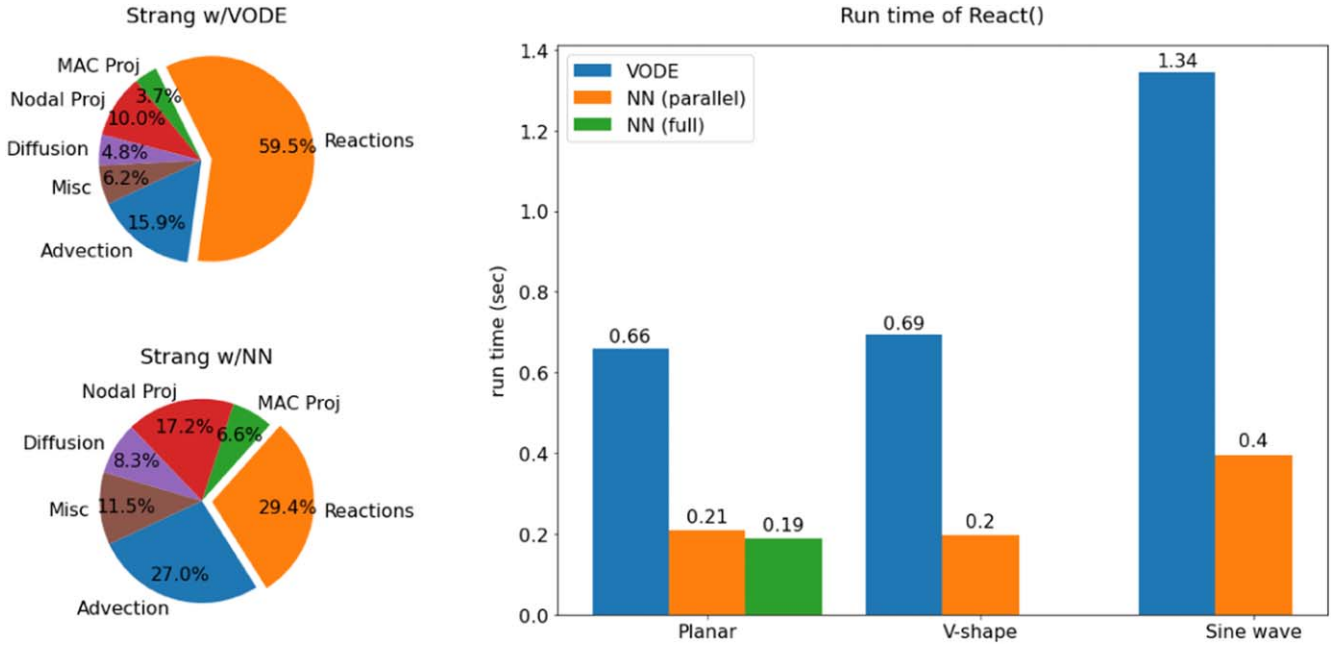


Figure 15. (Left) Relative timing plots of subroutines in MAESTROeX. (right) Comparison of runtimes of the computational reaction kernel using VODE and DNN models.

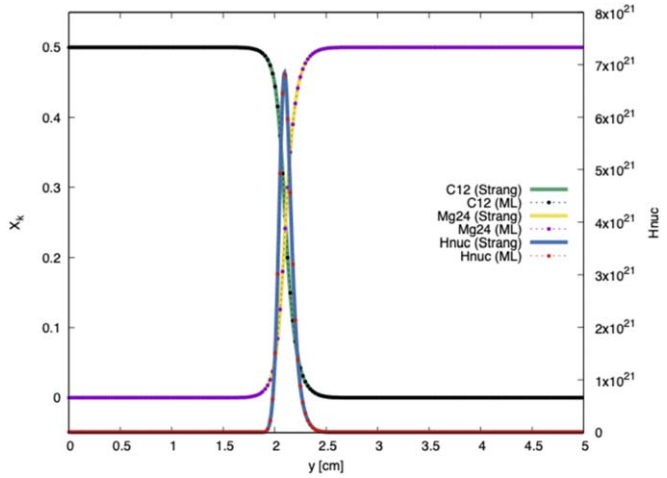


Figure 16. Profiles of mass fractions and nuclear energy generation rate along the x-center line for the V-shaped flame problem at final time $t = 12$ μs using a parallel DNN in the predictor step of second-order Strang splitting algorithm.

