Neural networks: solving the chemistry of the interstellar medium

L. Branca[1]★, A. Pallottini[1]
[1] *Scuola Normale Superiore, Piazza dei Cavalieri 7, I-56126 Pisa, Italy*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT

Non-equilibrium chemistry is a key process in the study of the InterStellar Medium (ISM), in particular the formation of molecular clouds and thus stars. However, computationally it is among the most difficult tasks to include in astrophysical simulations, because of the typically high (>40) number of reactions, the short evolutionary timescales (about 10^4 times less than the ISM dynamical time) and the characteristic non-linearity and stiffness of the associated Ordinary Differential Equations system (ODEs). In this proof of concept work, we show that Physics Informed Neural Networks (PINN) are a viable alternative to traditional ODE time integrators for stiff thermo-chemical systems, i.e. up to molecular hydrogen formation (9 species and 46 reactions). Testing different chemical networks in a wide range of densities ($-2 < \log n/\mathrm{cm}^{-3} < 3$) and temperatures ($1 < \log T/\mathrm{K} < 5$), we find that a basic architecture can give a comfortable convergence only for simplified chemical systems: to properly capture the sudden chemical and thermal variations a Deep Galerkin Method is needed. Once trained ($\sim 10^3$ GPUhr), the PINN well reproduces the strong non-linear nature of the solutions (errors $\lesssim 10\%$) and can give speed-ups up to a factor of ~ 200 with respect to traditional ODE solvers. Further, the latter have completion times that vary by about $\sim 30\%$ for different initial n and T, while the PINN method gives negligible variations. Both the speed-up and the potential improvement in load balancing imply that PINN-powered simulations are a very palatable way to solve complex chemical calculation in astrophysical and cosmological problems.

Key words: ISM: evolution, molecules – methods: numerical – software: development

1 INTRODUCTION

Thermal and chemical evolution are crucial processes in astrophysical and cosmological environments. Gas cooling, heating, ionization, and photo-dissociation are vital drivers of the evolution of the interstellar (ISM) and intergalactic (IGM) medium. Chemical processes are widely included in theoretical works and numerical simulations, playing key roles in determining the evolution in the early Universe history (Galli & Palla 1998; Glover & Abel 2008), the IGM during the Epoch of the Reionization (Maio et al. 2007; Theuns et al. 1998), galaxy formation and evolution (Pallottini et al. 2017; Lupi 2019), and Giant Molecular Cloud (GMC Kim et al. 2018; Decataldo et al. 2019).

The typical chemical network involves hydrogen, helium, (possibly individual) metals, and molecules, by further coupling of all these various species with the radiation field (i.e. photoheating and photoionization), dynamics (i.e. shocks) and interaction with dust; moreover, going to progressively smaller scales, the chemical processes become more and more complex. To follow a non-equilibrium chemical and thermal evolution in a numerical simulation is necessary an additional set of Ordinary Differential Equations (ODEs) that describe the coupling of the various species.

Various numerical schemes and implementations have been developed to accomplish this task: KROME (Grassi et al. 2014), XDELOAD (Nejad 2005), ASTROCHEM (Kumar & Fisher 2013), ALCHEMIC (Semenov et al. 2010), GRACKLE (Smith et al. 2017), and GGCHEMPY (Ge 2022). The strategy/approximations adopted in a specific implementation can vary, however all schemes have to face similar key problems: i) the chemical ODE system is often stiff and ii) the typical time-scales are much shorter than the dynamical/hydrodynamic time , e.g. $\Delta t_{chem} \ll 10^{-4}\Delta t_{hydro}$. Thus, adopting robust multi-step implicit schemes is needed in order to numerically solve the ODEs with procedural methods (Byrne & Hindmarsh 1987). Such schemes often rely on matrix inversion, which – at face value – has a computational complexity that scales as $O(N_{spec}^3)$, with N_{spec} being the number of species in the network. The matrix associated with chemical ODEs is often sparse, which ameliorate the burden of the inversion (Grassi et al. 2021); nonetheless, systems experience a fast growth of the computational cost in including progressively more complex chemical network, ultimately making the CPU time spent on chemistry a relevant fraction of a hydrodynamical simulation. Further, the precise computational time for the inversion is difficult to estimate given the ODEs, which thus can spoil the load-balancing of the typical astrophysical code.

Thus – in order to try to overcome these limitations – it is interesting to consider emulators as possible fast alternatives to a procedural resolution of chemical networks. The usage of emulators in astrophysics, based on machine learning (ML) and deep learning techniques, has steadily increased its importance in recent years (LeCun et al. 2015). However, so far ML applications in astrophysics are mainly limited to data driven inferences as parameters or classi-

★ lorenzo.branca@sns.it

fications; for example Ucci et al. (2018) use decision trees to infer key ISM physical properties from emission line ratios, Chardin et al. (2019) emulates radiative transfer calculation using Epoch of Reionization simulations as a training dataset, Prelogović et al. (2022) infer astrophysical parameters from 21 cm light cone images by adopting recurrent neural networks, and Dropulic et al. (2021) shows how to predict stellar line-of-sight velocity from Gaia observations of the Milky Way (MW) by training on phase space mock data sets. Currently, there are few attempts to alleviate the cost of computing chemistry by using auto-encoders: Grassi et al. (2021) tries to reduce the complexity of chemical ODE with high dimensionality by compression in a latent space of a smaller dimension; while Grassi et al. (2021) showcased the approach for isothermal models, its generalization seems non-trivial. On the other hand, Holdship et al. (2021) recently implement a fast emulator that includes temperature evolution; it works combining auto-encoders (to reduce the dimensionality) and emulators to follow the temperature and abundances evolution. All these solutions seem encouraging, promoting future usage of emulators for thermo-chemistry in hydrodynamic simulations. However, as these are supervised learning strategies, they require the creation of datasets containing the solution of the ODEs system using a traditional solver.

An intriguing alternative consist in direct approximation of the solution for the chemical system with a neural network (NN), so that once the training is complete the chemical evolution can be directly computed at lower computational cost. Solving differential equations with NN can be done adopting the Neural Ordinary Differential Equation (NODE) model (Chen et al. 2018), that use Recurrent Neural Networks with an high number of layers to emulate the discretization of a differential operator. A more general approach consist in using Physics Informed Neural Network (PINN) (Raissi et al. 2019). The PINN method is very flexible and has been used to solve a large class of differential system associated with wide range of physical problems, as 2D acoustic wave equations (Moseley et al. 2020), turbulent fluid-dynamics (Hennigh et al. 2020) or full radiative transfer calculation (Mishra & Molinaro 2021).

In this work we choose to focus on the PINN, in part because of its flexibility and in part since the mode seems simpler to extend with respect to the original implementation presented in Raissi et al. (2019), by using the eXtended Physics Informed Neural Network (X-PINN, Hu et al. 2022). While various implementation of the PINN frameworks have been developed (Lu et al. 2019; Haghighat & Juanes 2021; Rackauckas et al. 2019; Hennigh et al. 2020), applications in the astrophysical and cosmological context has yet to be explored, except in Chantada et al. (2022), where PINN (in this works named Cosmological Informed neural Networks, CINN) are used to solve the background dynamics of the universe for four different models and then to perform statistical analyses to estimate the values of each model's parameters with observational data.

To our knowledge, only Ji et al. (2021) performs an in-depth study of stiff chemical systems via PINN, by showcasing the solution of standard ODE benchmarks, as the ROBER (3 non-linear ODEs) and POLLU problems (20 non-linear ODEs); however, the adopted reaction coefficients are constant, while for the typical ISM network they can vary by several orders of magnitude, mainly due to temperature[1]; such dependence is key for astrophysical problems and promotes the system from chemical to thermo-chemical, thus making its resolution more complex.

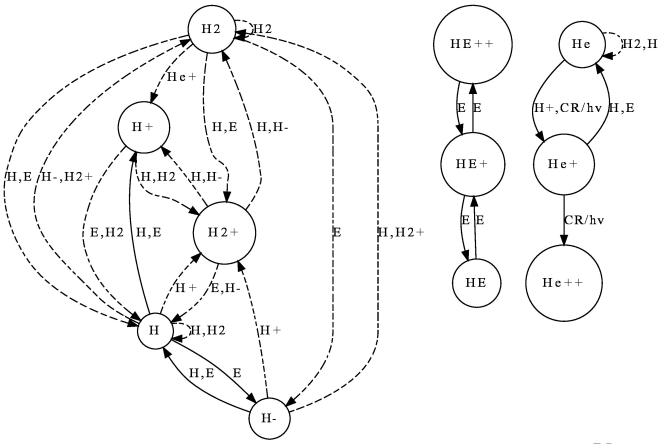

Figure 1. Sketch of the chemical networks adopted in this work. Circles represent the chemical species, while dashed and solid lines connected with arrows represents the two-body and photo-reactions (including cosmic rays). The dashed lines underline reactions involving molecular hydrogen.

In this proof of concept work, we show that it is possible to train a PINN to emulate a complex and realistic chemical network that can be used to simulate the ISM. Our aim is to showcase the PINN scheme in a non-trivial astrophysical context, by proposing an efficient and accurate alternative to the procedural solvers.

In Sec. 2 we introduce chemical networks and their usage in astrophysical context (Sec. 2.1), we present the general framework of the PINN method (Sec. 2.2), and we benchmark PINN for simplified ODE systems (Sec. 2.3). Then, in Sec. 3, we detail and implement the further modelling needed to adopt the PINN framework to treat realistic ISM chemical networks. Sec. 4 presents the results of our models in terms of accuracy, efficiency, and in comparison with procedural solvers. In Sec. 5 we give our conclusions.

2 METHODS OVERVIEW

In this Sec. we provide a summary of the thermal and chemical evolution of the ISM (Sec. 2.1), present the procedural tool used as reference to test our results, and introduce the usage of neural network solvers, in particular focusing on PINN algorithm (Sec. 2.2), we benchmark the model for a chemical and thermo-chemical like ODEs system (Sec. 2.3).

2.1 Chemistry of the Interstellar medium

Given a chemical network, following its evolution entails knowing the density of each species and the temperature at each time step. This chemical and thermal evolution can be described by an ODEs system, with one equation for each species involved and one for the temperature evolution. Each equation is given by a sum of terms that depends on the chemical reaction involved.

In general, reactions depends on the number density of each species ($\mathbf{n} = n_1, \ldots, n_{N_{spec}}$), the temperature ($T$) and the radiative flux (ionizing photons, photo-dissociation photons, cosmic rays, ...) The ODEs system for the chemical species can be written as (e.g. Grassi

[1] Formally, this is correct when only two body reactions are considered; for a general chemical network, further variation are present, e.g. photoionization reactions are proportional to the radiation field and, in cosmic ray induced reactions, their flux plays a similar role.

et al. 2014)

$$\dot{n}_k = \sum_{j \in \text{reaction}_k} \left(a_j \prod_{r \in \text{reactant}_j} n_{r(j)} \right), \quad (1)$$

where n_k is the k-th species, and a_j are the rate coefficients for all the reactions considered in the chemical network.

Considering only 2-body reactions and photo-reactions[2], eq. 1 can be re-written as:

$$\dot{n}_k = A_k^{ij} n_i n_j + B_k^i n_i \,, \quad (2)$$

where $A_k^{ij} = A_k^{ij}(T, \mathbf{n})$ are 2-body reaction coupling coefficients, $B_k^i = B_k^i(\mathbf{F})$ describe the photo-reactions rates, with $\mathbf{F}$ quantifying the photon and cosmic ray flux in various energy bins.

The evolution of the thermal state of the gas is accounted evolving the gas temperature, that depends on the heating and cooling process (chemical, radiative, ...):

$$\dot{T} = \frac{(\gamma - 1)}{k_b \sum_i n_i} (\Gamma - \Lambda) \,, \quad (3)$$

where k_b is the Boltzmann constant, γ is the gas adiabatic index, $\Gamma = \Gamma(T, \mathbf{n}, \mathbf{F})$ and $\Lambda = \Lambda(T, \mathbf{n}, \mathbf{F})$ are the heating and cooling functions, respectively.

In this work we focus on a ISM chemical network originally presented in Bovino et al. (2016) and that has been used for studies on molecular cloud scales (Decataldo et al. 2019, 2020) and the on evolution of high-redshift galaxies (Pallottini et al. 2017, 2019, 2022). The network has $N_{spec} = 9$ species: e^-, H^-, H, H^+, He, He^+, He^{++}, H_2, and H_2^+. Following Bovino et al. (2016), the evolution of the species is regulated by 46 reactions (for a schematic view, see Fig. 1), involving dust processes, i.e. H_2 formation on dust grains (Jura 1975), photo-chemistry, and cosmic rays ionization. In particular, the rates are taken from Bovino et al. (2016): reactions 1 to 31, 53, 54, and from 58 to 61 in their tables B.1 and B.2, photo-reactions P1 to P9 in their table 2.

