Discovering Artificial Viscosity Models for Discontinuous Galerkin approximation of Conservation Laws using Physics-Informed Machine Learning

Matteo Caldana[a,*], Paola F. Antonietti[a], Luca Dede'[a]

[a] *MOX, Dipartimento di Matematica, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy*

October 8, 2024

Abstract

Finite element-based high-order solvers of conservation laws offer large accuracy but face challenges near discontinuities due to the Gibbs phenomenon. Artificial viscosity is a popular and effective solution to this problem based on physical insight. In this work, we present a physics-informed machine learning algorithm to automate the discovery of artificial viscosity models. We refer to the proposed approach as an "*hybrid*" approach which stands at the edge between supervised and unsupervised learning. More precisely, the proposed "*hybrid*" paradigm is not supervised in the classical sense as it does not utilize labeled data in the traditional way but relies on the intrinsic properties of the reference solution. The algorithm is inspired by reinforcement learning and trains a neural network acting cell-by-cell (the viscosity model) by minimizing a loss defined as the difference with respect to a reference solution thanks to automatic differentiation. This enables a dataset-free training procedure. We prove that the algorithm is effective by integrating it into a state-of-the-art Runge-Kutta discontinuous Galerkin solver. We showcase several numerical tests on scalar and vectorial problems, such as Burgers' and Euler's equations in one and two dimensions. Results demonstrate that the proposed approach trains a model that is able to outperform classical viscosity models. Moreover, we show that the learnt artificial viscosity model is able to generalize across different problems and parameters.

Key words: Artificial viscosity, Conservation laws, Discontinuous Galerkin, Physics-informed machine learning, Neural networks, Reinforcement learning.
MSC subject classification: 35L65, 65M60, 68T01

1 Introduction

Hyperbolic conservation laws, such as Euler equations or the equations of magnetohydrodynamics, have a wide range of applications in Engineering, ranging from aerodynamics, weather predictions, and civil engineering to astrophysics and nuclear fusion. The Discontinuous Galerkin (DG) method [11, 12, 15, 17, 31] is a well-established (and continuously growing in popularity) high-order solver that offers conservation of physical properties, high-order accuracy, geometric flexibility [2], and parallel efficiency [33, 38]. A known drawback of any high-order methods is that in the presence of discontinuities of the solution, the Gibbs phenomenon may appear [23], which causes spurious numerical oscillations that invalidate the physical meaning of the solution. Indeed, Godunov's theorem [60] states that any linear monotone numerical scheme can be at most first-order accurate. This problem is especially relevant since in certain regions of the domain continuous initial conditions may lead to a loss of regularity of the solution and thus numerical instability appears.

Several approaches have been developed in the literature to correct and reduce the effect of the Gibbs phenomenon for DG methods. A common feature of most of these approaches is that they rely on the identification of troubled cells, that is mesh elements where the solution loses regularity. For instance, slope limiters [9, 13, 37] are widely used, but they can significantly compromise the global solution accuracy, and an improper parameter selection may lead to increased computational costs. Other authors have investigated Weighted Essentially Non-Oscillatory (WENO) reconstruction [49, 50, 55], which maintains high-order accuracy but may exhibit a large computational cost. More recently, the Multidimensional Optimal Order Detection (MOOD) approach [18] and filtering methods [45] have been proposed. These approaches switch

*Corresponding author: matteo.caldana@polimi.it

the employed method on the troubled cells: from a high-order DG method to a monotone first-order method. However, their performance relies on the quality of the indicator and parameter tuning, respectively.

Instead, in this paper, we focus on a different approach called Artificial Viscosity (AV), which is based on the addition of a suitable amount of dissipation near discontinuities so to recover a smooth solution and thus high-order accuracy. A common characteristic of artificial viscosity approaches is the definition of a metric to evaluate the regularity of the solution. For instance, they may use differential operators that have a particular physical meaning [43], the jump of the flux on the element boundaries [7], or the residual in the mesh element [27]. Other models instead rely on estimating the average decay rate of modal coefficients [34], or just targeting the highest modes [48]. A very successful model is the so-called entropy viscosity (EV) model, which employs as a measure of regularity the residual of an entropy pair [24, 64]. A significant drawback of all these models is that their accuracy, robustness, and stability strongly depend on parameters tuned by physical insight and intuition. Indeed, the lack of an established rule for estimating optimal values and the fact that parameters must be picked on a problem-dependent basis makes their usage expensive.

In recent years, a large effort has been poured into overcoming these problems by employing supervised learning methods and more specifically, deep learning. In particular, neural networks (NN) have been used to create a universal artificial viscosity model that does not require to be tuned on a problem-dependent basis. Indeed, in [16, 54] the authors prove that is possible to train a NN that identifies regions where dissipation of numerical oscillations is needed. In [59] the authors use a NN to determine the optimal parameter for the SUPG stabilization method. In general, NNs have been proven to be a valuable tool to surrogate models and automate the expensive phase of trial-and-error present in many numerical applications [10, 20, 21, 28, 51].

Supervised learning however has some limitations. On one hand, building an extensive and high-quality dataset demands a considerable investment of time and computational resources. Indeed, in this case, its construction entails an expensive phase of parameter tuning of classical viscosity model in order to create examples that the NN could mimic. Moreover, supervised learning is not able to generalize beyond what is present in the dataset: the NN may generalize to problems not present in the dataset, but it is not possible to surpass the accuracy of the best among the classical viscosity models, which do not guarantee optimality. Namely, a critical issue of this kind has been exposed in a-posteriori testing of turbulence models trained by supervised learning [61]. A possible solution proposed in [44] is to employ reinforcement learning (RL) [58]. RL is a well-established but expensive paradigm where an agent learns to make decisions by interacting with an environment, receiving feedback in the form of rewards, and adjusting its behavior to maximize cumulative reward over time.

Our goal in this paper is to propose and test an algorithm to train a NN-based artificial viscosity model in a *hybrid* learning framework, which stands at the edge between supervised and unsupervised learning. To this end, we formulate the problem as an RL task and we enhance it with physics-informed machine learning (PIML) [26, 32]. Namely, thanks to automatic differentiation, we can differentiate through the RL environment, effectively embedding information on the physics of the problem into the learning dynamics. The resulting approach shares some similarities with model-based RL such as [25], differentiable RL [42], and optimal control. Therefore, our approach overcomes the shortcomings of supervised learning and is cheaper than RL since we can exploit the differentiability of the environment (the DG solver). The reward is defined as the dissimilarity from a reference solution with the addition of suitable regularization terms. This also enables seamless integration of noisy data obtained from measurements of quantities of interest into the loss function, thereby being able to learn new physical models directly from data. The NN that we employ is a generalization of the one presented in [16] aimed at increasing flexibility and accuracy. An ablation study proves the efficacy of the proposed modifications.

The paper is organized as follows. Section 2 introduces the mathematical model of conservation laws and its semi-discrete and algebraic formulations. In Section 3, we define the classical artificial viscosity models and we introduce the NN viscosity model. In Section 4 we introduce the novel training algorithm we are proposing and discuss the advantages of physics-informed machine learning. In Section 5 we show convergence results for a smooth problem and we present several applications to problems with discontinuous solutions. Finally, in Section 6, we draw some conclusions and discuss further developments.

2 The mathematical model

In this section, we state the formulation of conservation law under consideration. The problem is defined in a bounded open domain in space $\boldsymbol{x} \in \Omega \subset \mathbb{R}^d$ $(d = 1, 2)$ and for a time $t \in (0, T]$, with $T > 0$ given. The unknown of our problem is the vector of m conserved variables $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}, t) : \Omega \times (0, T] \to \mathbb{R}^m$. The considered convention-diffusion problem reads as follows:

$$\begin{cases} \partial_t \boldsymbol{u} + \nabla \cdot (-\mu(\boldsymbol{u}) \nabla \boldsymbol{u} + \boldsymbol{f}(\boldsymbol{u})) = 0 & \text{in } \Omega \times (0, T], \\ \boldsymbol{u}(\boldsymbol{x}, 0) = \boldsymbol{u}_0(\boldsymbol{x}) & \text{in } \Omega \times \{0\} \end{cases} \tag{1}$$

endowed with suitable boundary conditions on $\partial\Omega \times (0,T]$. Namely, we either employ periodic boundary conditions or mixed Dirichlet-Neumann boundary conditions

$$\begin{cases} \boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{g}_D(\boldsymbol{x}) & \text{on } \Gamma_D \times (0,T], \\ \boldsymbol{f}(\boldsymbol{u}(\boldsymbol{x},t))\boldsymbol{n} = \boldsymbol{g}_N(\boldsymbol{x}) & \text{on } \Gamma_N \times (0,T], \end{cases}$$

where Γ_N and $\Gamma_D \neq \emptyset$ are disjoint sets $\mathring{\Gamma}_N \cap \mathring{\Gamma}_D = \emptyset$ such that $\partial\Omega = \overline{\Gamma}_N \cup \overline{\Gamma}_D$ and $\boldsymbol{g}_D : \Gamma_D \to \mathbb{R}^m, \boldsymbol{g}_N : \Gamma_N \to \mathbb{R}^m$ are a regular enough boundary datum. In Eq. (1), $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^{m\times d}$ is the advection flux and $\boldsymbol{u}_0 : \Omega \to \mathbb{R}^m$ is a given initial condition. The viscous flux is controlled by the viscosity coefficient $\mu : \mathbb{R}^m \to \mathbb{R}$ that is a positive and bounded function that has the physical meaning of an artificial diffusion term.

2.1 The semi-discrete discontinuous Galerkin formulation

For the spatial discretization, we employ the discontinuous Galerkin method. Let $\mathcal{K}_h$ be a shape-regular family of open, disjoint cells (intervals for $d = 1$ or triangles for $d = 2$) K that approximates Ω. Moreover, let $\mathcal{E}_h$ be the set of all faces (points for $d = 1$ or edges for $d = 2$). On each face $F \in \mathcal{E}_h$, approximating the boundary $\partial\Omega$, a single type of boundary condition is imposed at a time (e.g. only Dirichlet or Neumann). We define $h = \max_{K\in\mathcal{K}_h} h_K$, where h_K denotes the diameter of K. We define the averages on and the jumps across the edges for a smooth enough scalar function $v : \Omega \to \mathbb{R}$, for a smooth enough vector-valued function $\boldsymbol{q} : \Omega \to \mathbb{R}^n$ and a smooth enough tensor-valued function $\boldsymbol{\psi}$. Let us consider two neighboring elements K^+ and K^-, let $\boldsymbol{n}^+$ and $\boldsymbol{n}^-$ be their outward normal unit vector and let $v^\pm, \boldsymbol{q}^\pm, \boldsymbol{\psi}^\pm$ be the restrictions to $K^\pm$, respectively. In accordance with [3], defined the symmetric outer product $\boldsymbol{q} \odot \boldsymbol{n} = \frac{1}{2}(\boldsymbol{v} \otimes \boldsymbol{n} + \boldsymbol{n} \otimes \boldsymbol{v})$, we define

- on the set of interior edges (denoted as $\mathring{\mathcal{E}}_h$):

$$\begin{aligned} \{\!\{v\}\!\} &= \frac{1}{2}(v^- + v^+), \qquad [\![v]\!] = v^+\boldsymbol{n}^+ + v^-\boldsymbol{n}^-, \\ \{\!\{\boldsymbol{q}\}\!\} &= \frac{1}{2}(\boldsymbol{q}^- + \boldsymbol{q}^+), \qquad [\![\boldsymbol{q}]\!] = \boldsymbol{q}^+ \odot \boldsymbol{n}^+ + \boldsymbol{q}^- \odot \boldsymbol{n}^-, \\ \{\!\{\boldsymbol{\psi}\}\!\} &= \frac{1}{2}(\boldsymbol{\psi}^- + \boldsymbol{\psi}^+), \qquad [\![\boldsymbol{\psi}]\!] = \boldsymbol{\psi}^-\boldsymbol{n}^- + \boldsymbol{\psi}^+\boldsymbol{n}^+. \end{aligned}$$