4.4. Timing Comparisons

To compare the runtime performance of MAESTROeX when using a neural network instead of the ODE solver VODE for the computational reaction kernel, we run the three previously defined flame problems (planar, V-shaped, and sinusoidally varying flame fronts) on four cores (pure MPI) on an AMD EPYC 7702P processor and record the time it takes to complete the reactions subroutine per time step. The left plots in Figure 15 show that in our simulations using VODE, the runtime is dominated by reactions, using 59.5% of total runtime, whereas using DNN reduced the reactions runtime to 29.4%. In terms of the time it takes to run the reaction kernels per time step, using the neural network results in speedups of 3.14–3.45, depending on the test problem (see the right plot in Figure 15). We note that more complex networks using neural networks should result in larger speedups on both a CPU and GPU.

5. Conclusions and Future Work

We have demonstrated that we can train DNNs with a ResNet architecture using data obtained from the ODE solver VODE given that the data have been perturbed slightly to represent a larger portion of the possible solution space of the solver. In terms of accuracy and stability during simulation using MAESTROeX, the parallel DNN showed much better performance than the full DNN. In addition, the trained neural network can be used successfully in a test case that it has never seen before (i.e., a flame with a sinusoidally varying front) given that the training data was obtained from a similar test (i.e., a flame with a V-shaped front). We also presented two strategies to mitigate the mass fraction scaling problem. Of the two strategies, the one that uses the symmetric mass fraction loss function performed slightly better than transforming the mass fraction inputs to its negative inverse log form.

One area of potential future work that looks promising is using a neural network model to compute the reaction solution in the predictor steps of a high-order numerical algorithm such as the spectral deferred corrections (SDC) method. Even in the second-order Strang splitting algorithm that MAESTROeX currently uses, a slight speedup of 1.32 over all reactions steps can be seen if we replace the reaction ODE solve in the predictor step with our trained DNN. We would expect to see larger speedups in high-order methods that consists of multiple predictor steps. To demonstrate the potential of this idea, when we applied this approach in the two-dimensional flame with a V-shape front problem using the same parallel DNN that we previously trained in Section 4.3, the final solution is exceedingly close (<0.2% error) to the one from the original Strang splitting algorithm (see Figure 16).

Another topic for future work could be to develop and train neural networks with an additional input of the time step, which would lead to the potential implementation of PINNs. However, in the case of stiff ODEs like the ones describing the nuclear reaction networks used in MAESTROeX, we would most likely need to consider a relaxation constraint instead of

directly computing the gradients in the ODE (Ji et al. 2021). Finally, we could look to model more complex nuclear reaction networks with 13 or 19 isotopes in the future.

The work at LBNL was supported by the U.S. Department of Energy's Scientific Discovery Through Advanced Computing (SciDAC) program under contract No. DE-AC02-05CH11231. The work at Stony Brook was supported by DOE/Office of Nuclear Physics grant DE-FG02-87ER40317. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under contract No. DE-AC02-05CH11231. The authors thank A. Harpole for valuable contributions to the Starkiller Microphysics library.

Facility: NERSC.

Software: Refer to Section 3.6.

Appendix
Nuclear Energy Generation Sensitivity Analysis

Theoretically, the enthalpy could be updated using only the mass fractions. This is because after the evolving the mass fractions from $\boldsymbol{X}^{\rm in} \rightarrow \boldsymbol{X}^{\rm out}$, the resulting reaction rates are computed as

$$(\rho\dot{\omega}_k)^{\rm out} = \frac{\rho^{\rm out}(X_k^{\rm out} - X_k^{\rm in})}{\Delta t} \tag{A1}$$

and the nuclear energy generation rate becomes

$$(\rho H_{\rm nuc}) = -\sum_k (\rho\dot{\omega}_k)^{\rm out} q_k. \tag{A2}$$

Substituting Equations (A1) and (A2) into the simplest discrete form of Equation (6) gives