For the temperature evolution (eq. 3) we account for the following processes: photoelectric heating from dust (Bakes & Tielens 1994), cosmic rays heating (Cen 1992), photo heating, heating/cooling due to exothermic/endothermic reactions, metal line cooling (Shen et al. 2013), Compton cooling from the CMB, molecular H_2 cooling (Glover & Abel 2008), and atomic cooling (Cen 1992). For simplicity, in this work, we adopt a constant solar value for the metallicity ($Z = Z_\odot$, Asplund et al. (2009)) and dust to gas ratio ($f_d = 0.3$, Hirashita & Ferrara (2002)).

Recall that, the two body reactions (A_k^{ij}, eq. 2) depends only on density $\mathbf{n}$ and the temperature T, however the coefficients of (B_k^i, eq. 2), and the heating and cooling terms (Γ and Λ, eq. 3) additionally depends on the flux $\mathbf{F}$.

For simplicity, in this work we consider a Spectral Energy Distribution (SED) of UV/Xray background from Haardt & Madau (2012) at redshift $z = 0$ and adopt MW like cosmic ray flux with rate $\zeta_{cr} = 3 \times 10^{-17} s^{-1}$. This SED is not completely appropriate for the typical ISM conditions, (e.g. see Draine 1978, for the MW), however we adopt it so that all photo-ionizations ($H + \gamma_{h\nu > 13.6eV} \rightarrow H^+ + e$, ...) in our chemical network are active, i.e. this choice allow us to robustly test all the reactions in the model.

[2] Neglecting secondary higher order processes, also reaction involving cosmic rays can be approximated with the same formalism, see e.g. Bovino et al. (2016).

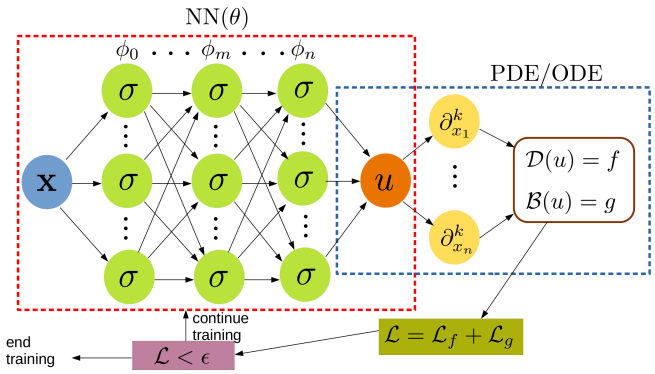

Figure 2. General scheme for a feed-forward Neural Network (NN) architecture. The aim is to resolve the PDE/ODE system (eq. 4) of the set of variables $\mathbf{x}$. This variables are used as the input for the NN: $\mathbf{x}$ passes through the Physics Informed NN (PINN) layers defined by the activation functions σ (eq. 6) and the set of parameters θ (eq. 7). The NN returns the emulated solution u, which is tested against the PDE/ODE system (eq. 4) by evaluating the residuals via the loss function (eq. 8). Parameters θ are updated and the process is repeated until convergence is reached (eq. 11).

Various implementations/schemes can be adopted to solve a chemical network. As a reference for this work, we adopt the flexible code KROME[3] (Grassi et al. 2014), which is a framework that – given an input chemical network – generates the code to solve the associated ODE system. To solve the system KROME use LSODES, which is included in ODEPACK (Hindmarsh 2019). LSODES is an implicit, robust, multistep, iterative high order solver (5 by default) that can take advantage of the sparsity of the Jacobian matrix of the ODEs. The default KROME relative and absolute tolerances are fixed at 10^{-4} and 10^{-20} respectively.

In this work, we adopt KROME i) to build the ODEs structure (eq.s 2 and 3) for our PINN scheme (Sec. 3) and ii) to test our results during the validation phase (Sec. 4).

2.2 Physics Informed Neural Network

In general, we can write a set of partial differential equations (PDE)/ODE[4] in the form

$$\mathcal{D}(u(\mathbf{x})) = f(\mathbf{x}), \ \forall \mathbf{x} \in \Omega, \quad (4a)$$
$$\mathcal{B}(u(\mathbf{x})) = g(\mathbf{x}), \ \forall \mathbf{x} \in \partial\Omega \,, \quad (4b)$$

where $\mathbf{x}$ is the set of independent variables, $\mathcal{D}$ is the differential operator of the PDE/ODE, $\mathcal{B}$ is a constrain operator – i.e. it represents the boundary/initial conditions (BC/IC) – and $u(\mathbf{x})$ is the solution of the PDE/ODE system.

Our aim is to approximate the solution of the system with a neural network (see Wang & Raj 2017, for a review analyzing the theoretical and practical aspects). In principle, this is possible because of the universal approximation theorem (Cybenko 1989), since multi-layer feed-forward neural networks are capable of approximating any Borel measurable function (Hornik et al. 1989). Namely, it is formally possible to replace the PDE/ODE solution $u(\mathbf{x})$ with the output of

[3] https://bitbucket.org/tgrassi/krome/src/master/

[4] While in the present we are focusing on ODE systems associated with a chemical network, the PINN scheme can also be applied to PDE.

the neural network $u_{net}(\mathbf{x}, \theta)$, which can be written as

$$u_{net}(\mathbf{x}, \theta) = \mathbf{W}_n(\phi_{n-1} \circ \phi_{n-2} \circ ... \circ \phi_1\phi_0)(\mathbf{x}) + \mathbf{b}_n\,, \tag{5}$$

i.e. the composition of the action of successive layers ϕ_i

$$\phi_i(\mathbf{x}) = \sigma(\mathbf{W}_i\mathbf{x} + \mathbf{b}_i)\,, \tag{6}$$

where σ is the activation function, $\mathbf{W}_i$ is the matrix of the weights and $\mathbf{b}_i$ is the bias vector. It is convenient to cluster $\mathbf{W}_i$ and $\mathbf{b}_i$ as

$$\theta = \{\mathbf{W}, \mathbf{b}\}\,, \tag{7}$$

i.e. the set of the parameters to be trained. For the PINN method, the necessary condition is that u_{net} is derivable at least p times – i.e. $u_{net(\mathbf{x},\theta)} \in C^p$ – with p being the maximum derivative order for the operator $\mathcal{D}$ in eq. 4a.

Using the auto-differentiation (Gunes Baydin et al. 2015) it is possible to define the partial (or total) derivative $\partial_i u_{net}(x_i, \theta)$ by selecting a proper activation function σ, e.g. a typically a sigmoid or hyperbolic tangent. The activation functions are often chosen among smooth (C^∞) analytical functions, so that the derivatives calculated by auto-differentiation are correct at machine precision, which is important for the numerical stability in the evaluation of the PDE/ODE terms.

The core of the PINN scheme consist in evaluating the residual in the space of the neural network parameters θ (eq. 4a). The aim is to optimize the *loss function* $\mathcal{L}$, that can be defined as follows:

$$\mathcal{L}_f(\theta) = d_\Omega(\mathcal{D}(u_{net}(\mathbf{x}, \theta)) - f(\mathbf{x}, \theta)), \tag{8a}$$

$$\mathcal{L}_g(\theta) = d'_{\partial\Omega}(\mathcal{B}(u_{net}(\mathbf{x}, \theta)) - g(\mathbf{x}, \theta)), \tag{8b}$$

$$\mathcal{L}_{tot}(\theta) = \mathcal{L}_f(\theta) + \mathcal{L}_g(\theta) \tag{8c}$$

where d_Ω and $d'_{\partial\Omega}$ are positive metrics that can be used to evaluate the distance to the exact solution u and IC/BC, e.g. the L_2 norm ($\|\cdot\|_2^2$). Including the ODE/PDEs residuals in the loss function is the reason why the algorithm is called *physics informed*.

In this context the residuals takes the place of what labels would be in a context of supervised learning, making the algorithm completely data independent (i.e. in this case, the solutions obtained with a numerical solver). Note that the metrics in eq.s 8 can be very general, e.g. for a ODE/PDE system each equation (and associated IC/BC term) can be weighted individually (see later Sec. 3.3, in particular eq. 25).

Once the loss function is defined, an optimization algorithm is applied on the parameters set θ. Various solution have been proposed for different problem, the most classical being the Stochastic Gradient Descent (SGD), i.e. a Gradient Descent (GD) algorithm applied on a random subset of the parameter space (Robbins & Monro 1951). The general idea beyond the GD method is to evaluate the gradient in a set of points of the domain and then mediate on them to find the optimal direction to minimize the loss function. However, when the domain is to huge (in terms of memory or in computational cost for the loss function gradient evaluation) or when the algorithm is distributed on more than one device (i.e. multi GPUs), the SGD is preferable: the method is equivalent to GD but on a subset of points (usually named *batch*). The parameters are update as follows:

$$\theta_{i+1} = \theta_i - \eta\nabla\mathcal{L}(\theta_i)\,, \tag{9}$$

where η is a scalar called learning rate. In this work we use the more sophisticated SGD variant with adaptive learning and gradient momentum rate ADAM (Kingma & Ba 2014) where the parameters update is defined as:

$$\theta_{i+1} = \theta_i - \eta\frac{m}{\sqrt{v} + \epsilon}\,, \tag{10}$$

where m and v are the first and second moments of the gradient, and ϵ is the smoothing term. The procedure can be considered concluded if the algorithm finds a set of parameters named θ^* such that for a given positive scalar δ the following condition is met:

$$\mathcal{L}_{tot}(\theta^*) < \delta\,, \tag{11a}$$

which implies that:

$$u_{net}(\mathbf{x}, \theta^*) \simeq u(\mathbf{x}) \tag{11b}$$

where δ is the required tolerance, that could be fixed a priori or determined on the fly, i.e. if the loss function is not decreasing for a large number of epochs.

In Fig. 2 we show the sketch of the feed-forward architecture of the PINN that is used in this work. To summarize, the proposed method follows these steps:

0) We define a sampling of the input parameter space $\mathbf{X}$ and we randomly initialize the PINN parameters θ.
1) Given θ, we evaluate the solution of the PDE/ODEs system (eq.s 4) on the input space $\mathbf{X}$ using the PINN, i.e. $u = \mathrm{NN}_\theta(\mathbf{X})$
2) We compute the partial derivatives that appear in the PDE/ODEs via automatic differentiation, thus we evaluate the residuals in the loss function defined in eq. 8.
3) We update the values of parameters θ with the optimizer in eq. 10, repeating 1)-3) until the convergence is reached (eq. 11).

The architecture is shown in Fig. 2 is the first introduced in the literature, and therefore the simplest. However, in recent years, more complex schemes have been drawn, such as: Fourier Network (FN, Tancik et al. (2020)) and its variations Modified Fourier Network (MFN, Wang et al. (2021)), Highway Fourier Network, inspired by Highway Network first introduced in (Srivastava et al. 2015) that are designed explicitly to take in account high frequency variations of the solution. Other variants are: SInusoidal REpresentation NEtwork (SIREN Sitzmann et al. (2020)), mainly designed for periodic-like solutions, MeshfreeFlowNet Jiang et al. (2020)), designed for super-resolution tasks and the Deep Galerkin Method (DGM (Sirignano & Spiliopoulos 2018)) which is inspired by Long Short Term Memory (LSTM) architecture but optimizes for PDEs computing.

2.3 Model benchmark

Before trying to solve the ISM chemistry via the PINN method, we validate the algorithm on a simplified and easily reproducible problem.

For this benchmark, we select an ODE system similar in shape to eq. 2. To have a formal description of the system, we rearrange the coefficient tensor as follows:

$$M_k^j = A_k^{ij} n_i + B_k^i\,, \tag{12}$$

so that we write the ODE system in a compact (and standard) way

$$\dot{n}_k = M_k^j n_j\,, \tag{13}$$

where $\mathbf{n} = (\mathrm{n}_1, \mathrm{n}_2, \mathrm{n}_3)$ is a 3-dimensional vector of *fake species*, and M is the rate coefficients matrix.