- on boundary edges:

$$\begin{aligned} \{\!\{v\}\!\} = v, \quad \{\!\{\boldsymbol{q}\}\!\} = \boldsymbol{q}, \quad \{\!\{\boldsymbol{\psi}\}\!\} = \boldsymbol{\psi}, \\ [\![v]\!] = v\boldsymbol{n}, \quad [\![\boldsymbol{q}]\!] = \boldsymbol{q} \odot \boldsymbol{n}, \quad [\![\boldsymbol{\psi}]\!] = \boldsymbol{\psi}\boldsymbol{n}. \end{aligned}$$

To further stabilize the scheme, as shown in [8, 31], we introduce the following weighted average, which introduces a stabilizing numerical flux

$$\{\!\{\boldsymbol{q}\}\!\}_\alpha = \{\!\{\boldsymbol{f}(\boldsymbol{q})\}\!\} + \frac{\alpha}{2}[\![\boldsymbol{q}]\!] \quad \forall\, F \in \mathcal{E}_h, \tag{2}$$

where $\alpha > 0$ is a scalar. For some choices of α we can recover already well-known numerical fluxes: if $\boldsymbol{f}$ is the identity and $\alpha = 1$ we have the upwind flux [4], if α is the largest eigenvalue (in absolute value) λ of the Jacobian $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}$ we have the Rusanov flux [52]. To construct the semi-discrete formulation, we define for any integer $k \geq 1$ the finite element space of discontinuous piecewise polynomial functions as

$$X_k^h = \{v \in L^2(\Omega) : v|_K \in \mathbb{P}^k(K) \ \ \forall\, K \in \mathcal{K}_h\}, \quad \boldsymbol{V}_k^h = [X_k^h]^d,$$

where $\mathbb{P}^k(K)$ is the space of polynomials of total degree less than or equal to k on K. Hence, the semi-discrete formulation reads as follows: For any $t \in (0,T]$, find $\boldsymbol{u}_h(t) \in \boldsymbol{V}_k^h$ such that:

$$\begin{cases} (\dot{\boldsymbol{u}}_h(t), \boldsymbol{v}_h)_\Omega + \mathscr{A}_h(\boldsymbol{u}_h(t), \boldsymbol{v}_h) - \mathscr{B}_h(\boldsymbol{u}_h(t), \boldsymbol{v}_h) + \mathscr{I}_h(\boldsymbol{u}_h(t), \boldsymbol{v}_h) = 0 & \forall\, \boldsymbol{v}_h \in \boldsymbol{V}_k^h, \\ \boldsymbol{u}_h(0) = \boldsymbol{u}_{0h} & \text{in } \Omega \times \{0\}, \end{cases} \tag{3}$$

where

- $\mathscr{A}_h : \boldsymbol{V}_k^h \times \boldsymbol{V}_k^h \to \mathbb{R}$ is defined by

$$\begin{aligned}\mathscr{A}_h(\boldsymbol{u}_h, \boldsymbol{v}_h) = &\sum_{K\in\mathcal{K}_h} \int_\Omega \mu(\boldsymbol{u}_h)\nabla\boldsymbol{u}_h{:}\nabla\boldsymbol{v}_h \,\mathrm{d}\Omega \\ &+ \sum_{F\notin\Gamma_N} \frac{\tau k^2}{|F|} \int_F \mu(\boldsymbol{u}_h)[\![\boldsymbol{u}_h]\!]{:}[\![\boldsymbol{v}_h]\!] \,\mathrm{d}\Sigma \\ &- \sum_{F\in\mathring{\mathcal{E}}_h} \int_F \{\!\{\nabla\boldsymbol{u}_h\}\!\}{:}[\![\boldsymbol{v}_h]\!] + [\![\boldsymbol{u}_h]\!]{:}\{\!\{\nabla\boldsymbol{v}_h\}\!\} \,\mathrm{d}\Sigma \quad \forall\, \boldsymbol{u}_h, \boldsymbol{v}_h \in \boldsymbol{V}_k^h,\end{aligned}$$

  where $\tau > 0$ is a suitable penalization constant and $|F|$ is the measure in the topology of the boundary.

- $\mathscr{B}_h : \boldsymbol{V}_k^h \times \boldsymbol{V}_k^h \to \mathbb{R}$ is defined as

$$\begin{aligned}\mathscr{B}_h(\boldsymbol{u}_h, \boldsymbol{v}_h) = &\sum_{K\in\mathcal{K}_h} \int_K \boldsymbol{f}(\boldsymbol{u}_h){:}\nabla\boldsymbol{v}_h \,\mathrm{d}\Omega \\ &- \sum_{F\in\mathring{\mathcal{E}}_h} \int_F \{\!\{\boldsymbol{u}_h\}\!\}_\lambda{:}[\![\boldsymbol{v}_h]\!] \,\mathrm{d}\Sigma \quad \forall\, \boldsymbol{u}_h, \boldsymbol{v}_h \in \boldsymbol{V}_k^h,\end{aligned}$$

  where $\{\!\{\boldsymbol{u}_h\}\!\}_\lambda$ indicates the Rusanov flux, as defined in Eq. (2).

- $\mathscr{I}_h : \boldsymbol{V}_k^h \times \boldsymbol{V}_k^h \to \mathbb{R}$ is a form that includes suitable terms for the boundary conditions.

2.2 Algebraic formulation

Denote as $(\boldsymbol{\varphi}_i)_{i=1}^I$ a suitable basis for the discrete space $\boldsymbol{V}_k^h$, where $I = \dim(\boldsymbol{V}_k^h)$, then

$$\boldsymbol{u}_h(t) = \sum_{i=1}^{I} U_i(t)\boldsymbol{\varphi}_i.$$

We collect the coefficients $U_i(t)$ of the expansions of $\boldsymbol{u}_h(t)$ in the vector function $\boldsymbol{U} : (0,T] \to \mathbb{R}^{mI}$. By defining the mass matrix

$$[M]_{ij} = (\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_j)_\Omega, \quad \forall\, i,j = 1,...,I$$

and the non-linear terms

$$[A(\boldsymbol{U}(t))]_i = \mathscr{A}_h(\boldsymbol{u}_h, \boldsymbol{\varphi}_i), \quad [B(\boldsymbol{U}(t))]_i = \mathscr{B}_h(\boldsymbol{u}_h, \boldsymbol{\varphi}_i), \quad [I(\boldsymbol{U}(t))]_i = \mathscr{I}_h(\boldsymbol{u}_h, \boldsymbol{\varphi}_i), \quad \forall\, i = 1,...,I.$$

We rewrite problem (3) in algebraic form as follows

$$\begin{cases} M\dot{\boldsymbol{U}}(t) + A(\boldsymbol{U}(t)) - B(\boldsymbol{U}(t)) + I(\boldsymbol{U}(t)) = 0 \qquad t \in (0,T], \\ \boldsymbol{U}(0) = \boldsymbol{U}_0, \end{cases} \tag{4}$$

where $\boldsymbol{U}_0$ is the projection of the initial datum $\boldsymbol{u}_0$ on $\boldsymbol{V}_k^h$. For the space discretization, we employ a nodal spectral basis derived from a multidimensional generalization of the Legendre-Gauss-Lobatto points as shown in [31].

2.3 Fully-discrete formulation

System (4) can be solved with various time-marching strategies, like Runge-Kutta schemes, by defining a partition of N intervals $0 = t^0 < t^1 < ... < t^N = T$. In literature (see [31]), this is usually coupled with an adaptive time-step $\Delta t = \Delta t(t)$ determined by the following CFL condition at a time t:

$$\Delta t = \frac{\mathrm{CFL}}{\dfrac{k^2}{h}\max\limits_{\boldsymbol{x}\in\Omega} \left|\boldsymbol{f}'(\boldsymbol{u}_h(\boldsymbol{x},t))\right| + \dfrac{k^4}{h^2}\max\limits_{\boldsymbol{x}\in\Omega} \mu(\boldsymbol{u}_h(\boldsymbol{x},t))}. \tag{5}$$

Specifically, we utilize one of the two following options: a third-order Strong Stability Preserving Runge-Kutta (SSPR3) scheme [19] or a fourth-order low-storage explicit Runge-Kutta with five stages [12]. The choice between the two is made so to minimize the number of total stages while maintaining a stable solution. To further verify the consistency of the results, we compute the maximum Courant number

$$C = \max_{\boldsymbol{x}\in\Omega} \left|\boldsymbol{f}'(\boldsymbol{u}_h(\boldsymbol{x},t))\right| \frac{\Delta t}{h}.$$

2.4 Convective flux models

This study investigates various conservation laws. For one-dimensional ($d = 1$) scenarios, we examine

- the linear advection equation ($m = 1$):
$$f(u) = \beta u, \tag{6}$$
where the transport field $\beta \in \mathbb{R}$ is constant;
- the Burgers' equation ($m = 1$):
$$f(u) = \frac{1}{2}u^2; \tag{7}$$
- the Euler system, namely, given the ideal gas law for pressure $p = (\gamma - 1)(e - \frac{1}{2}\rho\,|v|^2)$, where $\gamma = 7/5$ is the heat capacity ratio, e is the total energy, ρ is the density and v is the velocity, we have ($m = 3$):
$$\boldsymbol{u} = \begin{bmatrix} \rho \\ \rho v \\ e \end{bmatrix}, \qquad \boldsymbol{f}(\boldsymbol{u}) = \begin{bmatrix} \rho v \\ \rho v^2 + p \\ v(e+p) \end{bmatrix}. \tag{8}$$

The two-dimensional ($d = 2$) models we consider are

- the linear advection equation ($m = 1$):
$$\boldsymbol{f}(u) = \boldsymbol{\beta} u, \tag{9}$$
where the transport field $\boldsymbol{\beta} \in \mathbb{R}^2$ is constant;
- the KPP rotating wave problem [24, 35] ($m = 1$), characterized by a non-convex flux in each component
$$\boldsymbol{f}(u) = \begin{bmatrix} \sin u \\ \cos u \end{bmatrix}; \tag{10}$$
- the Euler system ($m = 4$), defined as
$$\boldsymbol{u} = \begin{bmatrix} \rho \\ \rho v_1 \\ \rho v_2 \\ e \end{bmatrix}, \qquad \boldsymbol{f}(\boldsymbol{u}) = \begin{bmatrix} \rho v_1 & \rho v_2 \\ \rho v_1^2 + p & \rho v_2^2 + p \\ \rho v_1 v_2 & \rho v_1 v_2 \\ v_1(e+p) & v_2(e+p) \end{bmatrix}, \tag{11}$$
where $\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ is the velocity field, e is the total energy, and the pressure $p = (\gamma - 1)(e - \frac{1}{2}\rho\,\|\boldsymbol{v}\|_2^2)$ is defined analogously as before.

3 Artificial viscosity

We provide a concise review of classical artificial viscosity models that hold a pivotal role in understanding the limitations of current methods. We then present the EV model which is the state-of-the-art model serving as the main tool for validating the proposed deep learning physics-informed technique. Finally, we introduce the artificial viscosity models based on neural networks. For the sake of simplicity, we describe the procedure for scalar problems. The standard way to generalize to systems is to choose one of the variables as the representative scalar quantity to be used within the viscosity model.

3.1 An outline of artificial viscosity

Most artificial viscosity models are based on a heuristic that may be very different from model to model. However, it is still possible to recognize some common features that all artificial viscosity models share:

- For each element $K \in \mathcal{K}_h$, a shock indicator measures the size of the local discontinuities and outputs a viscosity μ_K to be added in that cell. The viscosity μ_K must be small or null when the solution is smooth, to avoid modifications to the physics of the problem. In the presence of discontinuities, μ_K must be the smallest value that is sufficient to eliminate Gibbs oscillations.