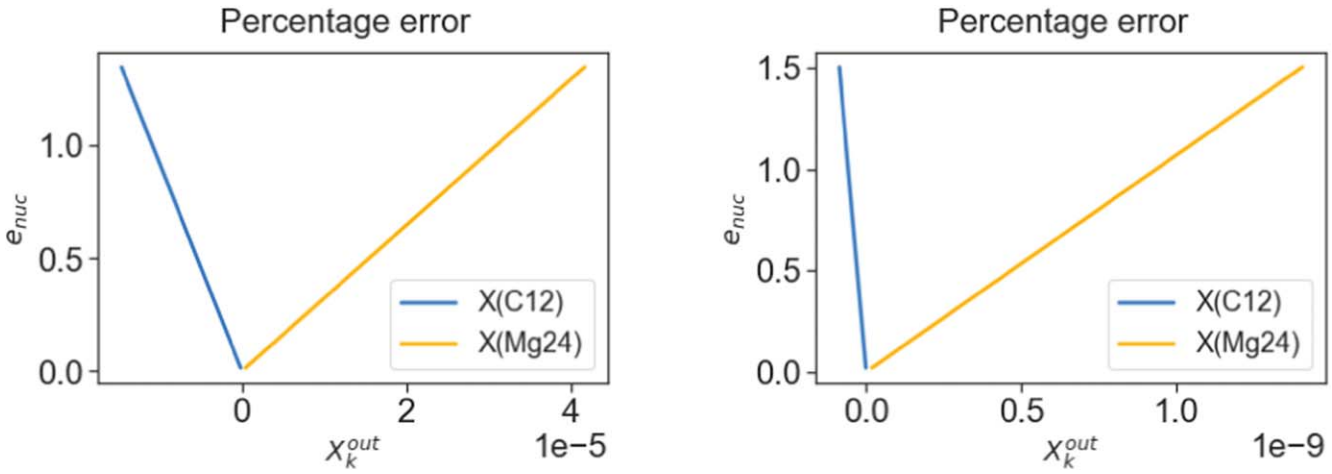
$$\begin{aligned}(\rho h)^{\rm out} &= (\rho h)^{\rm in} - \Delta t \sum_k (\rho\dot{\omega}_k)^{\rm out} q_k \\ &= (\rho h)^{\rm in} + \rho^{\rm out} e_{\rm nuc},\end{aligned} \tag{A3}$$

where $e_{\rm nuc}$ is a function of $(\boldsymbol{X}^{\rm in}, \boldsymbol{X}^{\rm out}, q_k)$ and satisfies

$$H_{\rm nuc} = \frac{\partial e_{\rm nuc}}{\partial t}.$$

Comparing Equations (10) and (A3), it can be easily seen that $\rm DNN_2$ is used to model $e_{\rm nuc}$. The question then is whether this additional $\rm DNN_2$ is necessary given that $e_{\rm nuc}$ can be computed directly using the updated mass fractions from $\rm DNN_1$.

To answer this question, we want to analyze the sensitivity of $e_{\rm nuc}$ with respect to changes in the updated mass fractions (i.e., the accuracy of $\rm DNN_1$). Figure 17 plots the error in $e_{\rm nuc}$ relative to the errors in $X_k^{\rm out}$ at two different initial conditions: (left) at peak $e_{\rm nuc}$ of the flame and (right) approximately 0.2 cm behind the peak. Note that due to mass conservation, any change in $X_{\rm C12}$ will result in the same exact (but opposite) change in $X_{\rm Mg24}$. The left plot in Figure 17 shows that at peak energy generation of 6.0954×10^{12} erg g^{-1}, to achieve ~99% accuracy for $e_{\rm nuc}$ (i.e., ~1% error) requires an error of $\sim 3 \times 10^{-5}$% for $X_{\rm Mg24}$ and $\sim 1.5 \times 10^{-5}$% for $X_{\rm C12}$. Even worse, with the initial conditions shown in the right plot of Figure 17, achieving a ~99% accuracy for $e_{\rm nuc}$ requires errors of $\sim 1 \times 10^{-9}$% and $\sim 1 \times 10^{-10}$% for $X_{\rm Mg24}$ and $X_{\rm C12}$, respectively. In practice, obtaining such high precision for neural networks is difficult and potentially unreasonably expensive due to memory limitations and longer training



$(X^{\rm in}_{\rm C12}, X^{\rm in}_{\rm O16}, X^{\rm in}_{\rm Mg24}) = (0.1483, 0.5, 0.3517)$
$\rho^{\rm in} = 4.2996 \times 10^7$ g cm^{-3}
$T^{\rm in} = 2.2745 \times 10^9$ K

$(X^{\rm in}_{\rm C12}, X^{\rm in}_{\rm O16}, X^{\rm in}_{\rm Mg24}) = (0.4381, 0.5, 0.06195)$
$\rho^{\rm in} = 4.7228 \times 10^7$ g cm^{-3}
$T^{\rm in} = 3.2572 \times 10^9$ K

Figure 17. Percentage error of nuclear energy generation $e_{\rm nuc}$ with respect to percentage error of updated mass fractions $X_k^{\rm out}$ at different initial conditions.

times. Therefore, we conclude that a second DNN is necessary to efficiently model the nuclear energy generation.