To give a proof of the performance of the model, we adopt the following shape for the matrix of coefficients

$$M = \begin{pmatrix} -(k_1 + k_3 n_3) & 0 & k_3 n_1 \\ k_1 + +2k_3 n_3 & -k_2 & 2k_3 n_1 \\ -k_3 n_0 & k_1 & -k_3 n_0 \end{pmatrix} \tag{14a}$$

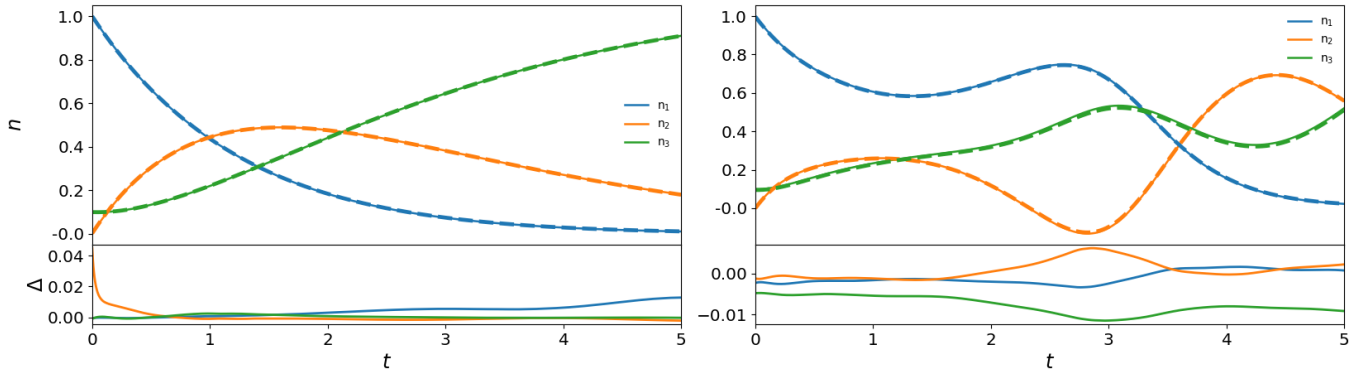

Figure 3. Benchmarks between the PINN and a procedural solver (Backward Differentiation Formula, BDF) for two simple ODEs systems with coefficients matrices given in eq. 14a. Left (right) panel show the system without (with) explicit time dependence that is given in eq. 14b (eq. 14c)). For each system, in the upper panel we show the comparison between our model (solid line) and BDF solver (dashed line), while the lower panel shows the absolute errors between the two methods (eq. 16).

and consider two different cases. For case **1)** we select rate coefficients as

$$\mathbf{k}_{cons} = [0.8, 0.5, 0.2]\,, \tag{14b}$$

i.e. the coupling matrix depend only on density $M = M(\mathbf{n})$, which mimics a temperature-independent system. For **2)** we adopt

$$\mathbf{k}_{t-dep} = [0.8 - \sin(t), 0.5 + \cos(t), 0.2 + \sin(2t)]\,, \tag{14c}$$

so that $M = M(\mathbf{n}, t)$, which mimics a temperature-dependent system. The initial conditions of the fake species for both models are $n_1^{in} = 1$, $n_2^{in} = 0.001$ and $n_3^{in} = 0.1$. For both models we consider the *fake time* interval [0,5).

A key obstacle in solving a chemical (or thermo-chemical) evolution of a system is represented by stiffness. The ratio $\mathcal{S}$ can be used to quantify the stiffness of the system

$$\mathcal{S} = \frac{|\mathrm{Re}(\lambda)|_{max}}{|\mathrm{Re}(\lambda)|_{min}}\,, \tag{15}$$

where $\mathrm{Re}(\lambda)_{max/min}$ are the largest/smallest value of the real part of eigenvalues of the matrix. For both our test cases the coefficients change as the system evolves, thus we compute the mean stiffness ratio in the *fake time* interval, obtaining $\langle \mathcal{S}(M(\mathbf{n})) \rangle \simeq 2.8 \times 10^3$ and $\langle \mathcal{S}(M(\mathbf{n}, t)) \rangle \simeq 2.9 \times 10^5$ for **1)** and **2)**, respectively. The difference in the stiffness ratio reflects the increase in complexity for a variable temperature system.

Thus, for the model with no explicit time dependence, we choose a neural network architecture with 6 layers of 64 neurons each. We adopt $2^{10} \simeq 10^3$ training points and – to assure convergence of the loss function – we run the training for 1.5×10^4 epochs, which takes $\sim$ 0.2CPUhr on a INTEL i7-9700 CPU using the ADAM algorithm introduced in Sec. 2.2. Instead, for the model with explicit time dependence, the setup consists of 6 layers of 128 neurons each, $2^{13} \simeq 8 \times 10^3$ training points, and 3×10^4 epochs ($\sim$ 1.1CPUhr), reflecting the increased complexity. For this benchmark we have selected a feed-forward architecture, that is the simpler setup which can achieve satisfactory results. In order to compare our model with a procedural solver, we solve the systems with the Backward Differentiation Formula (BDF) method adopting the SCIPY implementation (Virtanen et al. 2019).

In Fig. 3 we show the benchmark validation. Qualitatively, the reconstruction of the trained models (dashed line) seem good in both cases. Quantitatively, we can define the errors as[5]

$$\Delta = y_{BDF} - y_{NN}\,. \tag{16}$$

The evolution of the error is show in the bottom panels of Fig. 3. For the model without/with explicit time dependence the mean errors are $\langle \Delta \rangle \simeq 10^{-3}$ and 2×10^{-3} respectively, which we consider a satisfactory result for our benchmark. Interestingly, Fig. 3 shows no evidence of a growth of the error during the time evolution, as it might be expected from a regular procedural method. Such behaviour is intrinsic of the PINN method, as the training algorithm aims to minimizing the residuals on the entire time domain simultaneously.

3 NEURAL NETWORKS FOR ISM CHEMISTRY

While the benchmark shows good potential in applying of the PINN method for ISM chemistry, multiple aspects need additional considerations. Specifically, the initial conditions must be generalized (Sec. 3.2), the chemical network requires further consideration (Sec. 3.1), the loss function requires modifications (Sec. 3.3), the neural network must be improved, and the training process needs more careful attention (Sec. 3.4).

3.1 ODE structure of the chemical networks

For the selected chemical network (Sec. 2.1), the associated ODEs system has non-trivial dependence of the coefficients on the temperature. This is determined by the elements of the interaction matrix M_k^i (defined in eq. 12): an example is shown in Fig. 4, where we plot a subset of the matrix element as a function of temperature. In the allowed temperature range, the selected M_k^i can vary by more than about 10 order of magnitude. The stiffness of the system is higher with respect to the benchmark (eq. 14); for instance, for normal ISM densities (i.e. the IC used in Sec. 3.2), the stiffness can reach values of $\mathcal{S} \sim 10^{16}$ for $T < 2.5 \times 10^4$K and $\mathcal{S} \sim 10^4$ for $T > 2.5 \times 10^4$K. This is a hint that i) the network architecture must be expanded ii) the training will be more expensive, also because of the generalized initial conditions.

Given these expectations and in order to explore different strategies

[5] For the simple benchmark, it is convenient to avoid using relative error, since all the *fake species* densities are order unit.

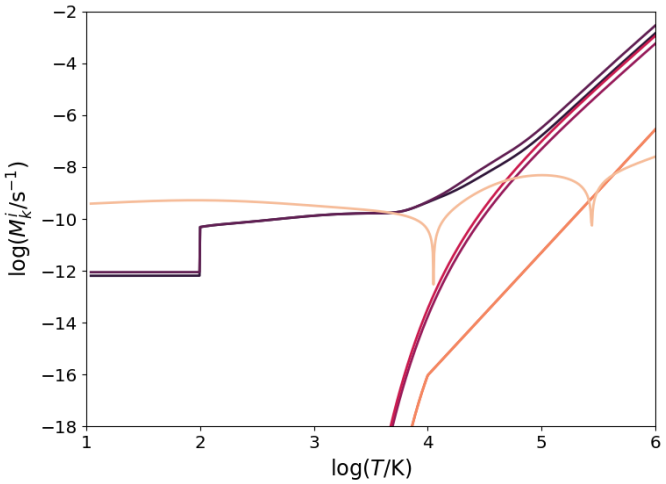

Figure 4. Example of temperature (T) dependence of matrix elements (M_k^i, definition in eq. 12.) of the adopted chemical network. We select 6 representative M_k^i out of the 62 total non-null element (see Sec. 2.1)

for both the neural network and training, for the chemical system we consider both a *molecular* network (introduced in Sec. 2.1) and a reduced *atomic* network. The *atomic* network is a simplified version of the *molecular* one, namely, it does not include the chemistry of molecular hydrogen and its cooling, simplifying both the reaction network and the temperature evolution. With respect to *molecular*, in *atomic* the number of species decreases from 9 to 7 and the number of reactions decreases from 46 to 24.

3.2 Setting up the initial conditions

The PINN model adopted in the benchmark (Sec. 2.3) is solved only with a specific set of fixed initial conditions: this is a major limitation. Adopting the same approach for ISM chemistry would considerably limit the applicability of the model, since training is expensive and would be needed for each different thermodynamic configuration. We can overcome this limit by generalizing the model as follows.

Recalling eq. 4, in the case of ODEs the operator it follows that $\mathcal{B}(\mathbf{x}) = (x_1(t_0), x_2(t_0), ..., x_k(t_0))$. Formally, to generalize for arbitrary initial conditions, we can promote the initial values from a scalar to a function $x_i(t_0) \rightarrow f_i(t_0)$ that map the **IC**s in a desire range, e.g. for temperature, $f_T(t_0) = [20, 10^6]$K. The procedure has been proposed by Flamant et al. (2020) and greatly increases the dimensionality of the problem; however, as shown by Mishra & Molinaro (2020), the PINN does not suffer too much from the *curse of dimensionality*, e.g. can be trained in a 100-dimensional space in case of heat equation. Summarizing, we can use this strategy to vary both the initial thermodynamic state (T_{in}, n_{in}) and the fractions of each species, C_k. We expect the procedure to increase the training time, however note that – once the training is completed – the computational time for predictions is mostly unaltered, since the latter depends only on the network architecture, i.e. the number of neurons and links.

Once trained our model maps the input space (time and initial conditions) to the evolution of the desired quantities:

$$(t, \mathbf{IC}) \rightarrow \mathrm{PINN}(t, \mathbf{IC}) = (\mathbf{n}(t, \mathbf{IC}), T(t, \mathbf{IC})) \,. \tag{17}$$

Stating eq. 17 differently, knowing the initial densities and temperature at the initial time $t = t_{in}$, the PINN return the evolved quantities at $t = t_{out}$:

$$[n_k(t = t_{in}), T(t = t_{in})] \rightarrow \mathrm{PINN}(t_{out}, n_k(t = t_{in}), T(t = t_{in})) = \\ = [n_k(t = t_{out}), T(t = t_{out})] \tag{18}$$

with k indexing the species.

Note that the adopted generalization for the IC (eq. 17) is not directly applicable to arbitrary PDE systems, which require more sophisticated and problem dependent techniques (i.e. Nakamura et al. 2021).

Therefore, in this paper we distinguish 2 type of model: *single models*, that have fixed initial condition, and *general model*. Although single models are subclasses of the general models, in the spirit of proof of concept it is convenient to keep the cases separate, since different strategies are adopted to achieve convergence in the training.

For *single models*, we adopt $T = 10^3$K as the initial temperature for both networks (*atomic/molecular*), while the total density ($n_{tot} \equiv \sum_k n_k$) and individual abundances ($C_k \equiv n_k / \sum_k n_k$) are set as follows. *atomic*: $n_{tot} = 90.4\mathrm{cm}^{-3}$, $C_{\mathrm{H}^-} = 0.0015$, $C_{\mathrm{H}} = 0.69$, $C_{\mathrm{He}} = 0.288$, $C_{\mathrm{H}^+} = 0.0069$, $C_{\mathrm{He}^+} = 0.0029$ and $C_{\mathrm{He}^{++}} = 0.0008$; *molecular*: $n_{tot} = 100\mathrm{cm}^{-3}$, $C_{\mathrm{H}} = 0.6241$, $C_{\mathrm{H}^-} = 0.001$, $C_{\mathrm{H}_2} = 0.104$, $C_{\mathrm{He}} = 0.26$, $C_{\mathrm{H}^+} = 0.0062$, $C_{\mathrm{H}_2^+} = 0.001$, $C_{\mathrm{He}^+} = 0.0026$ and $C_{\mathrm{He}^{++}} = 0.0007$. In both cases, C_{e^-} is set to ensure charge neutrality.

For *general models*, the initial values for the temperature, total density, and individual abundances span the following ranges:

$$20 \leq \frac{T}{\mathrm{K}} \leq 10^6 \tag{19a}$$

$$10^{-2} \leq \frac{n_{tot}}{\mathrm{cm}^{-3}} \leq 10^3 \tag{19b}$$

$$10^{-6} \lesssim C_k \lesssim 1 \,, \tag{19c}$$

where C_k are chosen such that global charge neutrality is respected and the total hydrogen (helium) fraction is $X \simeq 70\%$ ($Y \simeq 30\%$).

3.3 Loss function definition

A crucial aspect in any ML approach is the design of the loss function. Directly evaluating the mean square errors using the metric in eq. 8 cannot capture the fine structure of the underlying solution. This is mainly driven by the large dynamical range spanned by the density of different species. For instance, for typical ISM conditions H^- is usually about 8 order of magnitude lower than H: using a simple (e.g. uniformly weighted) loss function would ignore the variation of H^-. Ideally, to maximize the convergence efficiency, all components of the loss function must be of the same order of magnitude. However, unlike in a supervised learning problem, the code does not know in advance the abundances (and temperature) during the evolution.

With this in mind, we have adopted the following strategy to model the loss function. The first step consists in considering the evolution of the logarithm of the abundances (eq. 2) and temperature (eq. 3); this "feature normalization" is a relatively standard technique for ML. Moreover, we solve the ODEs system in a logarithmic time scale; with this precaution we can better capture the sudden ($t \lesssim 10^2$ yr) large (more than one order of magnitude) variations that species can experience for some initial conditions.