- Since large viscosity values produce over-dissipation, rendering the model unable to capture significant features of the solution, at every time instant t, and for each element $K \in \mathcal{K}_h$ we define a viscosity threshold

$$\mu_K^{\text{bnd}}(\boldsymbol{x}, t) = \max\{\max_{\boldsymbol{x} \in K} \mu_K(\boldsymbol{x}, t), \mu_{\max}(t)\}, \quad \mu_{\max}(t) = c_{\max} \frac{h}{k} \max_{\boldsymbol{x} \in K} \left|\boldsymbol{f}'(u_h(\boldsymbol{x}, t))\right|, \tag{12}$$

  where $c_{\max}$ is a scalar parameter to be tuned for each problem [34, 48] and $\boldsymbol{f}'$ defines the velocity of propagation of the problem. For $m > 1$ we consider the restriction to the equation of the representative variable. This restriction does not constitute a limitation; all the components of $\boldsymbol{f}'$ can be utilized as inputs for the neural network. The primary drawback of this approach is the increased computational cost. Notice that the definition of $\mu_{\max}$ assures that the viscous contribution is at most linear.

- The viscosity is globally smoothed as outlined in [62]. Despite the existence of advanced techniques [1], we focus on a straightforward three-step piecewise linear interpolator approach (this procedure can easily be generalized to an arbitrary degree):

  1. On each vertex of the discretization $\mathcal{K}_h$, compute the average viscosity among all the elements that share that vertex;
  2. Compute the element-wise linear interpolator, given the average viscosities on the vertexes of each element;
  3. Evaluate the interpolator on the required points in order to build the algebraic system.

  Our numerical findings and [6] support the thesis that the smoothing mitigates numerical oscillations.

In this study, we consider updates to the viscosity which are computed at every time step. However, it is possible to update the viscosity less frequently in order to decrease the computational cost.

3.2 Entropy viscosity

The entropy viscosity model [14, 24, 64] is a state-of-the-art model that produces a viscosity coefficient based on the local size of entropy production. Namely, it is known [5] that the scalar inviscid continuous initial boundary value problem (1) has an unique entropy solution satisfying

$$\partial_t E(u) + \nabla \cdot \boldsymbol{F}(u) \leq 0$$

where the two functions $E : \mathbb{R} \to \mathbb{R}$ and $F : \mathbb{R} \to \mathbb{R}^d$ are such that

$$\boldsymbol{F}(w) = \int E'(w) \boldsymbol{f}'(w) \, \mathrm{d}w, \qquad E \text{ is convex},$$

and are called an entropy pair. We then define the local cell viscosity as

$$\mu_K(\boldsymbol{x}, t) = c_K \left(\frac{h}{k}\right)^2 \frac{\max\{|D_h(\boldsymbol{x}, t)|, |H_h(\boldsymbol{x}, t)|\}}{\left\|E(u_h) - \overline{E}(u_h)\right\|_\infty} \tag{13}$$

where c_K is a parameter that must be manually tuned for each problem, $\overline{E}$ is the spatial integral average of the entropy over the domain, D_h is the residual of the space-time entropy equation

$$D_h(\boldsymbol{x}, t) = \partial_t E(u_h(\boldsymbol{x}, t)) + \nabla \cdot \boldsymbol{F}(u_h(\boldsymbol{x}, t)),$$

and H is the effect of the jump of the entropy flux across the element boundaries

$$H_h(\boldsymbol{x}, t) = \left(\frac{h}{k}\right)^{-1} [\![\boldsymbol{F}(u_h(\boldsymbol{x}, t))]\!] \cdot \boldsymbol{n}.$$

Finally, we impose an upper limit as prescribed by (12). Let us now use this model to make some observations that actually extend to almost all AV models. In particular, we can outline three main weaknesses of classical AV models that we would like to overcome:

(A) The evaluation of the viscosity μ_K^{bnd} may be computationally expensive. In this particular case, computing D_h and H_h is expensive.

(B) They depend on parameter(s) that must be chosen manually. Specifically, $c_{\max}$ and c_K have to be tuned with trial-and-error. This requires deep know-how of the model or a time-consuming phase of trial-and-error.

(C) They do not guarantee optimality: e.g., the EV model is based on an entropy heuristic argument and does not provide any error bound estimate.

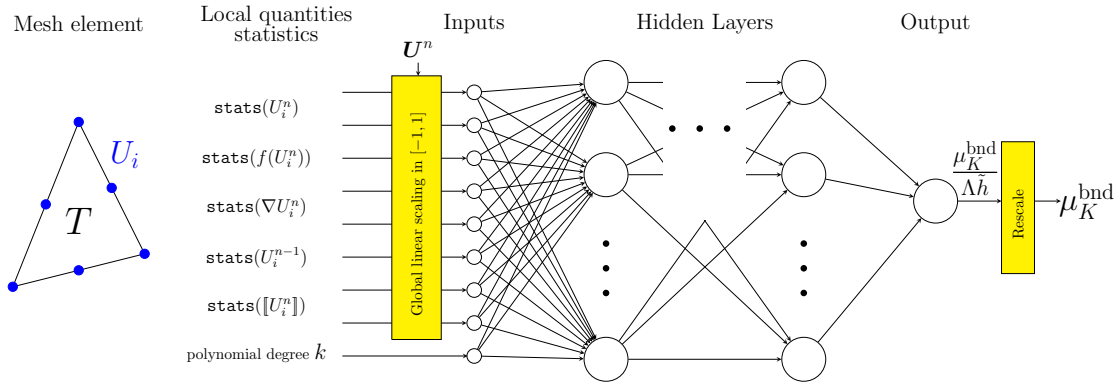

Figure 1: Scheme of the online phase of the neural viscosity approach. The neural network returns as output a viscosity value μ_K^{bnd} for each element of the mesh independently of the other elements by extracting relevant statistics `stats` of the nodal values, namely the minimum, the maximum, the mean, and the standard deviation. Here, U_i^n is the evaluation of u_h at time t^n in the i-th degree of freedom of K.

3.3 Basic concepts of Neural networks

Let $N_I, N_O \in \mathbb{N}$ be strictly positive. An artificial neural network is a function $\mathcal{F} : \mathbb{R}^{N_I} \to \mathbb{R}^{N_O}$ that maps an input vector $\boldsymbol{x} \in \mathbb{R}^{N_I}$ to an output vector $\boldsymbol{y} \in \mathbb{R}^{N_O}$ and depends on a set of parameters that can be thought to be ordered into a vector $\boldsymbol{\theta}$. In this work, we only consider the simplest version of neural networks: dense feed-forward neural networks (FNN), which are defined as follows. Fix two positive integers L and Z. Let $\boldsymbol{y}^{(0)} = \boldsymbol{x}$ and $\boldsymbol{y}^{(L)} = \boldsymbol{y}$. An FNN of depth L and width Z is the composition of L functions called layers defined by

$$\boldsymbol{y}^{(l)} = \sigma^{(l)}(\mathrm{W}^{(l)}\boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)}), \quad l = 1, ..., L,$$

where $\mathrm{W}^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$ are matrices of parameters called weights and $\boldsymbol{b}^{(l)} \in \mathbb{R}^{N_l}$ are vectors of parameters called biases. Here, $N_0 = N_I$, $N_L = N_O$ and $N_l = Z$ for $l = 1, ..., L-1$. Finally, at the end of each layer, a scalar non-linear activation function $\sigma^{(l)}$ is applied component-wise. We employ the same activation function (to be specified later, see Section 4.3) for all the layers apart from the last one where we use a softplus activation function to guarantee the positiveness of the output. It can be seen as a smooth approximation of the ReLU, namely:

$$\sigma^{(L)}(t) = \log(1 + e^t)$$

Several learning strategies are available to determine the vector of parameters $\boldsymbol{\theta}$. Among these, one of the most popular paradigms is supervised learning. Namely, we define and subsequently minimize a non-linear cost function, called loss function $\mathcal{L}$. The optimization is usually performed using a gradient-based optimizer, where the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ is computed by means of automatic differentiation [46]. The loss is defined by means of a given dataset of input target pairs $\{(\boldsymbol{x}, \hat{\boldsymbol{y}})\}_{i=1}^{N_{\text{train}}}$ that the NN must learn. Usually, the loss function takes a form similar to the following

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left\| \hat{\boldsymbol{y}} - \boldsymbol{y}(\boldsymbol{x}; \boldsymbol{\theta}) \right\|_q, \tag{14}$$

where $\|\cdot\|_q$ indicates the discrete q-norm, $q = 1, 2$. A sequence of gradient updates made by iterating over the dataset one time is called an epoch. Even if we will propose an alternative paradigm, this is what is used in literature to train the artificial viscosity model and it is useful to fix this as a benchmark to be compared to our approach. Regardless of the paradigm used to train the NN, if properly optimized, neural networks are able to generalize, meaning that they make good predictions even for input data not used during the training phase. For a comprehensive description of neural networks, we direct the reader to [22, 63].

3.4 The neural viscosity approach

The neural viscosity approach, that is the idea of using NNs as a surrogate for AV models, was first proposed in [16, 54]. Indeed, NNs are an attractive substitute for classical AV models since they overcome the drawbacks (A) and (B) in the following way:

(sA) NNs are computationally efficient during the online stage since they involve mainly simple matrix-matrix multiplications. Moreover, in recent years large effort was put into developing efficient libraries and hardware to accelerate these linear algebra operations.

(sB) NNs can learn highly complex and non-linear functions without the necessity of manually specifying parameters after the training. By leveraging the information present in a large dataset during the training phase, NNs can learn and generalize beyond a single problem.

The architecture that we employ for our NN introduces some improvements with respect to the classical one. The real novelty of this paper lies in how we tackle the third drawback (C), that is how to train the NN and what it learns (or better discovers), which is a matter of Section 4.

The way neural viscosity works is macroscopically the same in which any AV model works (as described in Section 3.1). The NN works independently on each cell, it receives as inputs some local quantities, and it outputs an artificial viscosity value μ_K^{bnd} for each cell K and finally, it performs a global smoothing step. Let us now describe this procedure in detail. The first difference with respect to the original neural viscosity model is that, instead of building and training a different NN for each polynomial degree $k > 0$, we build just one independently of k, reducing training costs. This is enabled by the fact that instead of feeding the local nodal values of the solution $\boldsymbol{U}^n|_K$ (the evaluation of u_h at time t^n in the degrees of freedom of K, which is equal to $u_h|_K(t^n)$ since we are employing a nodal basis) to the NN, we extract some representative statistics of these values, namely the mean and the standard deviation and we add as input also the local polynomial degree k. The input can be enriched with other information such as the median, the quartiles, the minimum, or the maximum. In Section 4.3 we present numerical experiments that show the tradeoff between accuracy and computational cost of this choice. The second difference is that to accelerate the training, we employ as inputs of the neural network not only the current state variable but all the following features:

- state variable;
- the gradient of the state variable;
- the jump of the state variable across the element boundary, since it is a key information to assess the smoothness of the current solution;
- the state variable at the previous time step, to provide information about temporal correlation;
- flux of the state variable, to provide information about the physics of the problem.

Then, as in the original neural viscosity model, we apply a linear scaling to map the inputs in the interval $[-1, 1]$ and a linear rescaling of the targets μ_K^{bnd} of the network. This is common practice for several reasons, including faster convergence during the training (prevents issues like vanishing or exploding gradients) and robustness to variations in input magnitudes (which also is improved generalization). The third difference is that our scaling, instead of being with respect to the local maximum over the element is done with respect to the global maximum. Finally, after the output of the NN, we apply the following rescaling for the element K

$$y = \frac{\mu_K^{\text{bnd}}}{\Lambda \tilde{h}}, \quad \text{where} \quad \Lambda = \max_K \left| \boldsymbol{f}'(u_h) \right|, \quad \tilde{h} = \min\{ \max_{\partial K} |[\![u_h]\!]| , h \}.$$

Observe that we denote with y the output of the NN since it is a scalar quantity. Moreover, the scaling is time dependent, so it changes at every time step. The insight behind this rescaling is that Λ makes the same network work for different PDEs since it represents the maximum local wave speed, and $\tilde{h}$ provides the optimal viscosity scaling h^{k+1} in the presence of smooth solution [31]. Indeed, even if the NN prescribes low dissipation when the solution is regular, due to the nature of the softplus function, it is never zero, which prevents the method from having optimal convergence. Instead, $\tilde{h}$ is a parameter-free option that provides the following scaling

$$\tilde{h} \sim \begin{cases} h^{k+1} & \text{if } u \text{ is smooth,} \\ h & \text{otherwise} \end{cases}$$

which enables a correct reduction of the artificial viscosity when the solution is smooth. In other words, at each timestep, the neural network prescribes a single viscosity value for each element of the mesh. A complete scheme of what we have just described is shown in Figure 1.