ORCID iDs

Duoming Fan https://orcid.org/0000-0002-3246-4315
Donald E. Willcox https://orcid.org/0000-0003-2300-5165
Christopher DeGrendele https://orcid.org/0000-0002-7815-1496
Michael Zingale https://orcid.org/0000-0001-8401-030X
Andrew Nonaka https://orcid.org/0000-0003-1791-0265

References

Alastuey, A., & Jancovici, B. 1978, ApJ, 226, 1034
AMReX-Astro Microphysics Development Team, Bishop, A., Fields, C. E., et al. 2022, AMReX-Astro/Microphysics: Release 22.07, Zenodo, doi:10.5281/zenodo.6787059
AMReX Development Team, Almgren, A., Beckner, V., et al. 2022, AMReX-Codes/amrex: AMReX 22.07, Zenodo, doi:10.5281/zenodo.6788444
Antil, H., Khatri, R., Lohner, R., & Verma, D. 2020, MLS&T, 2, 015003
Bell, J. B., Day, M. S., Rendleman, C. A., Woosley, S. E., & Zingale, M. 2004a, ApJ, 606, 1029
Bell, J. B., Day, M. S., Rendleman, C. A., Woosley, S. E., & Zingale, M. 2004b, ApJ, 608, 883
Bell, J. B., Day, M. S., Rendleman, C. A., Woosley, S. E., & Zingale, M. A. 2004c, JCoPh, 195, 677
Brown, P. N., Byrne, G. D., & Hindmarsh, A. C. 1989, SJSC, 10, 1038
Brown, T. S., Antil, H., Löhner, R., Togashi, F., & Verma, D. 2021, High Performance Computing (Cham: Springer), 23
Colella, P. 1990, JCoPh, 87, 171
Duraisamy, K., Iaccarino, G., & Xiao, H. 2019, AnRFM, 51, 357
Echekki, T., & Mirgolbabaei, H. 2015, CoFl, 162, 1919
Fan, D., Nonaka, A., Almgren, A. S., Harpole, A., & Zingale, M. 2019a, ApJ, 887, 212
Fan, D., Nonaka, A., Almgren, A., et al. 2019b, JOSS, 4, 1757
Graboske, H. C., Dewitt, H. E., Grossman, A. S., & Cooper, M. S. 1973, ApJ, 181, 457
Grimberg, S. J., & Farhat, C. 2020, in AIAA Scitech 2020 Forum (Reston, VA: AIAA), 0363
He, K., Zhang, X., Ren, S., & Sun, J. 2016, in 2016 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (Piscataway, NJ: IEEE), 770
Itoh, N., Totsuji, H., Ichimaru, S., & Dewitt, H. E. 1979, ApJ, 234, 1079
Ji, W., Qiu, W., Shi, Z., Pan, S., & Deng, S. 2021, JPCA, 125, 8098
Karniadakis, G. E., Kevrekidis, I. G., Lu, L., et al. 2021, NatRP, 3, 422
Lye, K. O., Mishra, S., & Ray, D. 2020, JCoPh, 410, 109339
Nonaka, A., Almgren, A. S., Bell, J. B., et al. 2010, ApJS, 188, 358
Papapicco, D., Demo, N., Girfoglio, M., Stabile, G., & Rozza, G. 2022, CMAME, 392, 114687
Paszke, A., Gross, S., Massa, F., et al. 2019, in 33rd Conf. on Neural Information Processing Systems (NeurIPS 2019), ed. H. Wallach et al. (Curran Associates), 8024, http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
Raissi, M., Perdikaris, P., & Karniadakis, G. E. 2019, JCoPh, 378, 686
Sirignano, J., & Spiliopoulos, K. 2018, JCoPh, 375, 1339
Tang, W., Shan, T., Dang, X., et al. 2017, in 2017 IEEE Electrical Design of Advanced Packaging and Systems Symp. (EDAPS) (Piscataway, NJ: IEEE), 1
Timmes, F. X., & Swesty, F. D. 2000, ApJS, 126, 501
Turk, M. J., Smith, B. D., Oishi, J. S., et al. 2011, ApJS, 192, 9
Wang, S., Teng, Y., & Perdikaris, P. 2021, SJSC, 43, A3055
Weaver, T. A., Zimmerman, G. B., & Woosley, S. E. 1978, ApJ, 225, 1021
Zhang, W., Almgren, A., Beckner, V., et al. 2019, JOSS, 4, 1370
Zhang, W., Myers, A., Gott, K., Almgren, A., & Bell, J. 2021, Int. J. High Perform. Comput. Appl., 35, 503
Zingale, M., Katz, M. P., Willcox, D. E., & Harpole, A. 2021, RNAAS, 5, 71
Zingale, M., Woosley, S. E., Rendleman, C. A., Day, M. S., & Bell, J. B. 2005, ApJ, 632, 1021