To summarize, in linear space the ODE in eq. 2 has the following residuals:

$$\mathcal{R}_{n_k} = \dot{n}_k - (A_k^{ij} n_j + B_k^i) n_i \,. \tag{20}$$

We perform the change of variables:

$$y_k = \log(n_k/\mathrm{cm}^{-3}) \tag{21a}$$
$$\tilde{T} = \log(T/\mathrm{eV}) \tag{21b}$$
$$\tau = \log(t/\mathrm{s})\,. \tag{21c}$$

Thus eq. 2 can be rewritten as

$$\dot{y}_k = \frac{10^{\tau}}{10^{y_k}}(A^{ij}_k n_j + B^i_k)n_i\,. \tag{22}$$

and the residuals are naturally defined as

$$\mathcal{R}_{y_k} = \dot{y}_k - \frac{10^{\tau}}{10^{y_k}}(A^{ij}_k n_j + B^i_k)n_i\,. \tag{23}$$

The relation between the residuals computed in the logarithmic space and in the linear space is

$$\mathcal{R}_{y_k} = \frac{t}{n_k}\mathcal{R}_{n_k}\,. \tag{24}$$

Dividing the residuals by n_k, the loss function should be balanced, even when the sum is extended to species with order of magnitude difference between abundances (e.g. H^- vs H).

Despite these precautions, the loss function remains complex and consequently the training procedure can present mythologies, e.g. excessive stiffness in the parameters update. To mitigate this problem, the weights λ_i are modified during the training, using the procedure detailed in Wang et al. (2021). Summarizing, we write the loss function as

$$\mathcal{L}(\theta) = \sum_{i=1}^{N}\lambda_i\mathcal{L}_i(\theta)\,, \tag{25}$$

where the sum is extended to $N = 2(N_{spec}+1)$ to account for the ODE residuals for the chemical species and the temperature (eq. 8a) and initial conditions (eq. 8b). The terms λ_i are adaptively regulated utilizing the back-propagated gradient statistics during model training. By experimenting different approaches, we noticed that it is convenient to adopt such adaptive regulation of the weights of the loss, mainly because it improves the convergence during the initial phases of training; this is later discussed in Sec. 4.1.

3.4 Network architecture and training strategy

In general, the optimization of neural network topology and the hyperparameters tuning is a complex task. Although there are tools designed for hyperparameters optimization, an accurate calibration of these is usually not practical, especially if the training procedure is time expensive. So we perform an optimization based on small variations of a fiduciary setting.

For our models, we adopt either a simple Feed-forward Neural Network (FNN, Fig. 2) or a more advanced DGM architecture (Sirignano & Spiliopoulos 2018). A DGM follows essentially the same 0)-3) steps described in Sec. 2.2, however the parameters update is more complex (see sketch in Fig. 5). At depth i^{th} first we calculate the output of a standard dense layer

$$\phi_i(\mathbf{x}) = \sigma(\mathbf{W}_i\mathbf{x} + \mathbf{b}_i)\,. \tag{26}$$

The result is then processed through a DGM layer by computing Z_i, G_i, R_i, and H_i as follows:

$$Z_i = \sigma(\mathbf{U}^{(z)}_i\mathbf{x} + \mathbf{W}^{(z)}\phi_i + \mathbf{b}^{(z)}_i) \tag{27a}$$
$$G_i = \sigma(\mathbf{U}^{(g)}_i\mathbf{x} + \mathbf{W}^{(g)}\phi_i + \mathbf{b}^{(g)}_i) \tag{27b}$$
$$R_i = \sigma(\mathbf{U}^{(r)}_i\mathbf{x} + \mathbf{W}^{(r)}\phi_i + \mathbf{b}^{(r)}_i) \tag{27c}$$
$$H_i = \sigma(\mathbf{U}^{(h)}_i\mathbf{x} + \mathbf{W}^{(h)}(\phi_i \odot R_i) + \mathbf{b}^{(h)}_i)\,, \tag{27d}$$

where U and $\mathbf{W}^{(\dots)}$ are weight matrices, b are biases, and $\odot$ represent the Hadamard (element-wise) product. For depth $(i+1)^{th}$, the outputs are then combined via

$$\phi_{i+1}(\mathbf{x}) = (1 - G_i)\odot H_i + Z_i \odot \phi_i \tag{28}$$

to define the next dense layer. A DGM layer requires roughly eight times more memory than standard FNN, since there are eight weight matrices per layer. The main advantage is the ability to capture the *sharp turns* of the underlying solution, as argued in Sirignano & Spiliopoulos (2018).

For the activation function, we have chosen an adaptive version of the sigmoid function, $\sigma(a,x)$:

$$\sigma(a,x) = \frac{1}{1+e^{-ax}} \tag{29}$$

where x is the input value and a is a NN adaptive parameter, which is an additional parameter that is optimized during the training. As shown in Jagtap et al. (2020), this is a useful strategy for dynamical problems that present a wide range of time scales, thus particularly in our case where the reaction rate that can vary by several order of magnitude (see Fig. 4). The overall architecture design is summarized in Fig. 5.

To fully exploit our hardware, the learning procedure is distributed on multi-GPUs (up to 4) and the train domain is divided in several mini-batches (4-32 per GPU). The optimizer is ADAM, with initial learning rate, η and decaying scheduling dependent on the specific model [6], which needs a gradient aggregation correction for the parameters update. Furthermore, the initial learning rate, η, is subjected to a gradual warm-up, following Goyal et al. (2018), which makes η less dependent on the user initialization.

The final aspect to address is setup for the training points. The higher the number of training point, the better the variability of the solution is captured. However, increasing such number does increase the training time and/or the required memory. It is therefore necessary to find a good trade-off between the amount of training points and computational cost. The number is chosen empirically, however, in the most general case of variable initial conditions described in Sec. 3.2, is shown in (De Ryck & Mishra 2021) that it grows less then exponentially. This result makes us confident that, net of our hardware availability, it is theoretically possible to make eq. 8 converge as the number of chemical species increases. In this work we find that $\sim 10^4 - 10^5$ points per GPU for a single batch gives a reasonable balance. The training points distribution follow the Halton sequence (Halton & Smith 1964); to increase the sampling density, we change the points cloud during the training after a fixed number of iterations, here fixed to 1000 epochs. In total we consider 4 cases, which originates from the combination of *single/general* models (IC) with *molecular/atomic* chemical networks. A summary of the parameters for each model is given in Tab. 1. The development of our codes, was perfermod using MODULUS (Hennigh et al. 2020), a TENSORFLOW (Abadi et al. 2015) based tool specific for PINN design.

4 RESULTS

To present our results, first we analyze the training procedure, by studying the convergence of the various model (Sec. 4.1) and detailing the trends for individual ions (Sec. 4.2). Then, we validate our models against procedural solvers by adopting KROME as our ground

[6] Note that the learning rate η can be tuned manually on-the-fly for stability reasons.

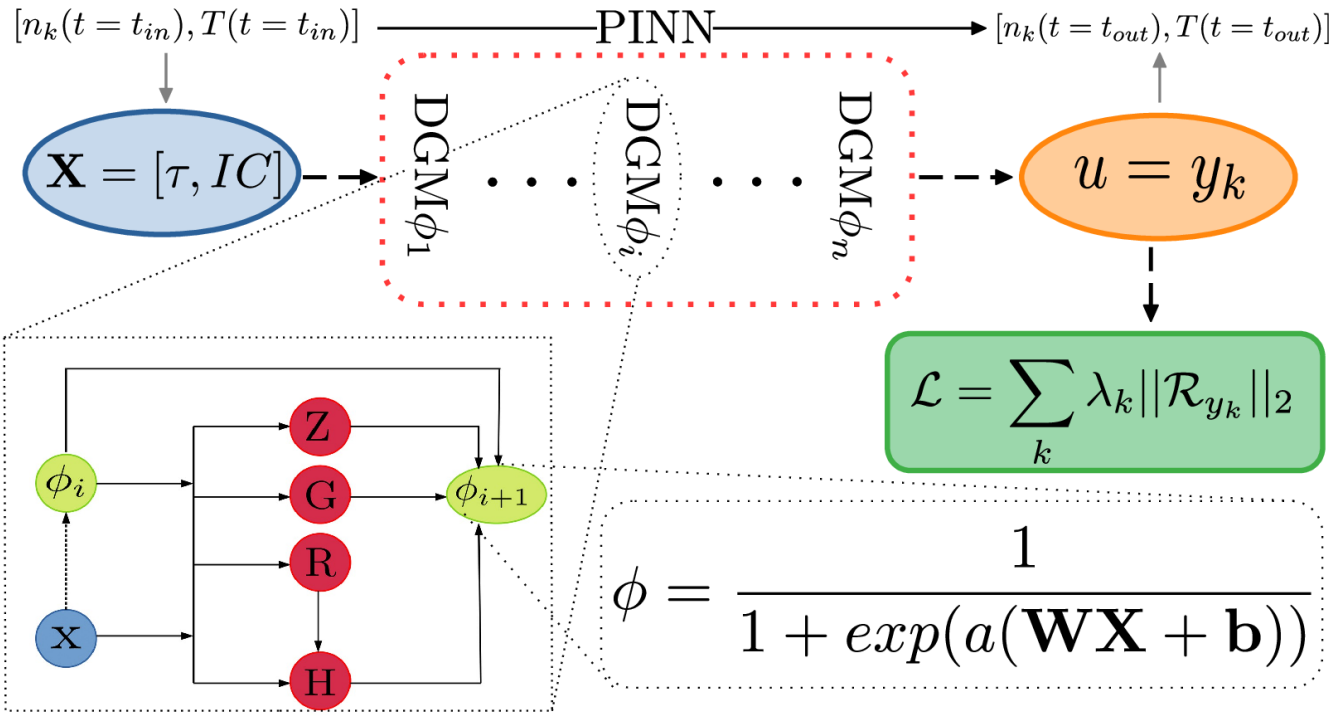

Figure 5. Representation of the PINN model to solve ISM chemistry. On the surface, the PINN takes the densities (n_k) and temperature (T) at initial time t_{in} and returns the evolved quantities at time t_{out} (eq. 18); the NN architecture is detailed as follows. **Top**: the model inputs **X** (logarithmic time τ and logarithmic initial conditions IC, eq.s 21) pass through the network layers (DMGϕ_i) and gives the outputs u (the logarithm of abundances and temperature, y_k), which is trained to minimize the loss function ($\mathcal{L}$), which is as a linear combination of weighed (eq. 25) residuals (eq. 23). **Bottom left**: inset representing the Deep Galerkin Network (DGM) layer (eq.s 27); X represents the input data that enters the first layer ϕ_i if $i = 1$ (dashed line, eq. 26). **Bottom right**: inset showing the action of the dense layer ϕ, that is designed using an adaptive sigmoid function (eq. 29), which depends on the weights **W**, the biases **b**, and the adaptive hyperparameter a.

	atomic single	molecular single	atomic general	molecular general
chemical system				
N_s	7	9	7	9
N_{rea}	24	46	24	46
IC fix	yes	yes	no	no
NN architecture				
N_{layers} (DGM)	6	8	8	10
N_{batch}	16	32	64	128
training points (per batch)	1.4×10^5	1.4×10^5	2×10^5	2×10^5
convergence time (GPUhr)	120	820	1224	1920

Table 1. Reference parameters for the chemical and neural network adopted in the present work. The difference between *atomic* and *molecular* chemical networks is detailed in Sec. 3.1. The difference between *single* and *general* model lays in the type of initial conditions, and is introduced in Sec. 3.2. The different combinations give a total of 4 models.

truth: first we focus on the *single atomic* network (Sec. 4.3), then we give a comparative analysis of the remaining models (Sec. 4.4).