4 Physics-informed machine learning

In this section, we describe in detail how the neural network is trained. We propose a novel approach that is inspired by reinforcement learning and is enhanced by physics-informed machine learning. Namely, thanks to automatic differentiation, we can differentiate through the DG solver (the *environment* in RL terminology) and define a loss function (the opposite of the *reward*) as the norm of the error between a reference solution and the solution obtained with the current parameters of the NN (the *policy*) that surrogates a viscosity model (*agent*). The loss is also enhanced with suitable regularization terms. To conclude the comparison

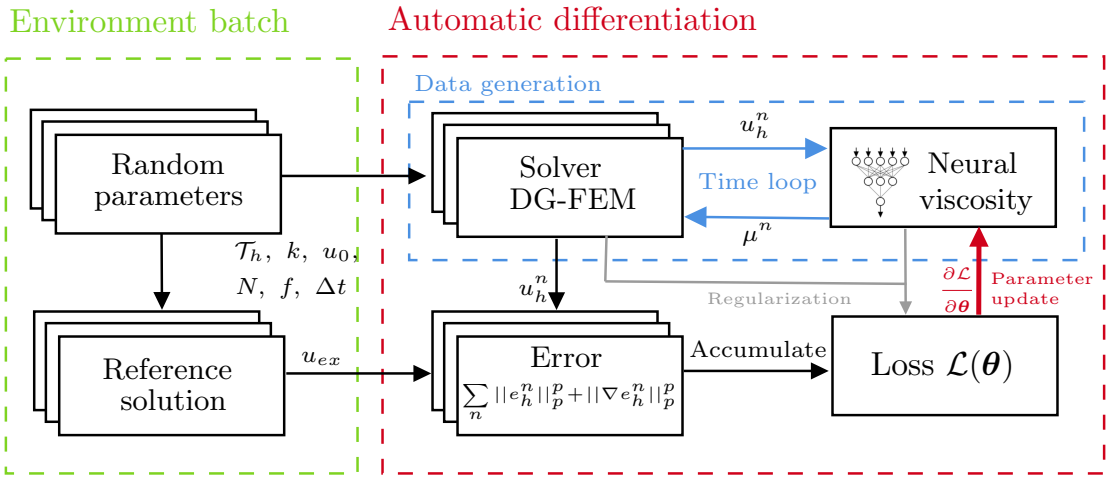

Figure 2: Schematic description of the proposed training procedure.

with RL terminology, the local statistics extracted from each mesh element are the *state* and the viscosity μ_K^{bnd} the *action* the agent chooses. The major difference with respect to RL is that we are able to differentiate through the environment. Thus, by using a gradient-based optimizer, we can directly compute updates to the parameter of the NN so to minimize the loss (maximize the reward). Hence, this training algorithm enables us to discover a new neural viscosity model that has the property of minimizing the norm of the error. This approach allows us to avoid building a dataset with a reference value of artificial viscosity. Indeed, many RL algorithms overcome the problem of a non-differentiable environment using a second neural network called *critic* that surrogates it. This is however expensive, since the environment dynamic must also be learnt, and may lead to instabilities due to the interaction of the policy with the critic [39]. The choice of using physics-informed machine learning, and in particular automatic differentiation through the DG solver, is motivated mainly by efficiency. Indeed, automatic differentiation enables the integration of the DG solver dynamics into neural network training, ensuring precision and efficiency. Concerning the architecture, the specific use of a dense feed-forward neural network is motivated by its simplicity, flexibility, and universality. They are straightforward to implement, computationally efficient, and can be tailored to various problems by adjusting their architecture.

4.1 Loss function

The definition of the loss function is more important in this context than usual since it steers the optimizer toward the discovery of a neural viscosity model with certain properties. To explain how the loss works, let us make the following remark. Given the discretized initial condition $\boldsymbol{U}^0$, the solution at the next time step $\boldsymbol{U}^1$ depends on the artificial viscosity model we choose, which in return depends on $\boldsymbol{U}^0$. More in general we could say that

$$\boldsymbol{U}^n = \boldsymbol{U}^n(\mu(\boldsymbol{U}^{n-1};\boldsymbol{\theta})) = \boldsymbol{U}^n(\mu(\mu(\boldsymbol{U}^{n-2};\boldsymbol{\theta});\boldsymbol{\theta})) = \ldots = \boldsymbol{U}^n(\boldsymbol{U}^0;\boldsymbol{\theta}), \quad \forall n > 0,$$

where the variable $\boldsymbol{\theta}$ explicits the dependence on the parameters of the NN. Hence, given a discrete reference solution $\boldsymbol{U}_{ref}^n$, that is the evaluation of a reference solution $\boldsymbol{u}_{ref} : \Omega \times (0,T] \to \mathbb{R}^m$ at the degrees of freedom at time t^n, by employ automatic differentiation we can compute

$$\frac{\partial}{\partial \boldsymbol{\theta}} \left\| \boldsymbol{U}^n(\boldsymbol{U}^0;\boldsymbol{\theta}) - \boldsymbol{U}_{ref}^n \right\|_q \quad \forall\, n > 0.$$

This is one of the main ingredients of our loss: using an optimizer, we can find the set of parameter $\boldsymbol{\theta}$ that defines the neural viscosity model that minimizes the error with respect to the reference solution. More precisely, the solution $\boldsymbol{u}_{ref}$ is either the analytical solution (when available) or an overkill solution obtained using the DG solver on a triangulation that has a mesh size h that is sufficiently small (in our computations eight times smaller). We define the q-loss as follows:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta};\mathscr{P}) = \sum_{n=1}^{N} \Bigg[&\left\| \boldsymbol{U}^n(\boldsymbol{\theta}) - \boldsymbol{U}_{ref}^n \right\|_q^q + w_0 \left\| \nabla \boldsymbol{U}^n(\boldsymbol{\theta}) - \nabla \boldsymbol{U}_{ref}^n \right\|_q^q \\ &+ \sum_{\ell_{r,q} \in \mathcal{R}} w_r \ell_{r,q}^q(\boldsymbol{U}^n(\boldsymbol{\theta}), \boldsymbol{U}_{ref}^n) \Bigg],\end{aligned} \tag{15}$$

where $\mathcal{R}$ is a set of regularization terms $\ell_{r,q}$ in q-norm that will be defined below and

$$\mathscr{P} = \{\mathcal{K}_h, k, N, \boldsymbol{f}, \Delta t, \boldsymbol{U}^0, \boldsymbol{u}_{ref}\}, \tag{16}$$

is a tuple that uniquely defines a problem. Namely, a triangulation $\mathcal{K}_h$, polynomial degree of the basis k, number of timesteps N, physical flux $\boldsymbol{f}$, time discretization Δt, initial condition $\boldsymbol{U}^0$ and the reference solution $\boldsymbol{u}_{ref}$. Notice that, for the sake of simplicity, we have removed the explicit dependence on $\boldsymbol{U}^0$ of $\boldsymbol{U}^n$ for $n > 0$. Indeed, in this loss function, it is implicit the fact that the DG solver described in Section 2.2 is employed to compute $\boldsymbol{U}^n$ from $\boldsymbol{U}^0$. Notice that with a slight abuse of notation, we apply (linear) differential and integral operators to $\boldsymbol{U}^n$, with the meaning of considering the vector of coefficients describing the functions obtained by applying said operator to $\boldsymbol{u}_h(\boldsymbol{x}, t)$. The metrics that we consider (which is a superset of the regularization terms $\ell_{r,q}$ in $\mathcal{R}$) are the following:

- Error:
$$\epsilon(t^n) = \left\|\boldsymbol{U}^n - \boldsymbol{U}^n_{ref}\right\|_q. \tag{17}$$

- Gradient of the error:
$$\nabla\epsilon(t^n) = \left\|\nabla\boldsymbol{U}^n - \nabla\boldsymbol{U}^n_{ref}\right\|_q. \tag{18}$$

- Jump of the error:
$$[\![\epsilon]\!](t^n) = \left\|[\![\boldsymbol{U}^n - \boldsymbol{U}^n_{ref}]\!]\right\|_q. \tag{19}$$
The insight of this term comes from the jump-jump term of the DG formulation: we need to penalize large jumps in order to obtain a solution that is not too oscillatory.

- Overshoot and undershoot (o/u):
$$\begin{aligned}\mathrm{o/u}(t^n) = &\left\|\boldsymbol{U}^n \mathbb{1}_{\boldsymbol{U}^n > \max \boldsymbol{U}^n_{ref}} - \max \boldsymbol{U}^n_{ref}\right\|_q \\ &+ \left\|\boldsymbol{U}^n \mathbb{1}_{\boldsymbol{U}^n < \min \boldsymbol{U}^n_{ref}} - \min \boldsymbol{U}^n_{ref}\right\|_q,\end{aligned} \tag{20}$$
where $\mathbb{1}_S$ is the indicator function of a set S. The amplitude of overshoots and undershoots with respect to a reference solution is one of the main criteria with which to evaluate a numerical solution since it provides insights into the behavior and stability of a system. To apply backpropagation to this term, we do not implement it as a product of an indicator function, but we restrict the computations only to the entries of $\boldsymbol{U}^n$ that meet the condition stated in the indicator function.

- Mass variation (mv):
$$\mathrm{mv}(t^n) = \left|\int_\Omega \boldsymbol{U}^n - \int_\Omega \boldsymbol{U}^{n-1}\right|. \tag{21}$$
We employ this term for problems that have zero mass flux through $\partial\Omega$. Checking that this term is small is key to verifying that the physics of the problem is preserved.

- Viscosity penalization (vp):
$$\mathrm{vp}(t^n) = \frac{\frac{1}{|K|}\int_K \mu}{\|\nabla\boldsymbol{U}^n\|_q + \varepsilon}, \tag{22}$$
where ε is a small number used for numerical stability, we choose $\varepsilon = 10^{-8}$. This term makes sure that we add artificial viscosity only when there are large gradients, therefore encouraging the network to put the least amount possible of viscosity. This term can be substituted by a "supervised" term of the kind $|\mu_K - \hat{\mu}_K|$, where $\hat{\mu}_K$ is a reference artificial viscosity given by a classical model. In exchange for a stronger bias on what the NN should learn, this alternative makes the training easier since it steers the learning dynamic towards a known good solution. Still, since this is only a weak constraint, the network is able to discover something new.

Not all of them are used in the final version of the loss, indeed we use the minimal number of terms that produce a good model since each term has a weight w_r that needs to be tuned (just once offline). The other terms are still used as metrics to assure the quality of the neural viscosity model. Namely, we have found that the best approach is to employ as the set of regularization terms $\ell_{r,q}$ in $\mathcal{R}$ the overshoot/undershoot (o/u) and the viscosity penalization (vp). The weights in the loss function for these two terms are hyperparameters to tune.