4.1 Training: convergence

An overview of the PINN training can be appreciated in Fig. 6. In the upper panel, we plot $\mathcal{L}$ as a function of the training epochs for all models. For all models, the trend of loss functions during the training phase appears qualitatively similar in shape, in particular with a sharp decline in the early epochs. In general, more complex models (from *single atomic* to *general molecular*) require more training resources to reach an acceptable convergence. In particular the *single atomic* model is much easier to emulate, in fact the convergence is about two orders of magnitude better with $\sim 5 \times 10^5$ less training epochs. Furthermore,

Recall that the training is stopped when $\mathcal{L}$ is below a certain threshold, however there is no generally accepted value for such threshold. Being able to check the residual interactively, it is possible to have an on-the-fly evaluation of the reliability of the approximation. Additionally, note that if the loss function is stable for a long period ($\sim 10^5$epochs), it is possible – and convenient – to stop the training and change some hyper-parameters. In particular, this is an important optimization strategy to adopt in the final stages of the training: when the ADAM algorithm is no longer able to lower the loss function, it is possible to switch to the second order method L-BFGS that uses the hessian matrix; this algorithm is slower but more powerful in the minimization, thus can be used during the last epochs to refine the convergence (Liu & Nocedal 1989; Schraudolph et al. 2007); we adopt such strategy and report the effect later in Sec. 4.3. further, to quantify the importance of the adaptive weights (see Sec. 3.3), we perform a control training for the *single molecular* model; without

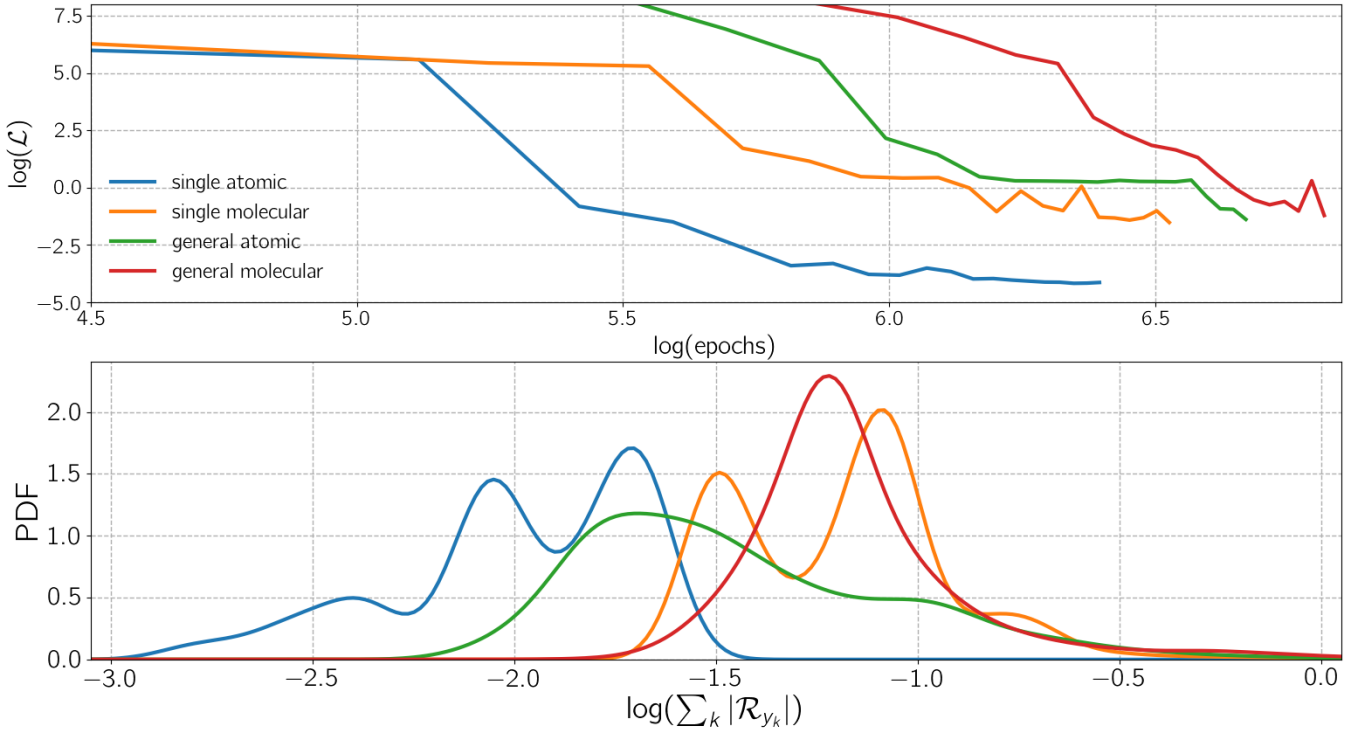

Figure 6. Summary of the neural network training. **Top panel**: Loss function ($\mathcal{L}$, eq. 8) evolution as a function of the training epoch. Each model is indicated with a different colour, according to the caption. See Tab. 1 for the main parameters of the models. **Botton panel**: Probability distribution functions (PDF) of sum of the residuals ($\sum_k |\mathcal{R}_k|$, eq. 23) after the training. PDFs are computed by using a kernel density estimation of the absolute value of the logarithm of the residual.

adaptive weights, after $\sim 3.2 \times 10^5$ epochs the loss function value is $\sim 20\times$ larger with respect to results shown in Fig. 6.

In the lower panel of Fig. 6 we plot the PDF of the sum of the residuals at the last training epoch. In general the peak is around $\sim 10^{-1.5}$ for all models except , with high value tails that can reach up to $\sim 10^{-1}$, with larger residuals for models with higher complexity. Table 1 summarizes the main hyper-parameters and the convergence time for each model. Apart from the *single atomic* model, the training time is around thousands of hours (despite the high performance of the GPU used). However, real time can be linearly reduced with the use of multi-GPUs. We also noticed that the *single molecular* and *general atomic* models exhibit similar results in terms of convergence (same number of layers and similar training time). While it is a convenient and compact comparison, summing the residual for each chemical species does hide some interesting aspects of the convergence.

4.2 Species by species convergence

Unlike the most ML applications, a simple evaluation of the loss function (both for training and validation points) it is not enough to guarantee the goodness of the model for PINN algorithm. Indeed, the loss function gives a domain-wide average, thus does not imply local convergence; this aspect is particularly important since we adopt multi-components loss function thus – despite the precautions taken (Sec. 3.3) – some chemical species or a particular set of **IC** might be not well reconstructed.

Fig. 7 shows an example of the training of each of the components of the loss function in the *general molecular* case. Up to $\simeq 1.2 \times 10^6$ training epochs, all the different components follow a similar decreasing trend. After $\simeq 4 \times 10^6$ the model start to converge, and the different components saturates at different values; in particular, we can see that the temperature reconstruction dominates the loss function for this specific model. To assess the situation, it is convenient to look at the distribution of residuals when the training is completed.

In the upper panels of Fig. 8 we show the various components of the loss function of the for both the *atomic* and *molecular* single model cases. For the *single atomic* model, all individual residuals are smaller than about 10^{-2}, with the exception of a negligible contribution for the high values tail of the distributions. For the *single molecular* model, in general similar values are present, but the high values tails are in general larger, as expected from different values of the loss function when the training is stopped; in particular, the high values tail of temperature evolution surpasses 10^{-2}.

Recall that, for *general models*, **ICs** span $20 \leq T_{in}/\mathrm{K} \leq 10^5$ and $10^{-2} \leq n_{in}/\mathrm{cm}^{-3} \leq 10^3$, and the fraction of each species is selected to respect charge neutrality and the constraint $X \simeq 0.7$ and $Y \simeq 0.3$, i.e. they have larger number of dimension with respect to *single models*. Looking on the bottom-left panel of Fig. 8, the residual distributions for all species in the case of *general atomic* model are centered around (or below for negative particles) 10^{-2} with tails that does not exceed 10^{-1}. For *general molecular* model (bottom-right panel of Fig. 8), the residuals of shown a similar behavior except temperatures show similar errors except temperature distribution, centered at $\sim 10^{-0.5}$ (compare with Fig. 7). From Fig. 8 we can draw the following conclusions: (**i**) except for *single model* the temperature evolution residuals are the most difficult to minimize and (**ii**) the residual distribution are for different species are about the same order of magnitude, which implies a good balance of the loss function.

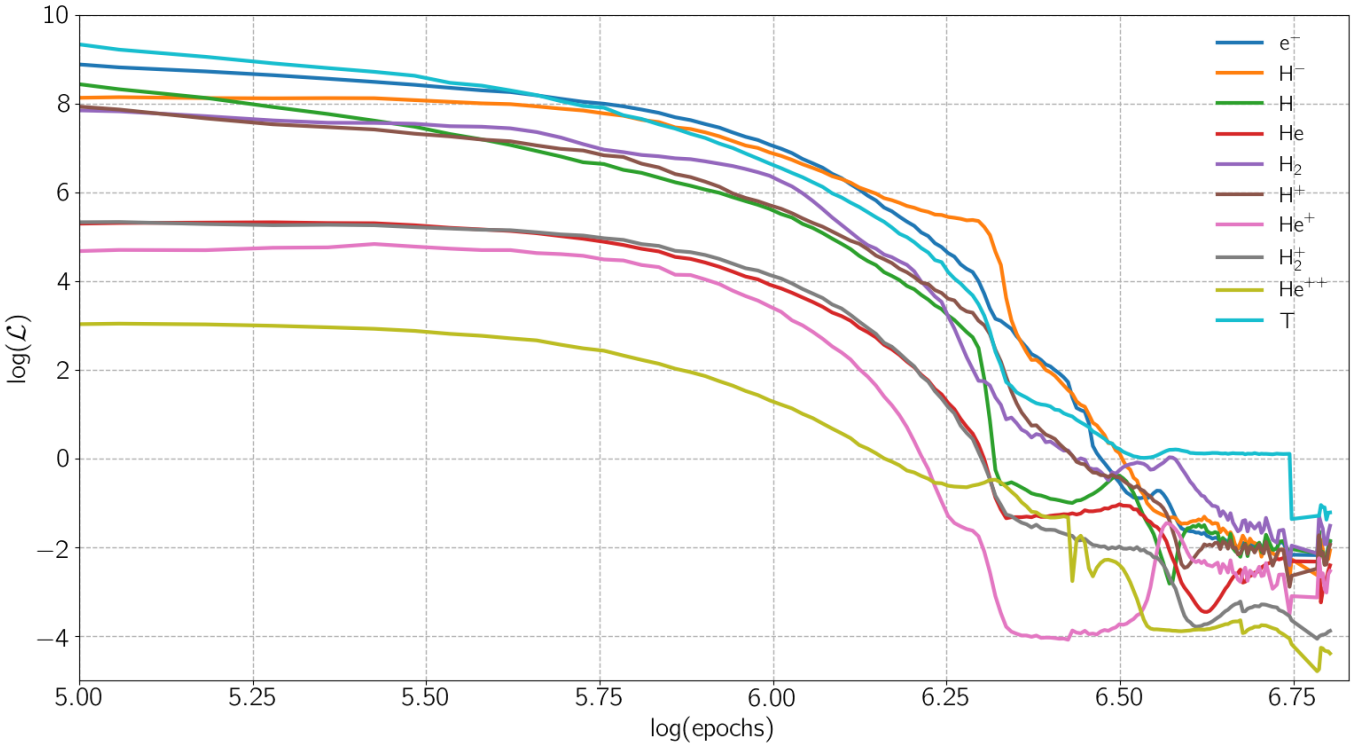

Figure 7. Species by species loss function during the training of the *general molecular* model. Notation is analogue to the one in upper panel of Fig. 6, however the sampling of the loss at different epoch is finer.

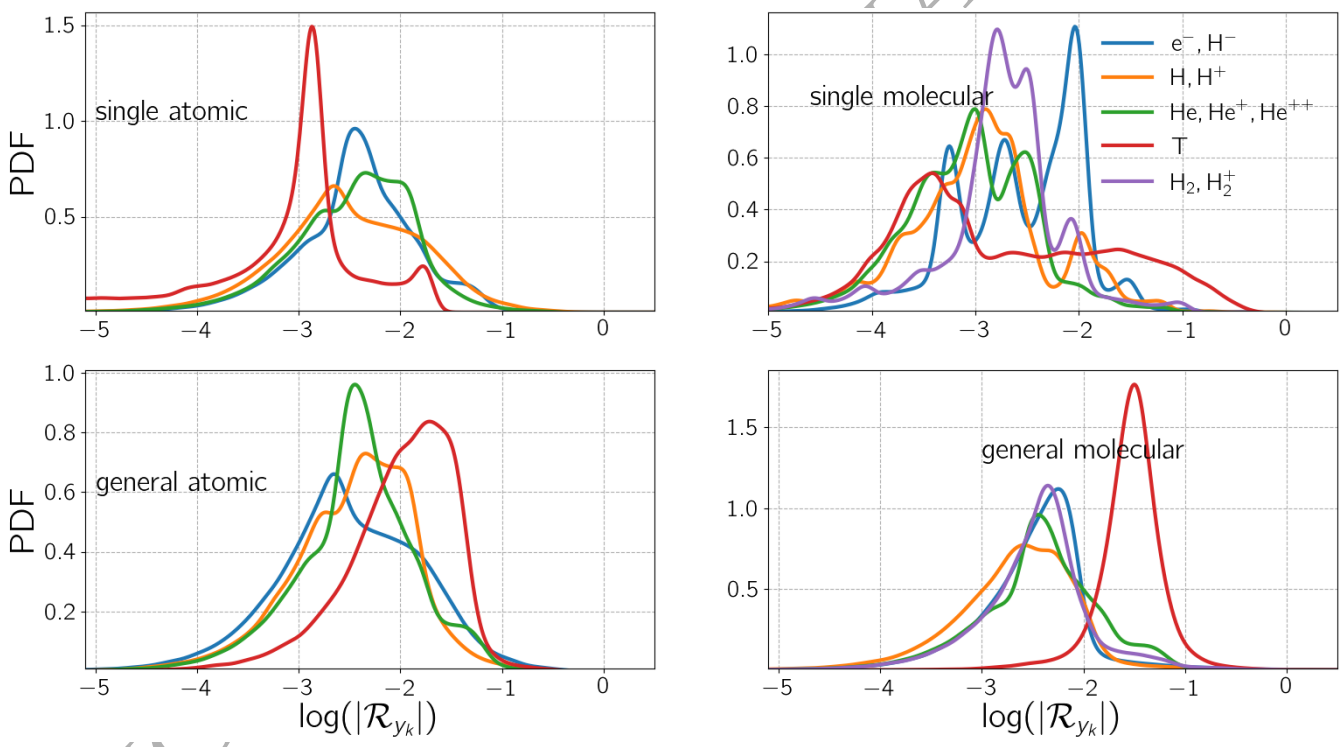

Figure 8. Species by species residuals ($\mathcal{R}_k$) after the training. In each panel we show the PDF for a different PINN model, as indicated in the inset. PDF of different contributions to $\mathcal{R}_k$ are grouped with different colours, according to the legend (top right panel). See Tab. 1 for references.