Another key choice is the value of q. We pick $q = 1$ since it tends to emphasize sparsity in the data, meaning it is less sensitive to small values and can be more robust in the presence of outliers. Namely, it

Algorithm 1 Training algorithm

procedure TRAINING($\{\mathscr{P}_j\}_{j=1}^{N_p}$, $\{\mathscr{P}_j^{\text{test}}\}_{j=1}^{N_{pt}}$, `lr`, scheduler, N_E, N_B, $\boldsymbol{\theta}$, $\mathcal{L}$, $L_{\max}$, n_{test})
  $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}, L_{\text{best}} \leftarrow \infty$
  for e **in** N_E **do** ▷ *Loop over each epoch*
    `lr` $\leftarrow$ scheduler(`lr`) ▷ *Update learning rate according to schedule*
    Randomly permute indexes j in $1, ..., N_p$
    for i **in** $N_p/N_B + \text{mod}(N_p, N_B) > 0$ **do** ▷ *Partitions problems in sets of size N_B*
      Reset gradient accumulation in the optimizer
      $L = 0$ ▷ *Reset the loss for the minibatch*
      for j **in** $jN_B, ..., \min\{N_p, (i+1)N_B\}$ **do** ▷ *For each problem in the batch*
        $L \leftarrow L + \mathcal{L}(\boldsymbol{\theta}; \mathscr{P}_j)$ ▷ *$\mathcal{L}$ is defined by q and the weights $\{w_i\}_{i\in\mathcal{R}}$, see Eq. (15)*
      end for
      $\boldsymbol{\theta} \leftarrow \text{AdamW}(L, \boldsymbol{\theta}, \texttt{lr})$ ▷ *Optimization step*
    end for
    if $\text{mod}(e, n_{\text{test}}) = 0$ **then** ▷ *Test outside the automatic differentiation*
      $L_{\text{test}} = \sum_{j=1}^{N_{pt}} \mathcal{L}(\boldsymbol{\theta}; \mathscr{P}_j^{\text{test}})$
      if $L_{\text{test}} < L_{\text{best}}$ **then**
        $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$ ▷ *Save model parameters because currently the best*
      else if $L_{\text{test}} > L_{\max}$ **then**
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^*$ ▷ *The model is unstable: load last stable model's parameters*
        `lr` $\leftarrow$ `lr`$/2$ ▷ *Force learning rate reduction*
      end if
    end if
  end for
return $\boldsymbol{\theta}^*$ ▷ *Parameters of the trained NN*
end procedure

is more appropriate in the presence of oscillations that have sparse and isolated peaks, which is our case. Indeed, we have tested other values of q, such as $q = 2$, and they produced more oscillating results.

The computations of Δt through Eq. (5) must be treated carefully. Indeed, the NN tends to increase to infinity the viscosity μ in order to have a timestep size Δt which is zero. Indeed, as $T = t^N \to 0$ then the loss tends to zero. There are different solutions to this problem: we can leave the computation outside the AD, we add a penalization term based on $\max_\Omega \mu$ or we can use a fixed Δt.

4.2 Training algorithm

Let us now detail the novel algorithm through which the training is performed. In particular, this training process does not belong to the paradigm of supervised learning since the explicit value of viscosity that the neural network should learn for a certain value of inputs is never computed. Our approach is built on the idea that target viscosity values are not directly used to train the model. Instead, we utilize reference solutions to guide the learning process. However, the approach can neither be classified as unsupervised: when an exact solution is not available, a reference solution must be built using a viscosity model with suitably chosen parameters. The training algorithm is stated in Algorithm 1 with detailed comments and is summarized in Figure 2. It works as follows. It takes as inputs:

- A list of N_p tuples of parameters describing the N_p problems $\{\mathscr{P}_j\}_{j=1}^{N_p}$ on which to train the viscosity model on, as defined in Eq. (16).
- A list of N_{pt} problems $\{\mathscr{P}_j^{\text{test}}\}_{j=1}^{N_{pt}}$ on which to test the algorithm.
- An optimizer with its hyperparameters. For instance, we employ AdamW [41] with learning rate $\texttt{lr} = 1\text{e}-3$ and default hyperparameters $\beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}, \lambda = 0.01$.
- A learning rate scheduler [40]. We employ a simple "reduce on plateaus" [47] with patience 30 (number of epochs to wait for a loss reduction) and reduction factor equal to $\frac{1}{2}$ (factor by which the learning rate is reduced).
- N_E, the number of epochs for which the training lasts.
- N_B, the batch size of problems considered in one optimization step (gradient computation).
- $\boldsymbol{\theta}$, the parameters of the NN. Their shape implicitly defines the architecture of the NN. It is assumed they are already initialized with standard random initialization or a pre-training procedure.

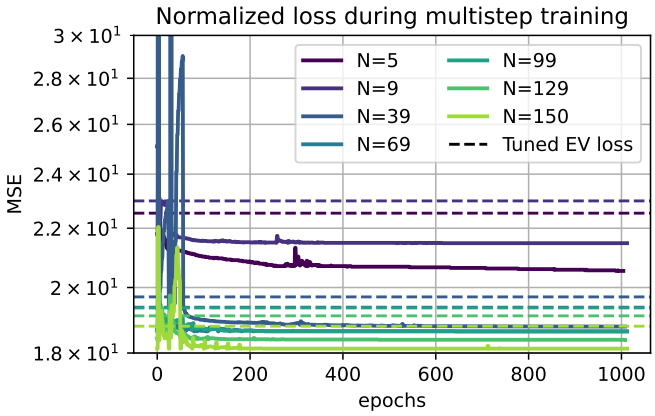

Figure 3: Example of the behavior of the loss for the proposed algorithm during the training for the multi-step procedure described in Section 4.2 for a one-dimensional problem.

- $\mathcal{L}$, the loss function defined in Eq. (15). It is defined by q and the weights $\{w_i\}_{i\in\mathcal{R}}$.
- $L_{\max}$, the maximum value of the loss. If the loss is larger than this threshold the model is considered unstable. We use $L_{\max} = 10^{30}$.
- n_{test}, the frequency with which the model is tested.

Then, the training algorithm closely resembles a supervised learning loop, where the loss is substituted with the one we described in the previous section. Namely, we create minibatches of parameters and make updates based on these parameters. The key point being that we do not have a target value that the NN should learn but a loss that describes the physics of the problem through a DG solver. Indeed, the peculiarity of this training algorithm is that, like in RL, it learns from the consequences of the actions (choices of μ_K) it makes in the DG solver, creating a feedback loop. In particular, the value of artificial viscosity at the timestep n influences the solution at $n+1$ which in turn is used as input of the network to determine the next value of the viscosity. This enables the model to learn long-term dependencies and optimal strategies in response to evolving scenarios.

Among the most interesting features of this algorithm is that, compared to the supervised training process, we can avoid designing and building a dataset. This is particularly expensive from the point of view of human time since it requires tuning the artificial viscosity parameters by hand and picking the best choice by confronting the various solutions. In particular, differently from the classical loss Eq. (14), our loss Eq. (15) necessitates only a reference solution, which is much cheaper to build. Coincidentally, this feature also solves the aforementioned drawback (C), since the neural network automatically learns the optimal viscosity values. Since the problem is highly non-convex, this cannot be a global optimum, however, we guarantee that this is at least a local minimum.

Let us now briefly discuss the choice of hyperparameters for this training algorithm. We observe that we obtain better results with a small batch size, usually between two and four. It is paramount to choose N properly: if it is too small, the neural network does not have enough data to generalize, if it is too large we may incur into exploding gradient problems unless the neural network has a proper weight initialization. We propose a multi-step pre-train in which we iterate the training Algorithm 1 for increasing values of N for a few epochs, starting from $N = N_0$ up to the desired final N with a certain step N_e. The increase in number of timesteps N_e is computed adaptively, namely, defined

$$\texttt{clip}(a, b, c) = \max(\min(b, c), a),$$

we compute N_e as $N_e = \texttt{clip}(N_e^{\min}, N_e^{\alpha}, N_e^{\max})$, where N_e^{α} is the largest integer so that the average loss per timestep with $N + N_e^{\alpha}$ timesteps is less than a constant (hyperparameter) $\alpha_e > 0$ multiplied by the average loss per timestep with N timesteps. Once the NN reaches good generalization properties it is observed that the average loss per timestep decreases when N increases. Indeed, this shows that the neural viscosity model is making accurate predictions of the "future" (data it has never seen before). Figure 3 shows an example of the training loss. This process is expensive from a computational point of view, however, it enables the NN to discover a viscosity model without any biases.

Since the magnitude of the loss changes significantly between the start and the end of the optimization process, a key ingredient to increase the stability and efficiency of the algorithm is a learning rate scheduler. A simple scheduler that geometrically reduces the error on plateaus worked best for us, even compared to more modern and popular alternatives like the cosine scheduler [40]. This also allows us to have an algorithm that

1D initial conditions
$u_0(x) = \frac{\omega}{2}\sin(\omega\pi x), \quad \omega = 1,3,5$
$u_0(x) = \mathbb{1}_{[\frac{1}{4},\frac{3}{4}]}$
$u_0(x) = 10(\frac{1}{2} - \lvert x - \frac{1}{2}\rvert)$
$u_0(x) = e^{-100(x-\frac{1}{2})^2}$
$u_0(x) = \omega_1\mathbb{1}_{[0,\frac{1}{5}]} + \omega_2\mathbb{1}_{[\frac{1}{5},\frac{2}{5}]} + \omega_3\mathbb{1}_{[\frac{3}{5},1]}, \quad \omega_i = -4,6,10$
$u_0(x) = \sin(\omega\pi x)\mathbb{1}_{[\frac{1}{4},\frac{1}{2}]} + \sin(2\omega\pi x)\mathbb{1}_{[\frac{1}{2},\frac{3}{4}]}, \quad \omega = 4,8$
$u_0(x) = -\sin(6\pi x)\mathbb{1}_{[\frac{1}{6},\frac{5}{6}]}$
$u_0(x) = \omega_1(x-\frac{1}{6})\mathbb{1}_{[\frac{1}{6},\frac{1}{3}]} + \omega_2(x-\frac{1}{2})\mathbb{1}_{[\frac{1}{3},\frac{2}{3}]}$ $+\omega_3(x-\frac{5}{6})\mathbb{1}_{[\frac{2}{3},\frac{5}{6}]}, \quad \omega_i = 2,6,10$
$u_0(x) = (16\lvert x - \frac{1}{2}\rvert - 2)\mathbb{1}_{[\frac{1}{4},\frac{3}{4}]}$

2D initial conditions
$u_0(\boldsymbol{x}) = \frac{\omega_1\omega_2}{2}\sin(\omega_1\pi x_1)\sin(\omega_2\pi x_2), \quad \omega_i = 1,3,5$
$u_0(\boldsymbol{x}) = \mathbb{1}_{[\frac{1}{4},\frac{3}{4}]}(x_1)\mathbb{1}_{[\frac{1}{4},\frac{3}{4}]}(x_2)$
$u_0(\boldsymbol{x}) = e^{-100\lVert\boldsymbol{x}-\frac{1}{2}\mathbf{1}\rVert^2}$
$u_0(\boldsymbol{x}) = \omega_1\mathbb{1}_{[0,\frac{1}{5}]}(x_1)\mathbb{1}_{[\frac{3}{5},1]}(x_2)$ $+\omega_2\mathbb{1}_{[\frac{1}{5},\frac{2}{5}]}(x_1)\mathbb{1}_{[\frac{1}{5},\frac{2}{5}]}(x_2)$ $+\omega_3\mathbb{1}_{[\frac{3}{5},1]}(x_1)\mathbb{1}_{[0,\frac{1}{5}]}(x_2), \quad \omega_i = -4,6,10$
$u_0(\boldsymbol{x}) = \sin(\omega\pi x)\mathbb{1}_{[\frac{1}{4},\frac{1}{2}]}(x_1)\mathbb{1}_{[\frac{1}{4},\frac{1}{2}]}(x_2)$ $+\sin(2\omega\pi x)\mathbb{1}_{[\frac{1}{2},\frac{3}{4}]}(x_1)\mathbb{1}_{[\frac{1}{2},\frac{3}{4}]}(x_2), \quad \omega = \frac{5}{2},4,8$
$u_0(\boldsymbol{x}) = (\omega_1 x_1 + \omega_2 x_2 - \frac{1}{4})\mathbb{1}_{[\frac{1}{4},\frac{3}{4}]}(x_1)\mathbb{1}_{[\frac{1}{4},\frac{3}{4}]}(x_2)$ $\omega_i = -1,0,1,4$

Table 1: Initial conditions u_0 employed in the training phase for advection and Burgers' fluxes.

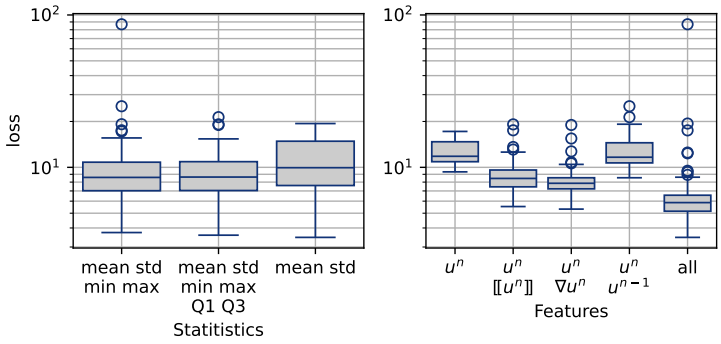

Figure 4: Loss of the trained viscosity model depending on the nodal quantities used as input of the NN (*right*) and on the functions used to agglomerate those values (*left*). See Section 3.4 for a complete definition of these quantities. The results are obtained after one step of the random search relative to the NN employed in the 1D cases.

is less influenced by the starting value of the learning rate since, if it is too large, is automatically decreased. In our experiments, we employed a starting learning rate of $5 \cdot 10^{-2}$.