4.3 Validation of the single atomic network

To validate the training, we compare with the results from KROME (Grassi et al. 2014), that it is used to build the procedural solvers for both the *atomic* and *molecular* chemical networks (see Sec. 2 for a description of the code).

In Fig. 9 we show the evolution of the *single atomic* model for 1Myr. We note a qualitative good agreement with procedural solvers in terms of logarithm of the density except for He^{++} and H^-. We have obtained a very good reconstruction of the initial conditions (which are a soft constraint in the model) and in general the algorithm mixes the variations in density with precision that decreases in cases of species with sharp turns (in log-space).

The choice of adopting logarithmic time (eq. 21c) plays an important role here; while it has no immediate advantage in terms of the loss function (eq. 24), it helps in the recovery of abundance evolution which starts far from the equilibrium, which consequently have a very steep – and thus difficult to reproduce – evolution. In practical terms, the logarithmic time scale has a smoothing effect on these sharp turns and convergence is thus facilitated. However, despite the smoothing effect, the fastest varying species have the largest relative errors.

Note that in the case show in Fig. 9, a non-negligible benefit comes from refining the training with a L-BFGS optimizer (Liu & Nocedal 1989; Schraudolph et al. 2007). This change lowered the training curve only by a factor $\simeq 1/3$; this is expected: with respect to ADAM, L-BFGS is of higher order but is more prone to get stuck in local minima when the ODEs are stiff (Lu et al. 2019). Importantly, the adoption of the L-BFGS refinement has significantly improved the validation with KROME. In particular, before He^{++} and H^- showed errors two order of magnitude larger. Further, by using L-BFGS, the reconstruction of the species at early time and at equilibrium is improved.

To be more quantitative, it is convenient to define both relative (Δ_r) and fractional (Δ_f) differences:

$$\Delta_f = \frac{|n_{NN} - n_{KR}|}{n_{tot}} \tag{30a}$$

$$\Delta_r = \frac{|n_{NN} - n_{KR}|}{n_{KR}}, \tag{30b}$$

where n_{NN} are the values predicted with the trained models and n_{KR} are computed with KROME. In the bottom panel of Fig. 9 we show the distribution of relative and fraction errors. While for most of the species the relative error are reasonably small ($\Delta_r \lesssim 10^{-1}$), the relative errors are dominated by H^-, which peaks at $\Delta_r \sim 1$ and He^{++}, which has a very high values tail. However, these two species have low abundances, thus the fractional errors of all species are small, i.e. $\Delta_f \lesssim 10^{-2}$.

In terms of usage, it seems encouraging that the dominant relative errors affect the less abundant species, since they lead to negligible fractional errors; however, low abundance species can be important in some situations, i.e. H^- abundance is critical in computing H_2 in low metallicity/dust environments. For a proper usage, such errors should be removed with further training and/or by reconsidering the architecture via a change of hyperparameters setup.

4.4 Model comparison

Our other models are compared with KROME in Fig. 10, where we show the PDF of the relative error Δ_r. For convenience, a summary of the quantiles of the distributions is reported in Tab. 2.

Results for the *single molecular* model are shown in the left panel of Fig. 10. Overall we found a good reconstruction, with the 75% of the points with a relative error $\log(\Delta_r) < -0.84$. However the goodness of the reconstruction presents a species to species variance, in particular we found $\Delta_r \simeq 1$ in the most of cases for He^{++}, similarly to what we found in the *single atomic* model. As this feature is present for both chemical network, it might be a sign that our models fails to accurately capture the ionization of He^+, which has a very small rate given the hardness of the impinging radiation field.

Quantile	atomic single	molecular single	atomic general	molecular general
50%	-1.51	-2.91	-3.87	-1.5
75%	-1.05	-0.84	-0.89	-0.15
90%	-0.77	0.16	-0.15	0.38

Table 2. Quantiles of stacked distributions shown in Fig. 10. We utilize quantiles instead of mean and standard deviation because of the non-trivial shape of the errors distribution. The quantile is defined as in Hyndman & Fan (1996). The numbers are the logarithm of the relative error value that corresponds to the x% element of the sorted entire ensemble.

Focusing on *general atomic* model (central panel of Fig. 10), we note that overall the 75% of the predictions have relative errors smaller than $\log(\Delta_r) < -0.89$; this behaviour is similar to the *single molecular* model, which have been trained for about the same number of epochs. While *general atomic* has a wider range of **IC**, *single molecular* have more reactions; however, with respect to other techniques ML is less affected by the curse of dimensionality, thus it is not straightforward to predict a hierarchy of complexity between different models, i.e. larger reactions set vs larger IC parameter space.

In the right panel of Fig. 10 we show the relative errors distribution for the *general molecular* model. We note an overall greater difficulty to emulate the procedural solvers, with the 75% of prediction is smaller than $\log(\Delta_r) < 0.05$, with larger errors on temperature, negatively charged ions and hydrogen. For most of the species, errors for *general molecular* are higher with respect to previously seen models, as a consequence of the increase of complexity for both the number of reactions and parameter space for IC; however He^{++} presents very small errors, thus overall the PINN convergence rate of individual species seems to be not to be directly linked to the reaction rate of the chemical system.

While up to this point we have considered only 1D PDF, it is interesting to see if the emulation power of *general* models is different depending on the initial position in the $T - n$ plane (Figs. 11 and 12). Compared to *atomic* models we have a better reconstruction of the evolution of H and H^+ in the *molecular* models. Although we do not have a definitive explanation for this improvement, we are led to think that the inclusion of H_2 and H_2^+ has some black-box effect that allows the model to better reconstruct the evolution of hydrogen.

In Fig. 11 we show the Δ_r distribution of *general atomic* computed at $\sim$ 1Myr as a function of the temperature and gas density. In Fig. 11, we note a good reconstruction over the entire parameter space for negative ions, while for what concerns hydrogen (H and H^+) and helium (He, He^+ and He^{++}) we have a good agreement at high densities which gradually worsens at low densities. While this is not shown in the figure, the error at low density is dominated by positively charged ions for both hydrogen and helium. Regarding the evolution of the temperature, we generally have a very good reconstruction, except when the initial temperature is $T_{in} \simeq 6 \times 10^3$K.

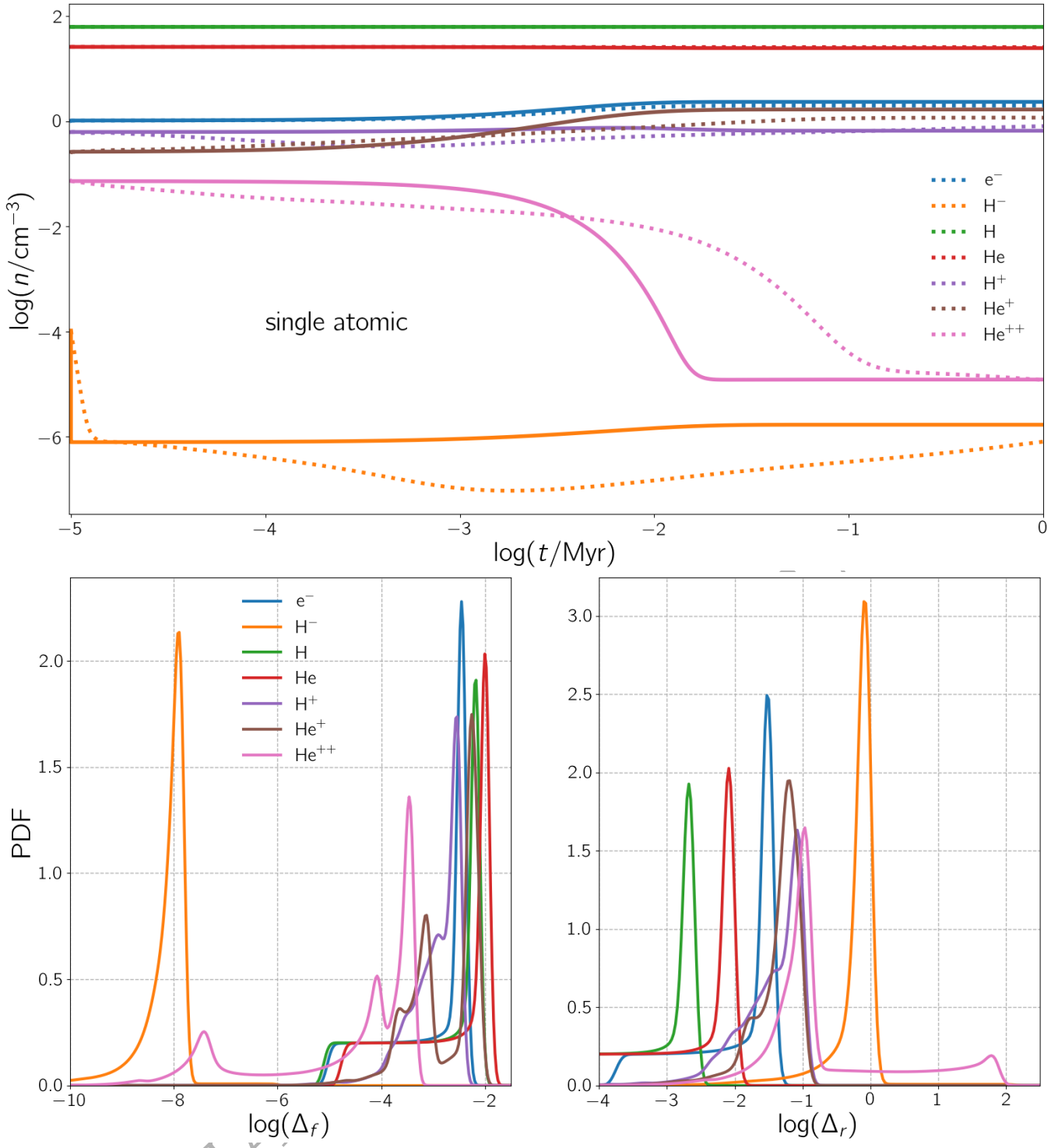

Figure 9. Model comparison between our PINN and KROME for the *singol atomic* model. **Upper panel**: number density (n_k) as a function of time (t) for each species. Solid lines represent the solution from KROME, while the dotted lines shown the PINN predictions. **Bottom panels**: PDF of the differences between the PINN and KROME. In the left (right) panel, we shows the absolute (relative) Δ_f (Δ_r) difference for each species. For the definitions, eq.s 30.

Similarly to *general atomic*, in Fig. 12 we show the relative errors to the *general molecular* model

Relative to *general atomic*, the errors for *general molecular* are larger (as also expected from Fig. 10), but have a very different behaviour in the *n-T* plane. Both the H, H^+ and He, He^+, He^{+}+ groups have flat distributions of errors, and show relatively a good agreement with KROME. Electrons have in general higher errors, with two *n-T* narrow stripes of lower errors, likely caused by a higher concentration of training points. The molecular hydrogen has high relative errors in the low density regime, where its density is negligible, and the temperature shows a tension that gradually rises with density. Overall,

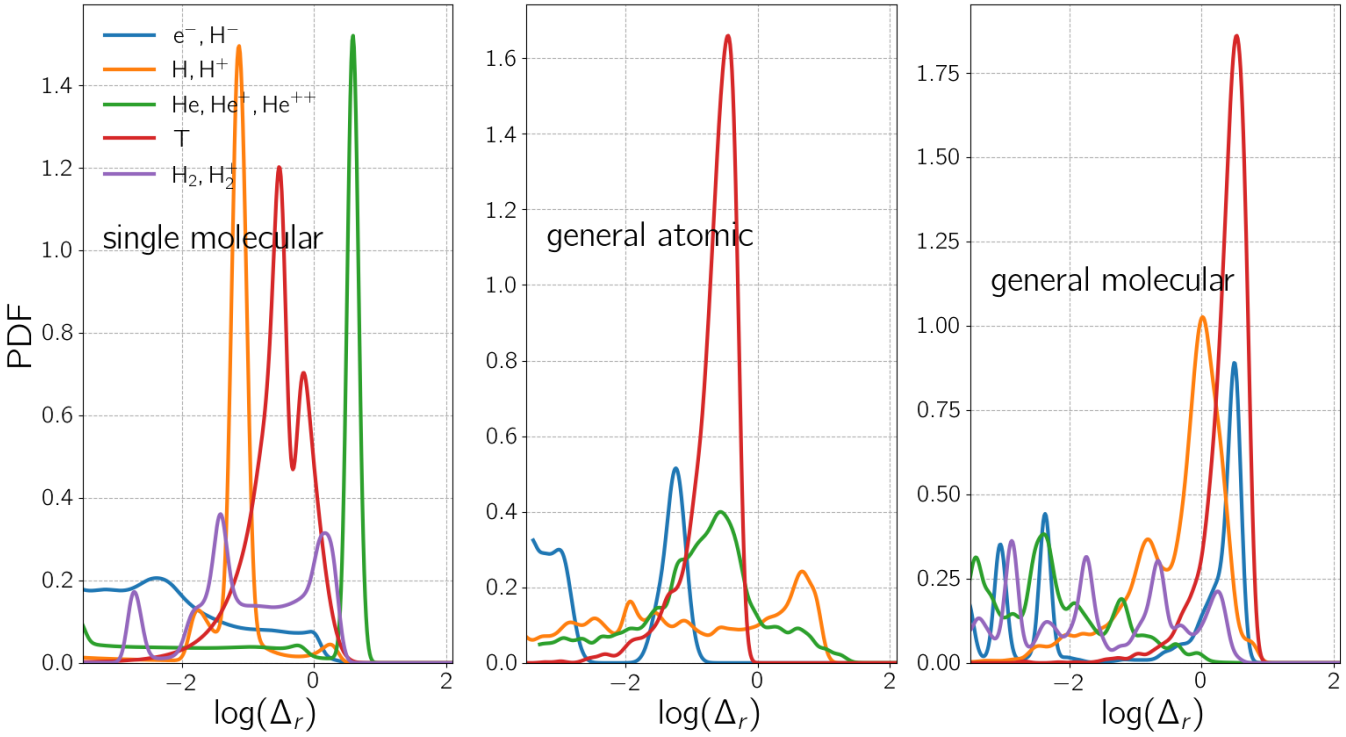

Figure 10. Relative errors distribution between KROME and the following trained PINN models: *single molecular* (left panel), *general atomic* (central panel) and *general molecular* (right panel). As done Fig. 9, different species grouped for clarity.

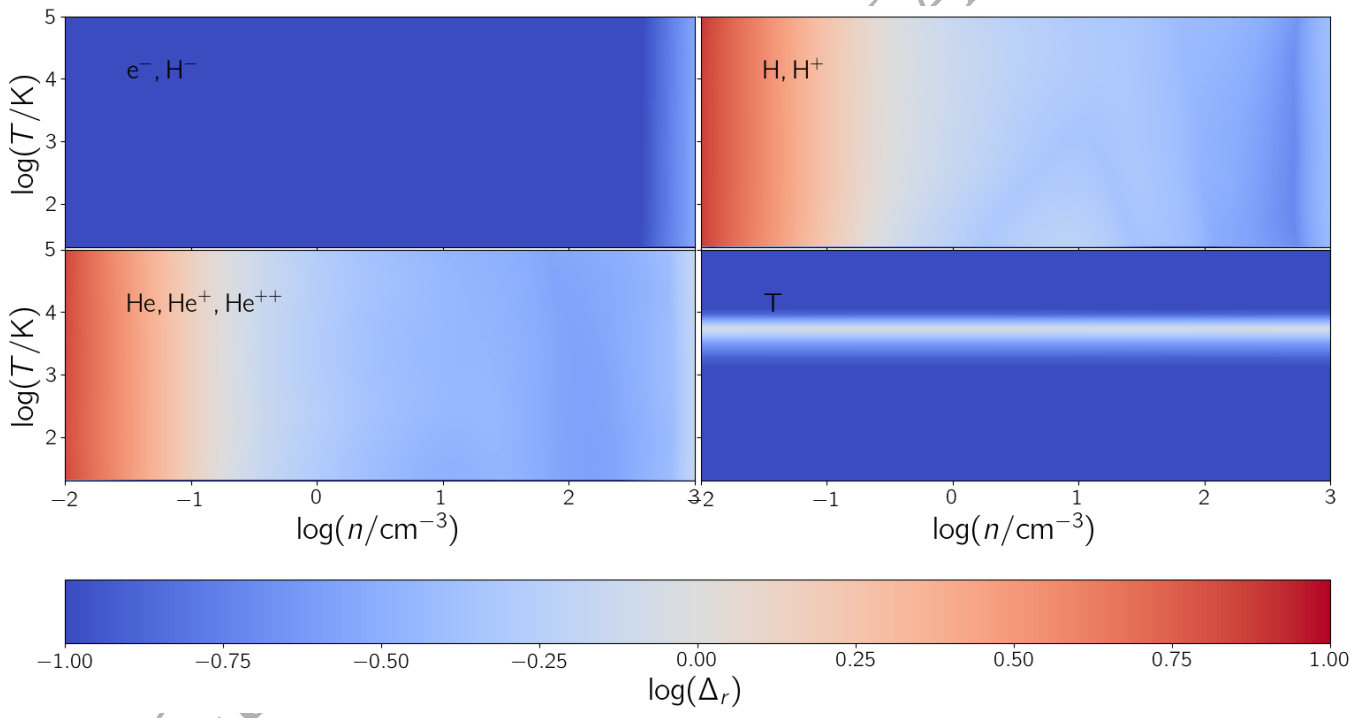

Figure 11. Relative errors for all species in temperature-density ($T - n$) plane for *general atomic* model. Results are grouped in different panels and computed after ∼ 1Myr of evolution from the initial condition. To help the visualization Δ_r has been cut at the lower (upper) min $\log \Delta_r = -1$ (max $\log \Delta_r = 1$) bound.

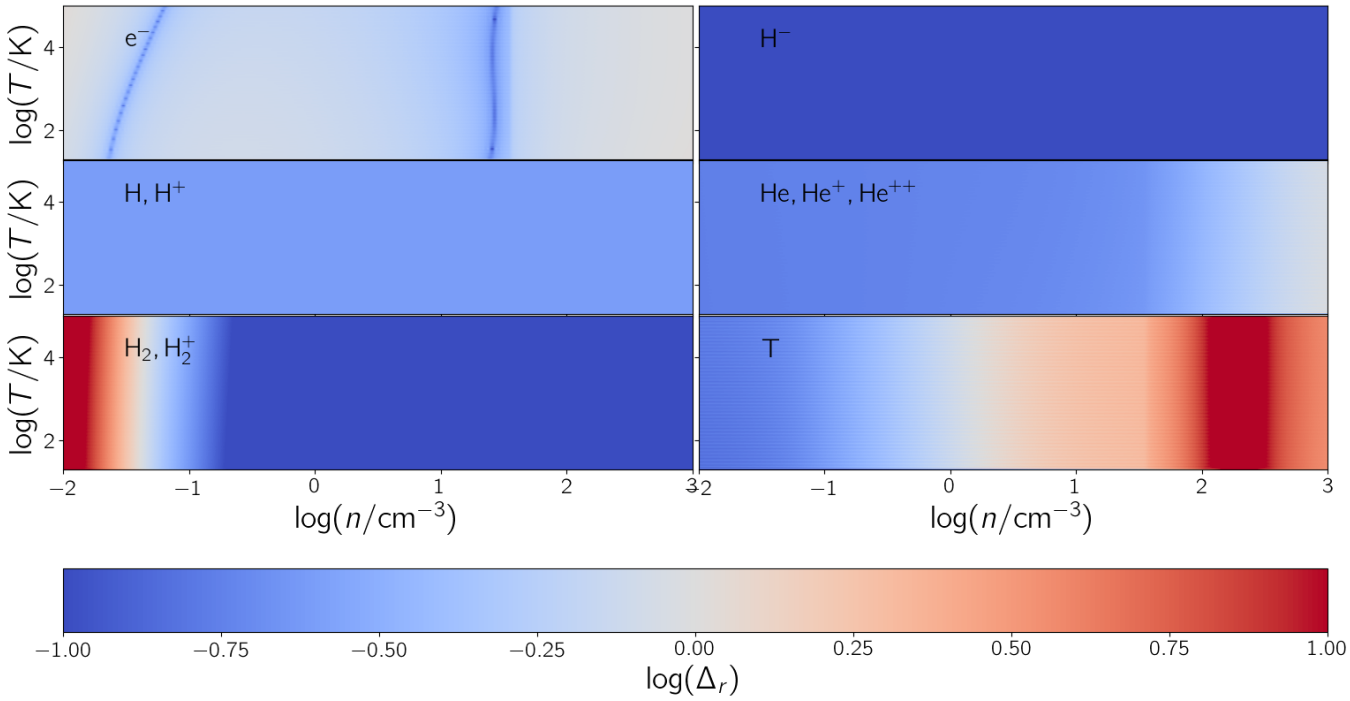

Figure 12. Relative errors for all species in temperature-density ($T-n$) plane for *general molecular* model. Results are grouped in different panels and computed after $\sim$ 1Myr of evolution from the initial condition. To help the visualization Δ_r has been cut at the lower (upper) min $\log \Delta_r = -1$ (max $\log \Delta_r = 1$) bound.

there seem to be no causal connection between regions with large errors and the underlying chemical system.

Summarizing, despite the inherent difficulty of the problem (especially in *general* cases) for a proof of concept work these are encouraging results, as regions where errors are high can be cured with a longer training. In view of a coupling with numerical simulations and to ease the convergence rate of the training, an exploration the hyperparameters space of the network architecture is needed, along with an adaptive addition of training points in the regions of parameter space where errors are higher.

It is interesting to conclude our analysis by comparing the timing of the PINN network compared to procedural solvers. For these tests both codes run on similar machines, specifically PINN runs using a single INTEL XEON Gold 6240 CPU with a frequency of 2.60GHz, the procedural solver KROME adopts a single INTEL i7-9700 CPU with a frequency 3.00GHz. For *single* models, we checked the evolution of system using 10^5 different end time t_{end} from 10yr to 1Myr. With KROME, we find that *single atomic* (*single molecular*) models have a completion time[7] of from 0.05s (0.09s) to 10.56s (11.16s) with increasing t_{end}. With the PINN, we obtain a speed up of a factor $\sim$ 207 ($\sim$ 116) for *single atomic* (*single molecular*). For the general models we prepare a 3D grid that is uniformly sampled in log-spaced: the grid features different initial species fractions, $C_k \in [10^{-6}, 1)$ (100 points), $T_{in} \in [20, 10^6]$K (512 points), and $n_{in} \in [10^{-2}, 10^3]$cm^{-3} (512 points); different initial conditions are evolved up to $t_{\rm end} = 1$Myr. With respect to the KROME solver, for the PINN we have a speed-up of a factor of $\sim$ 108 and $\sim$ 91 for the *general atomic* and *general molecular* respectively. The speed-up is better in cases of the simpler thermo-chemical system. However, with exploration of the hyperparameter space of the PINN, it should be possible to obtain lighter networks, i.e. needing fewer algebraic operations to compute the output, to achieve a further the speed-up for complex chemical networks.

It is important to note that varying the initial conditions, the *general atomic* (*general molecular*) KROME yields a model to model standard deviation of $\sim$ 24% ($\sim$ 30%) with respect to the mean completion time. For the PINN, the variance is negligible, i.e. less than 10^{-5}%. If exploited, this is a critical advantage over procedural solvers, since the latter are prone to yield load balancing problems, while the usage of the PINN can improve the scaling of parallel numerical codes with hydrodynamic coupled with chemistry.

Finally, we compare with other works have tried to adopt machine learning techniques to solve the chemical evolution, i.e. LATENT_ODE (Grassi et al. 2021) and CHEMULATOR (Holdship et al. 2021). With respect to the chemical network adopted here, these models emulate more complex chemical networks in a higher density range ($\log n/\mathrm{cm}^{-3} \gtrsim 1$), i.e. with 33 and 29 species for CHEMULATOR and LATENT_ODE, respectively, even tough LATENT_ODE have no temperature dependence nor evolution. Differently to these methods, our PINN is completely unsupervised, i.e. the PINN does not requires a pre-computed dataset for the solutions (CHEMULATOR) or a procedural solver in latent space (LATENT_ODE). Further, while both the present and LATENT_ODE are developed with hydrodynamic code coupling in mind, CHEMULATOR is tough to be a faster alternative to photoionization models (Röllig et al. 2007), i.e. its structure is too computationally demanding to be included in numerical simulations (Holdship et al. 2021).

Overall our PINN method has better validation errors with respect to LATENT_ODE and comparable to CHEMULATOR. For the speed-up, the PINN yields performances that are slightly above the one of LATENT_ODE ($\times$65), with both ML techniques being tested against procedural ODE solvers. The speed-up for CHEMULATOR is much

[7] Note that a non-negligible amount of time is spent in the warm-up of the solver.

better (×50000), but it is obtained against the photoionization code UCLCHEM (Holdship et al. 2017), which is a much more complex program with respect to an ODE solver. Final caveat, these comparisons should be taken with care, as all these models are still at a proof of concept stage, thus further optimizations are possible.

5 CONCLUSION

Chemical processes are key in regulating the evolution of the interstellar (ISM) and intergalactic medium. However, in cosmological and astrophysical simulations, finding solutions for thermo-chemistry networks is numerically costly: the systems are stiff and there are orders of magnitude of differences between the time scale of chemical reactions and the ones of astrophysical processes (e.g. gravity and fluid dynamics). This requires the usage of robust, high-order, multi step backward integrator for ordinary differential equations (ODE), potentially leading to computational bottlenecks for the numerical simulations.