Finally, we stress that, despite all these precautions, it seldom happens that a gradient update produces a viscosity model that is not able to handle the Gibbs phenomenon. For this reason, it is important to constantly monitor the loss, save the best parameters of the model, and use them for a restart with a reduced learning rate. This simple but effective procedure completely eradicates this issue.

4.3 Training in practice

In this section, we provide more details on how the training is carried out in the one-dimensional ($d = 1$) case. This procedure can be easily generalized to different dimensions. The first step is to choose which problems to train the neural network on, namely, we need to choose different combinations $\{\mathscr{P}_j\}_{j=1}^{N_p}, \{\mathscr{P}_j^{\text{test}}\}_{j=1}^{N_{pt}}$ of problems on which to train and test. The two sets are an 80-20 random split of a union of cartesian products of parameter spaces. For the 1D case we consider uniform discretizations of $(0,1)$ with $h = \frac{2^{-i}}{10}, i = 1,...,4$, degrees $k = 1,2,4$, the flux $\boldsymbol{f}$ of linear advection, Burgers' equation and Euler system, a constant timestep Δt that is the largest such that the considered problem is stable (it varies from problem to problem), number of timesteps $N = 100$ for linear advection and appositely chosen as the smallest $N \in \mathbb{N}$ such that shocks and waves appear in Burgers' and Euler cases. Concerning initial conditions, we use the functions reported on the left of Table 1 for advection and Burgers' problems, and a Sod shock tube problem [57] for Euler's. A representation of the initial conditions is shown in Figure 8. For the 2D case, the problems are chosen in an analogous way apart from the following parameters. Since unstructured grids offer flexibility in discretizing computational domains, allowing for efficient representation of intricate shapes and unbounded regions, we use both structured and unstructured triangulations $\mathcal{K}_h$ to train the NN. Namely, we discretize $(0,1)^2$ with $h = \frac{2^{-i}}{10}, i = 1,...,3$. The initial conditions are shown on the right of Table 1 for advection and Burgers' equations. We also add the Euler's system in the Riemann configuration 12 of [36] to the training environment.

Once the parameters defining the training problems are chosen, we pass to hyperparameter tuning. As it is commonly done, we employ a random search: we specify the range or distribution for each hyperparameter we want to tune, and we define the search space as the cartesian product of these ranges. The model is

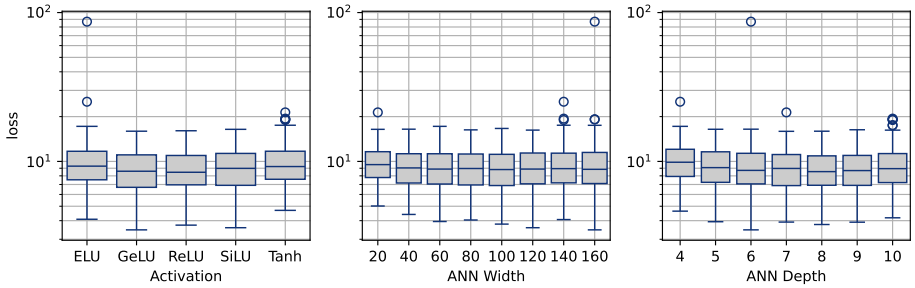

Figure 5: Loss of the trained viscosity model depending on hyperparameters defining the architecture of the NN agglomerated in boxplots. The results are obtained after one step of the random search relative to the NN employed in the one-dimensional cases.

trained using random choices of hyperparameter values, and the loss is used as an evaluation metric. This process is repeated while, at each iteration, reducing the search space and increasing the fraction of the combinatorial space probed and the number of epochs N_E.

As an optimizer, we employ AdamW (Adam with Weight Decay) [41]. In our experiments, it significantly outperformed stochastic gradient descent (SDG) and proved to be less sensitive to the choice of the learning rate with respect to Adam.

We then focus on five hyperparameters that we consider to be particularly relevant, namely the width and depth of the NN, the activation function, the local features extracted from each element K, and which statistics are extracted from these features. In Figure 4 and 5 we show the results after one iteration of random search (about ten thousand combinations were tested). After three iterations of random search we, conclude the following (for the one-dimensional case):

- using the mean, standard deviation, minimum, and maximum of the nodal values as describing statistics provides the best balance between computational cost and accuracy;
- the gradient of the state variable, the jump across the element, and the state variable at the previous timestep are all features that significantly help the training;
- the Gaussian error linear unit (GeLU) [29] is the activation function that we choose since it reached the smallest loss when compared to the rectified linear unit (ReLU), exponential linear unit (ELU), sigmoid linear unit (SiLU) and hyperbolic tangent (Tanh);
- a large model, with a width greater than 100, has a smaller loss on average. In particular, we choose to use 120 neurons per layer;
- models with depth between six and eight (which are neither shallow nor very deep) reach the lowest loss on average. In our experiments, we employ seven layers.

Therefore, the input vector of the neural network has size 19 (4 for the reference variable, its gradient, its value at the previous time step, its flux; 2 for the jump; 1 for the polynomial degree) in 1D and 29 in 2D (4 for the reference variable, its gradient, its value at the previous time step, its flux and its jump; 1 for the polynomial degree)

5 Numerical Results

In this section, our objective is to assess the accuracy of the proposed training algorithm in practice. In the following, we present a comprehensive set of numerical results, encompassing both scalar equations and systems, to showcase the efficacy of physics-informed machine learning. All the tests have been implemented in an appositely developed Python library for the solution of conservation laws with the DG method that employs Pytorch [47] for automatic differentiation and the implementation of the NNs.

First, we present a convergence test to verify the functioning of our numerical solver and neural viscosity model. Then, we present tests carried out on problems that are not included in the training environment and are characterized by different parameters, such as: flux, initial condition, boundary conditions, domain discretization, polynomial degree, and CFL. In this way, we can assess the generalization capabilities of the trained networks and test the robustness of our algorithm in an impartial way. Our findings consistently demonstrate performance superior to the optimally tuned EV models. This is especially relevant considering that tuning is itself a significantly expensive operation.

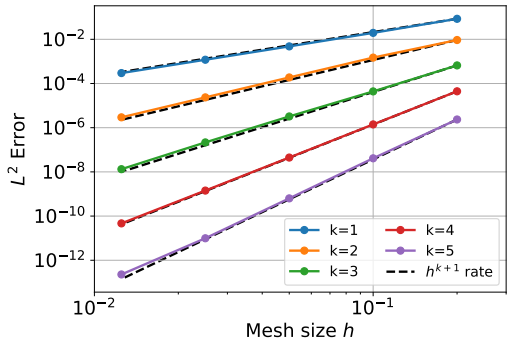

Figure 6: Computed errors and convergence rates for test case of linear advection ($d = 1$).

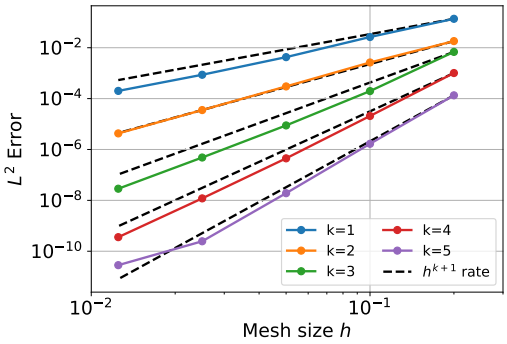

Figure 7: Error estimates and convergence rates for the 2D smooth test case of linear advection.

5.1 Verification tests: convergence rate

Here, we aim to verify the accuracy of the presented approach. In particular, we assess the ability of the neural viscosity model to maintain the expected accuracy for a smooth solution. Indeed, the addition of a dissipative term might adversely affect the accuracy. The assessment of the numerical scheme employs the $L^2(\Omega)$-norm of the discretization error vector, evaluated at the final time T. Namely, theoretical results for inviscid hyperbolic problems [30, 31] show that the error

$$\epsilon = \|u_h(\cdot, T) - u(\cdot, T)\|_{L^2(\Omega)} \lesssim h^{k+1},$$

where the hidden constant depends on k, given that u is sufficiently regular and a Rusanov flux is employed.

5.1.1 Test case 1: one-dimensional smooth problem

To test the convergence we consider Problem (1) with the linear flux defined by Eq. (6), namely with a constant unitary transport field and with the smooth initial conditions

$$u_0(x) = \frac{1}{2} + \sin(2\pi x).$$

The problem is endowed with periodic boundary conditions. Concerning the discretization, we use the spatial domain $\Omega = (0, 1)$, CFL $= 0.05$, final time $T = 0.4$ and Runge-Kutta of the fourth-order. In Figure 6 we report the computed errors in the $L^2(\Omega)$-norm, together with the expected rates of convergence as a function of the mesh size h and the polynomial degree k. We can notice that we recover the expected convergence rate.

5.1.2 Test case 2: two-dimensional smooth problem

Similarly, we extend the previous test case in two dimensions. Namely, we consider Problem (1) with constant unitary transport field $\boldsymbol{\beta} = [1, 1]^\top$ as defined in Eq. (9) with smooth initial conditions

$$u_0(x_1, x_2) = \frac{1}{2} + \sin(2\pi x_1)\sin(2\pi x_2).$$

The problem is endowed with periodic boundary conditions. Concerning the discretization, we employ a structured triangular grid of the domain $\Omega = (0, 1)^2$ of granularity $h = \frac{\sqrt{2}}{2^i 10}, i = 1, ..., 5$. In Figure 7 we report the computed errors in the $L^2(\Omega)$-norm and the expected rates of convergence. The NN technique achieves the optimal convergence rate. Numerical experiments show also that we obtain the same rate of convergence independently of the mesh type (structured or unstructured).

5.2 One-dimensional problems

To gain more insight into how the algorithm works, we first analyze one-dimensional problems. The following test cases can be divided into two categories. The first two deal with scalar problems. To test the generalization properties of the NN, we consider an initial condition, grid spacing, polynomial degree, and CFL not included in the training environment. In particular, we consider Problem (1) with flux defined by

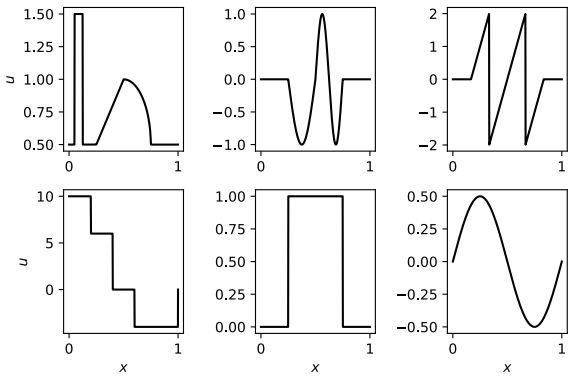

Figure 8: Examples of initial conditions u_0 used to train the NN.

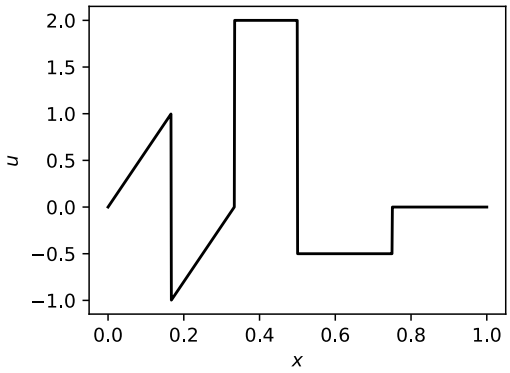

Figure 9: Initial condition u_0 for test case 3 and 4.