In this work, we explored the possibility to use trained unsupervised physics informed neural networks (PINN) as an alternative to solve or ameliorate such problems. The main idea consists in expressing the underlying ODEs solution with a neural network (NN) and treat the problem in a variational way, i.e. by minimizing the residuals of the chemical system. This procedure (Sec. 2) has been first introduced in Raissi et al. (2019) but has never been adopted in the context of thermo-chemistry for astrophysical problems.

We first tested the method using benchmark cases (Sec. 2.3), then we developed the necessary technical solutions for our case study (Sec. 3.1). We adopted two different thermo-chemical networks that can solve the ISM chemistry with and without molecular hydrogen formation. We build PINN with fixed and arbitrary initial conditions in *single* and *general* models, respectively. Our main results can be summarized as follows.

- A simple feed-forward architecture cannot reconstruct the evolution of a realistic thermo-chemical network. The minimal setup to achieve good results consist in adopting: a Deep Galerkin Method as Neural Network architecture coupled with adaptive sigmoid activation function, adaptive weights in the loss function, an annealing learning rate, and ADAM optimization algorithm (possibly followed by L-BFGS, see Sec.s 3.3 and 3.4). Moreover, a considerable benefit consists in solving the equations in log-space, both for the abundance and time; while the former is a standard normalization technique, a logarithmic time helps for far-from equilibrium situations, as it increase the time sampling.
- Even when running on state-of-the-art GPUs, for all models the training time for convergence is relatively large when compared to typical PINN cases of study that are in the literature. We train the models for $\sim 3 \times 10^6$ epochs, with a total computational time that can vary from $\simeq 10^2$GPUhr to $\simeq 2 \times 10^3$GPUhr. Unless dedicated resources are allocated, these relative long training times make it expensive to scan the hyperparameter space of the network to improve the convergence and validation.
- As expected, the simplest realistic case (*single atomic*) costs much less respect to the most complex case (*general molecular*), both in terms of training time, number of training points and dimension of the network itself (and thus associated memory). However, the hierarchy of complexity between the other model is not clear. On the one hand, the *general atomic* model is multidimensional, thus i) the problem become more stiff as the reaction rates vary more wildly and ii) the curse of dimensions starts to play a role, even tough traditionally ML is less affected with respect to other techniques. On the other hand, in the case of *single molecular* model, there are about double the reactions and molecular heating and cooling processes. It is unclear which of these two factors is dominant in hindering a fast convergence.
- We find overall a good agreement with procedural solver. For almost all models, more than 75% of the points with relative errors less than 15% and 50% of the prediction smaller than 1%; for *single atomic* we have errors smaller than 17% in 90% of cases. In the case of *general* models we obtain better results in the high density region of the thermodynamic parameter space.
- For all models, we obtained a significant speed-up respect to the procedural solvers, from $\sim$ 200 for the more simple model to $\sim$ 90 for the more complex. Furthermore, the very low variance of computational time in different thermodynamic conditions can potentially solve load balancing problems that can occur in the context of massively parallel codes.

Although the chemical networks emulated in this work are relatively small (up to 9 ions and 46 reactions), we do not expect to have excessive problems in increasing the number of species, mainly because, with respect to other methods, the PINN suffer less from the curse of dimensionality (i.e. see Mishra & Molinaro (2020) for the PINN method in up to $\sim$ 100 dimensions). Thud, it would be interesting to test the PINN for possible application to wider chemical networks, i.e. explicit non equilibrium metals evolution following carbon and oxygen chemistry (Glover et al. 2010): it would entail tracing the evolution of 32 chemical species and 218 reactions; this would represent an interesting possibility for future developments, which should also give a fairer benchmark with respect to other attempts done in emulating chemistry with different ML techniques (Grassi et al. 2021).

Limited to our knowledge, the present work is the first case of PINN application for systems of ODEs with this level of complexity. Thus, this proof-of-concept work is a necessary step before using the trained models as emulators in simulations. In the range of applicability tested here, we can conclude that the models can be used as emulator without a significant loss of precision, as long as further refinement is included for those n-T regions where some of the species tracked by the network experience a loss of precision. For the future, the promising speed-up (up to $\sim$ 200) and the absence of variance in the completion time of the calculation make the PINN a very palatable tool for the solution of chemical networks in astrophysical simulations.

ACKNOWLEDGEMENTS

AP acknowledges support from the ERC Advanced Grant INTERSTELLAR H2020/740120. We gratefully acknowledge computational resources of the Center for High Performance Computing (CHPC) at SNS. We acknowledge the use of the Python programming language (Van Rossum & Drake 2009), Matplotlib (Hunter 2007), NumPy (van der Walt et al. 2011), and Scipy (Virtanen et al. 2019).

Data availability

The derived data generated in this research will be shared on reasonable requests to the corresponding author.

REFERENCES

Abadi M., et al., 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, https://www.tensorflow.org/

Asplund M., Grevesse N., Sauval A. J., Scott P., 2009, ARA&A, 47, 481

Bakes E. L. O., Tielens A. G. G. M., 1994, ApJ, 427, 822

Bovino S., Grassi T., Capelo P. R., Schleicher D. R. G., Banerjee R., 2016, Astronomy & Astrophysics, A15, 1

Byrne G. D., Hindmarsh A. C., 1987, Journal of Computational Physics, 70, 1

Cen R., 1992, ApJS, 78, 341

Chantada A. T., Landau S. J., Protopapas P., Scóccola C. G., Garraffo C., 2022, arXiv e-prints, p. arXiv:2205.02945

Chardin J., Uhlrich G., Aubert D., Deparis N., Gillet N., Ocvirk P., Lewis J., 2019, MNRAS, 490, 1055

Chen R. T. Q., Rubanova Y., Bettencourt J., Duvenaud D., 2018, arXiv e-prints, p. arXiv:1806.07366

Cybenko G., 1989, Mathematics of Control, Signals, and Systems (MCSS), 2, 303

De Ryck T., Mishra S., 2021, arXiv e-prints, p. arXiv:2106.14473

Decataldo D., Pallottini A., Ferrara A., Vallini L., Gallerani S., 2019, MNRAS, 487, 3377

Decataldo D., Lupi A., Ferrara A., Pallottini A., Fumagalli M., 2020, MNRAS, 497, 4718

Draine B. T., 1978, ApJS, 36, 595

Dropulic A., Ostdiek B., Chang L. J., Liu H., Cohen T., Lisanti M., 2021, ApJ, 915, L14

Flamant C., Protopapas P., Sondak D., 2020, arXiv e-prints, p. arXiv:2006.14372

Galli D., Palla F., 1998, A&A, 335, 403

Ge J., 2022, Research in Astronomy and Astrophysics, 22, 015004

Glover S. C. O., Abel T., 2008, MNRAS, 388, 1627

Glover S. C. O., Federrath C., Mac Low M. M., Klessen R. S., 2010, MNRAS, 404, 2

Goyal P., et al., 2018, Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour (arXiv:1706.02677)

Grassi T., Bovino S., Schleicher D. R. G., Prieto J., Seifried D., Simoncini E., Gianturco F. A., 2014, MNRAS, 439, 2386

Grassi T., Nauman F., Ramsey J. P., Bovino S., Picogna G., Ercolano B., 2021, Reducing the complexity of chemical networks via interpretable autoencoders (arXiv:2104.09516)

Gunes Baydin A., Pearlmutter B. A., Andreyevich Radul A., Siskind J. M., 2015, Automatic differentiation in machine learning: a survey (arXiv:1502.05767)

Haardt F., Madau P., 2012, ApJ, 746, 125

Haghighat E., Juanes R., 2021, Computer Methods in Applied Mechanics and Engineering, 373, 113552

Halton J. H., Smith G. B., 1964, Commun. ACM, 7, 701

Hennigh O., et al., 2020, NVIDIA SimNet{TM}: an AI-accelerated multi-physics simulation framework (arXiv:2012.07938)

Hindmarsh A. C., 2019, ODEPACK: Ordinary differential equation solver library (ascl:1905.021)

Hirashita H., Ferrara A., 2002, MNRAS, 337, 921

Holdship J., Viti S., Jiménez-Serra I., Makrymallis A., Priestley F., 2017, AJ, 154, 38

Holdship J., Viti S., Haworth T. J., Ilee J. D., 2021, A&A, 653, A76

Hornik K., Stinchcombe M., White H., 1989, Neural Networks, 2, 359

Hu Z., Jagtap A. D., Karniadakis G. E., Kawaguchi K., 2022, SIAM Journal on Scientific Computing, 44, A3158

Hunter J. D., 2007, Computing in Science Engineering, 9, 90

Hyndman R., Fan Y., 1996, The American Statistician, 50, 361

Jagtap A. D., Kawaguchi K., Karniadakis G. E., 2020, Journal of Computational Physics, 404, 109136

Ji W., Qiu W., Shi Z., Pan S., Deng S., 2021, The Journal of Physical Chemistry A, 125, 8098

Jiang C. M., et al., 2020, arXiv e-prints, p. arXiv:2005.01463

Jura M., 1975, ApJ, 197, 575

Kim J.-G., Kim W.-T., Ostriker E. C., 2018, ApJ, 859, 68

Kingma D. P., Ba J., 2014, Adam: A Method for Stochastic Optimization (arXiv:1412.6980)

Kumar A., Fisher R. T., 2013, MNRAS, 431, 455

LeCun Y., Bengio Y., Hinton G., 2015, Nature, 521, 436

Liu D. C., Nocedal J., 1989, Mathematical Programming, 45, 503

Lu L., Meng X., Mao Z., Karniadakis G. E., 2019, DeepXDE: A deep learning library for solving differential equations (arXiv:1907.04502)

Lupi A., 2019, MNRAS, 484, 1687

Maio U., Dolag K., Ciardi B., Tornatore L., 2007, MNRAS, 379, 963

Mishra S., Molinaro R., 2020, arXiv e-prints, p. arXiv:2007.01138

Mishra S., Molinaro R., 2021, J. Quant. Spectrosc. Radiative Transfer, 270, 107705

Moseley B., Markham A., Nissen-Meyer T., 2020, arXiv e-prints, p. arXiv:2006.11894

Nakamura Y., Shiratori S., Nagano H., Shimano K., 2021. , doi:10.11159/htff21.113

Nejad L. A. M., 2005, Ap&SS, 299, 1

Pallottini A., Ferrara A., Bovino S., Vallini L., Gallerani S., Maiolino R., Salvadori S., 2017, MNRAS, 471, 4128

Pallottini A., et al., 2019, MNRAS, 487, 1689

Pallottini A., et al., 2022, MNRAS, 513, 5621

Prelogović D., Mesinger A., Murray S., Fiameni G., Gillet N., 2022, MNRAS, 509, 3852

Rackauckas C., Innes M., Ma Y., Bettencourt J., White L., Dixit V., 2019, DiffEqFlux.jl - A Julia Library for Neural Differential Equations (arXiv:1902.02376)

Raissi M., Perdikaris P., Karniadakis G. E., 2019, Journal of Computational Physics, 378, 686

Robbins H., Monro S., 1951, Annals of Mathematical Statistics, 22, 400

Röllig M., et al., 2007, A&A, 467, 187

Schraudolph N. N., Yu J., Günter S., 2007, in Meila M., Shen X., eds, Proceedings of Machine Learning Research Vol. 2, Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics. PMLR, San Juan, Puerto Rico, pp 436–443, https://proceedings.mlr.press/v2/schraudolph07a.html

Semenov D., et al., 2010, A&A, 522, A42

Shen S., Madau P., Guedes J., Mayer L., Prochaska J. X., Wadsley J., 2013, ApJ, 765, 89

Sirignano J., Spiliopoulos K., 2018, Journal of Computational Physics, 375, 1339

Sitzmann V., Martel J. N. P., Bergman A. W., Lindell D. B., Wetzstein G., 2020, arXiv e-prints, p. arXiv:2006.09661

Smith B. D., et al., 2017, MNRAS, 466, 2217

Srivastava R. K., Greff K., Schmidhuber J., 2015, arXiv e-prints, p. arXiv:1505.00387

Tancik M., et al., 2020, arXiv e-prints, p. arXiv:2006.10739

Theuns T., Leonard A., Efstathiou G., Pearce F. R., Thomas P. A., 1998, MNRAS, 301, 478

Ucci G., Ferrara A., Pallottini A., Gallerani S., 2018, MNRAS, 477, 1484

Van Rossum G., Drake F. L., 2009, Python 3 Reference Manual. CreateSpace, Scotts Valley, CA, https://dl.acm.org/doi/book/10.5555/1593511

Virtanen P., et al., 2019, arXiv e-prints, p. arXiv:1907.10121

Wang H., Raj B., 2017, arXiv e-prints, p. arXiv:1702.07800

Wang S., Teng Y., Perdikaris P., 2021, SIAM Journal on Scientific Computing, 43, A3055

van der Walt S., Colbert S. C., Varoquaux G., 2011, Computing in Science Engineering, 13, 22