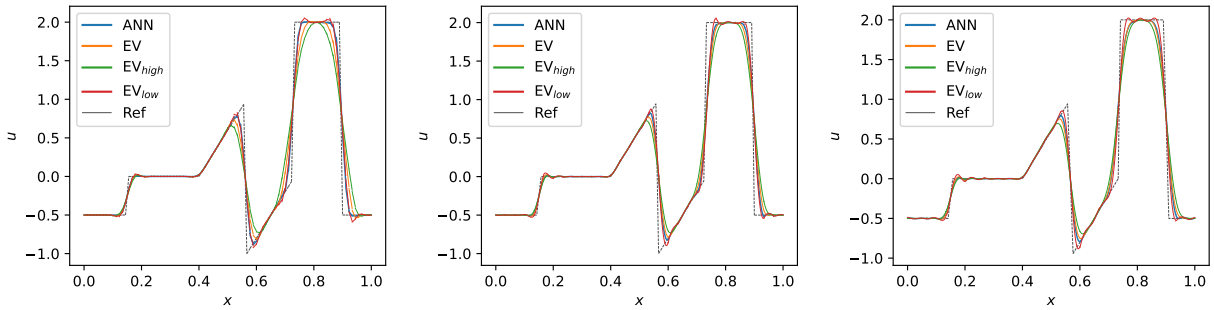

Figure 10: Test case 3: solution at $T = 0.4$ for $k = 1, 3, 5$ (from left to right).

Eq. (6) and Eq. (7) and we employ as initial condition the function

$$u_0(x) = \begin{cases} 6x & x \in (0, \frac{1}{6}], \\ 6(x - \frac{1}{3}) & x \in (\frac{1}{6}, \frac{1}{3}], \\ 2 & x \in (\frac{1}{3}, \frac{1}{2}], \\ -\frac{1}{2} & x \in (\frac{1}{2}, \frac{3}{4}], \\ 0 & x \in (\frac{3}{4}, 1), \end{cases} \tag{23}$$

which is represented in Figure 9. Instead, the other two test cases seek validation in a more challenging setting. Namely, we consider two common tests for the accuracy of the solution of the Euler equations: the Sod shock tube problem [57] and the Shu-Osher problem [56].

5.2.1 Test case 3: linear advection

The first test case we consider is a linear advection problem with a highly discontinuous initial condition, namely (23) in the $\Omega = (0, 1)$ domain. The simulation is performed until $T = 0.4$ with polynomial degrees $k = 1, 3, 5$ and endowed with periodic boundary conditions. To compare solutions with a similar number of degrees of freedom we select $h = \frac{1}{60}, \frac{1}{30}, \frac{1}{15}$, CFL $= 0.2, 0.5, 0.75$ and $C = 0.2, 0.0556, 0.03$, respectively. The solution obtained with the neural viscosity model is compared with a tuned EV model ($c_K = 0.6$, $c_{\max} = 0.3$, see Eq. (12), (13)) and two instances of EV model that are not optimally tuned, namely one case where the injected dissipation is too small and too large. In Figure 10 we compare the solutions at the final time: we can see that the NN based solution preserves the same symmetry properties of the exact solution, essentially non-oscillatory, and well captures the front of the wave. By analyzing the space-time plot of the viscosity introduced via the NN, as shown in Figure 11, we notice that the NN tends to add a larger amount of less localized viscosity when compared with EV for $k = 1$. It is instead more similarly localized for $k = 3, 5$. More quantitative results about the error are shown in Table 2, we notice that the NN always outperforms the EV model in terms of L^1 error and overshoots/undershoots. It also obtains comparable results when considering mass variation (mv), an important metric to asses that the physics of the problem is respected. This comparison is limited by the choice of parameters c_K and $c_{\max}$ we made and by the considered metrics. Indeed, while we manually tuned these parameters, there might exist a choice that leads to smaller error metrics for the EV model.

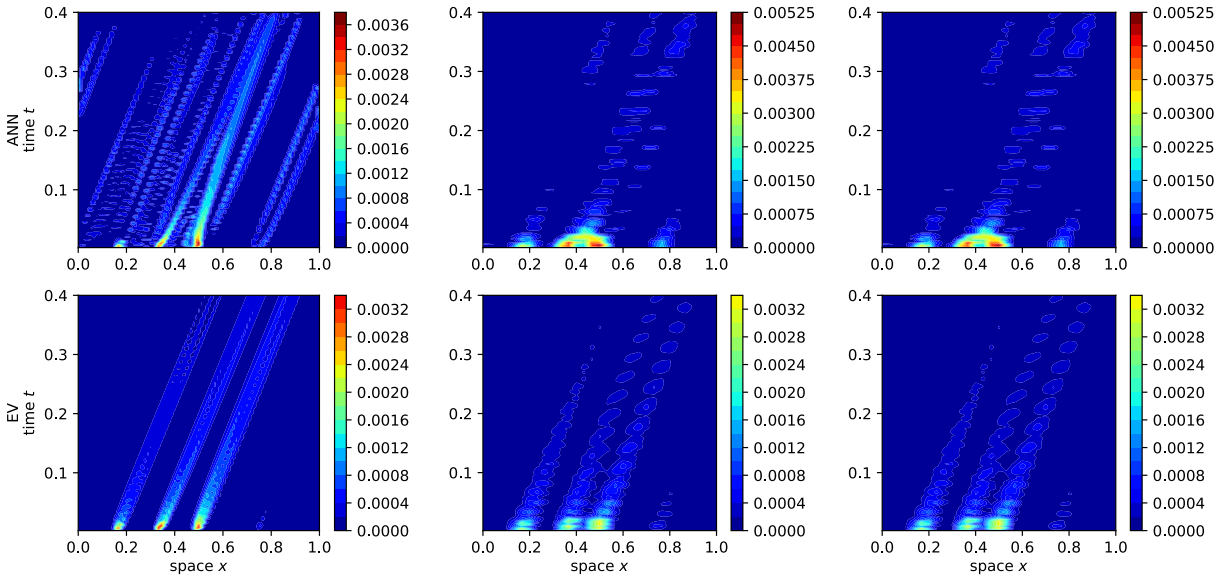

Figure 11: Test case 3: comparison of space-time artificial viscosity for $k = 1, 3, 5$ (from left to right).

	$k=1$		$k=3$		$k=5$	
	NN	EV	NN	EV	NN	EV
ϵ	2.2311e+03	3.1745e+03	1.7990e+03	2.1943e+03	1.5680e+03	1.6753e+03
$\nabla\epsilon$	1.7306e+03	1.9666e+03	6.7370e+03	6.9916e+03	1.1302e+04	1.1380e+04
$[\![\epsilon]\!]$	1.5933e+02	1.1955e+02	4.2461e+01	3.4186e+01	2.0260e+01	2.0562e+01
o/u	3.4889e+00	1.2134e+01	2.2778e+00	6.2186e+00	2.5903e+00	7.3782e+00
mv	4.0360e-03	4.5267e-03	1.6535e-03	2.8121e-03	1.4988e-03	1.6169e-03

Table 2: Test case 3: comparison of cumulative L^1 error metrics Eq. (17) - (21) over all timesteps from 0 to T.

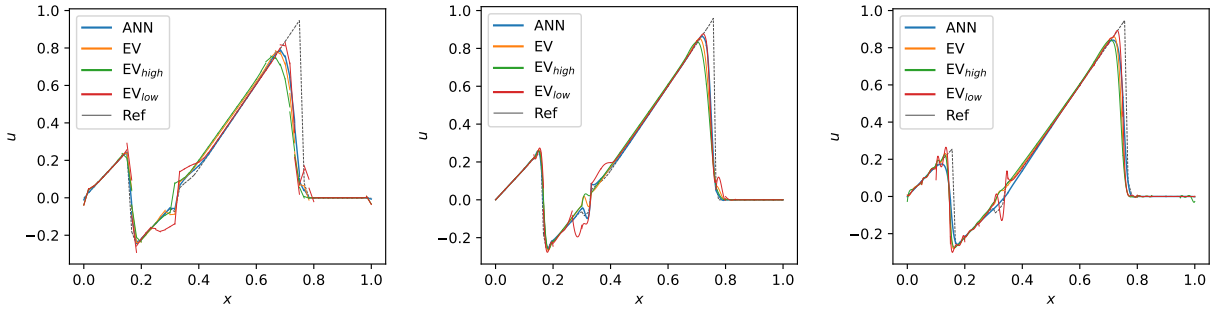

Figure 12: Test case 4: solution at final time for $k = 1, 3, 5$ (from left to right).

	$k=1$		$k=3$		$k=5$	
	NN	EV	NN	EV	NN	EV
ϵ	8.7204e+02	1.1423e+03	9.7797e+02	1.3587e+03	2.6155e+03	3.2874e+03
$\nabla\epsilon$	5.3064e+02	6.1239e+02	3.1401e+03	3.5330e+03	1.4578e+04	1.7510e+04
$[\![\epsilon]\!]$	1.0947e+02	1.1138e+02	4.4744e+01	5.0420e+01	3.4530e+01	1.2684e+02
o/u	6.5979e-01	9.8408e-01	6.5894e-01	1.3003e+00	9.7566e-01	2.5800e+00

Table 3: Test case 4: comparison of cumulative L^1 error metrics Eq. (17) - (20) over all timesteps from 0 to T.

5.2.2 Test case 4: Burgers' equation

In this test case, we keep the same parameters of the previous test case but we change the physical flux, namely, we employ a quadratic one, and the CFL $= 0.15, 0.4, 0.4$. The Courant number is $C = 0.3317, 0.0924, 0.0302$. These settings are more challenging due to the fact that as the solution evolves rarefaction fans and shock waves develop and collide. We compare the NN solution with a tuned EV model ($c_K = 3.0, c_{\max} = 1.0$) and two other not optimally tuned EV models. Figure 12 shows that one of the EV models ($c_K = 1.0, c_{\max} = 0.5$) is not able to add enough viscosity thus leading to large oscillation. The parameters used in this case are not far from the manually tuned: this showcases how time-consuming the parameter selection can be. In this case, the NN model is by far the best in avoiding overshoots/undershoots while correctly predicting the profile of the wave. The history of the viscosity in Figure 13 shows that as in the previous test cases, the NN tends to inject more viscosity than the EV model, in a more spread and less

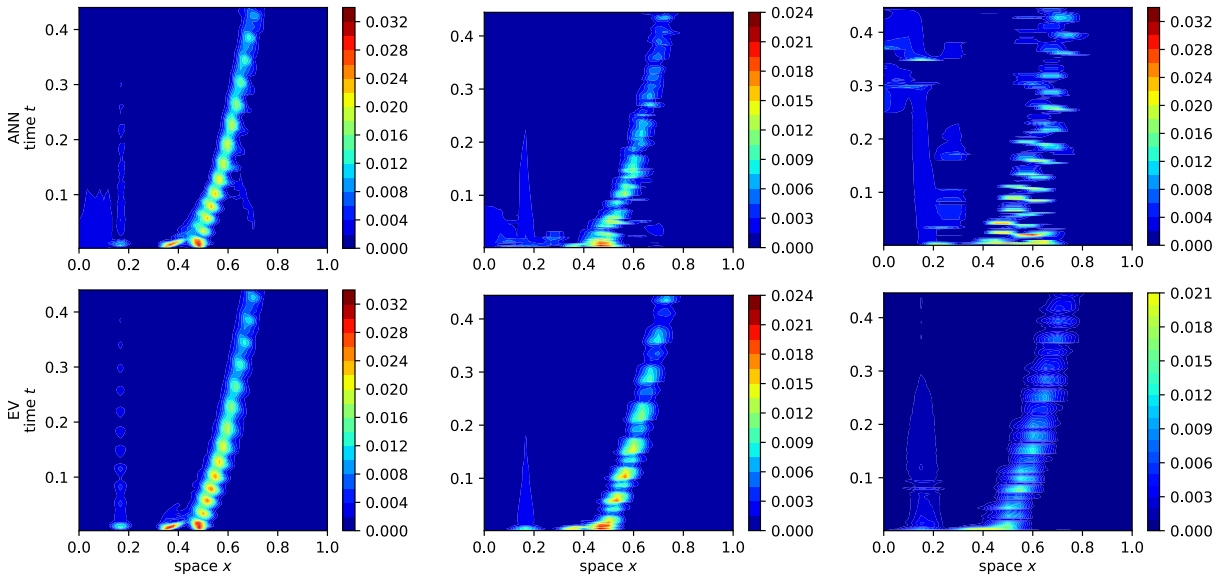

Figure 13: Test case 4: comparison of space-time artificial viscosity for $k = 1, 3, 5$ (from left to right).

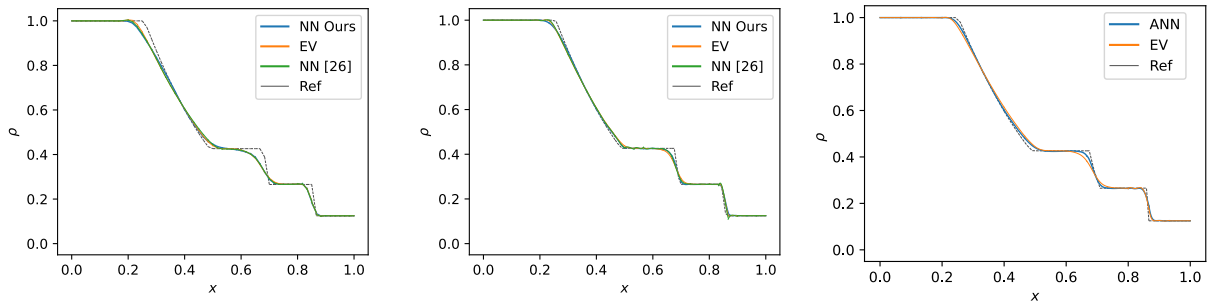

Figure 14: Test case 5: density ρ at $T = 0.2$ for $k = 1, 3, 5$ (from left to right).

continuous way. For $k = 5$ in particular, it is possible to see that the NN also tends to add dissipation in $x = 0.2, t = 0.35$ where the EV model did not. It remains to be investigated if this is a generalization error (since $k = 5$ was not present in the training problems) or a feature of the discovered neural viscosity model. As shown in Table 3, the NN model outperforms all the EV models under all the considered metrics. This is an especially interesting result when considering that the reduction of the L^1 error for $k = 3$ is of 25%, while also reducing overshoots/undershoots of almost 50%.

5.2.3 Test case 5: Sod shock tube

We now consider the first of two test cases for the Euler equations, namely the Sod shock-tube problem [57]. The problem is defined by Eq. (8), it prescribes $\Omega = (0, 1)$, $T = 0.2$ and the initial state

$$\boldsymbol{u}_0(x) = \begin{bmatrix} \rho_0 \\ v_0 \\ p_0 \end{bmatrix} = \begin{cases} [1, 0, 1]^\top & x \in (0, \frac{1}{2}], \\ [\frac{1}{8}, 0, \frac{1}{10}]^\top & x \in (\frac{1}{2}, 1). \end{cases}$$

It is endowed with constant Dirichlet boundary conditions (cf. [57]). For the discretization we select three cases with $k = 1, 3, 5$, $h = \frac{1}{60}, \frac{1}{30}, \frac{1}{15}$, CFL $= 0.27, 0.61, 0.88$, respectively. Even if this problem is present in the training dataset, the polynomial degree k or CFL used to discretize it are different. The Courant number is $C = 0.2970, 0.1393, 0.0785$. This is done to compare solutions with a similar number of degrees of freedom. The discontinuities in the initial conditions give rise to three characteristic waves, demanding robust numerical schemes to accurately compute the solution in the presence of shock waves and rarefactions. Namely, these are a right-moving contact wave and shock wave, and a left-moving rarefaction wave. Classical models aim to add a small amount of viscosity near the contact while constantly introducing dissipation in the region close to the shock. In Figure 14 we compare the NN solution at final time with the EV tune model with $c_K = 1.0, c_{\max} = 0.5$ and the NN method proposed in [16]. Our neural viscosity model provides a solution that has much smaller overshoots (see $x = 0.2, k = 1$ for instance) and that captures with more precision the wavefront (see $x = 0.7, k = 1$ for instance). In this case, it is particularly interesting to analyze the history of the viscosity shown in Figure 15. Namely, here we can notice a large discrepancy between what

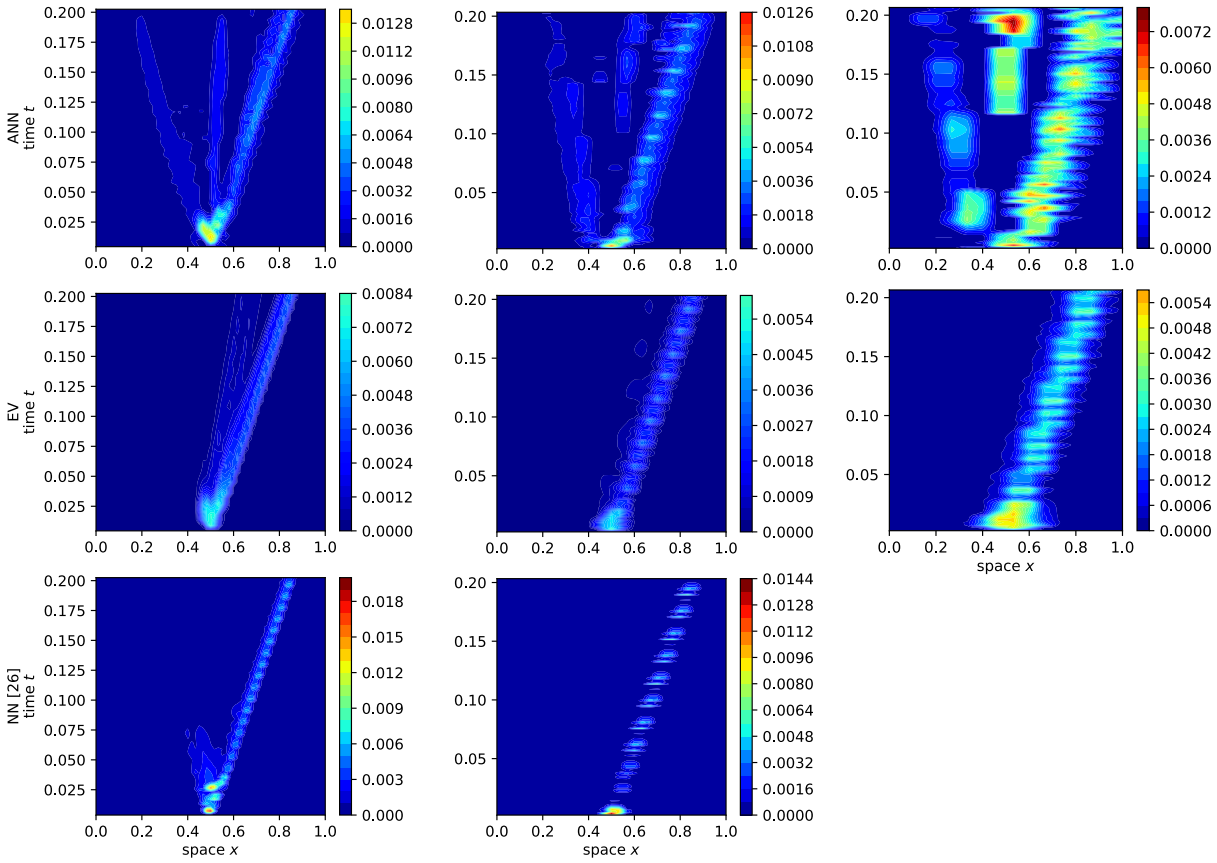

Figure 15: Test case 5: comparison of space-time artificial viscosity for $k = 1, 3, 5$ (from left to right).

	$k=1$			$k=3$			$k=5$	
	NN Ours	EV	NN [26]	NN Ours	EV	NN [26]	NN Ours	EV
ϵ	5.7046e1	5.8845e1	5.9923e1	5.8787e1	7.2200e1	6.5057e1	4.3138e1	5.7406e1
$\nabla\epsilon$	2.7103e1	2.7001e1	2.7639e1	1.3087e2	1.3356e2	1.3892e2	2.0463e2	2.1445e2
$[\![\epsilon]\!]$	1.5025e0	1.8547e0	1.8261e0	1.0929e0	1.4114e0	2.3391e0	8.7019e-1	1.0149e0
o/u	3.0200e-1	7.5084e-1	6.3882e-1	2.0631e-1	7.5516e-1	1.7252e0	8.7122e-1	1.1302e0
mv	1.0951e-8	5.0988e-8	1.7928e-8	2.3772e-7	3.2462e-6	2.9315e-6	1.0951e-8	5.0988e-8

Table 4: Test case 5: comparison of cumulative L^1 error metrics Eq. (17) - (20) over all timesteps from 0 to T.

	$k=1$			$k=3$			$k=5$	
	NN Ours	EV	NN [26]	NN Ours	EV	NN [26]	NN Ours	EV
ϵ	2.6293e3	3.0574e3	3.6781e3	2.2953e3	4.0510e3	3.1186e3	3.7385e3	5.5862e3
$\nabla\epsilon$	1.7160e3	1.8731e3	2.0671e3	5.6320e3	7.8568e3	6.1255e3	1.3576e4	1.6539e4
$[\![\epsilon]\!]$	1.2497e2	1.0530e2	8.9058e1	9.8602e1	7.1604e1	7.2939e1	5.2003e1	5.4259e1
o/u	3.1512e0	3.9106e0	1.0716e0	2.8027e0	5.7163e0	2.5512e0	3.8023e0	4.6424e0
mv	4.2955e-7	2.2112e-6	1.0787e-6	8.0499e-5	2.2317e-4	1.1017e-4	7.3125e-7	4.5271e-6

Table 5: Test case 6: comparison of cumulative L^1 error metrics Eq. (17) - (20) over all timesteps from 0 to T.

the NN and the EV model aim to do: the EV model stabilizes only the shock, meanwhile the NN identifies and tries to stabilize all three present waves. This is counterbalanced by injecting a smaller dissipation near the shock. For $k = 5$, our model injects a large amount of viscosity near the contact discontinuity, which does not make physical sense. However, it is hard to identify the precise reasons why the model introduces such a large amount of artificial viscosity. Indeed, one of the limitations of our *hybrid* approach is that the model learns the viscosity from the data without explicit guidance on what constitutes correct or incorrect behavior in specific contexts. Despite being more intrusive, our NN outperforms the EV model and the NN of [16] under all the considered metrics, as shown in Table 4, where we report the error metrics in time and their cumulative value.

5.2.4 Test case 6: Shu-Osher problem

The second Euler problem we consider is the Shu-Osher problem [56]. It deals with a shock front moving inside a one-dimensional inviscid flow with artificial sinusoidal density fluctuations. This test case is relevant because it benchmarks the ability of the solver to represent both shocks and physical oscillations created

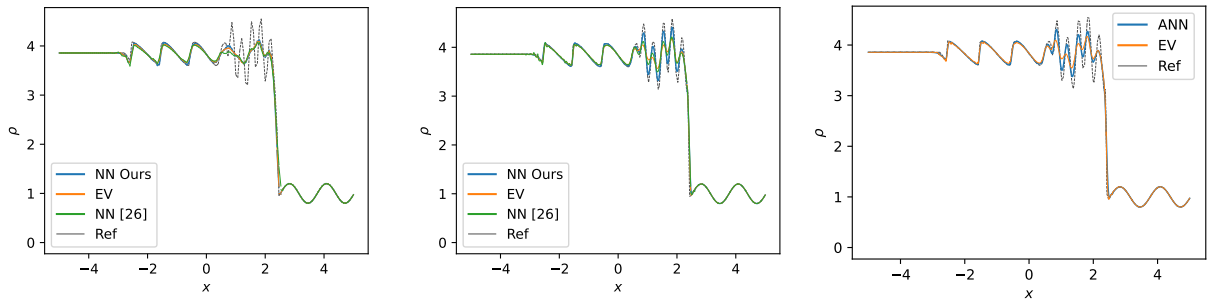

Figure 16: Test case 6: density ρ at $T = 1.8$ for $k = 1, 3, 5$ (from left to right).

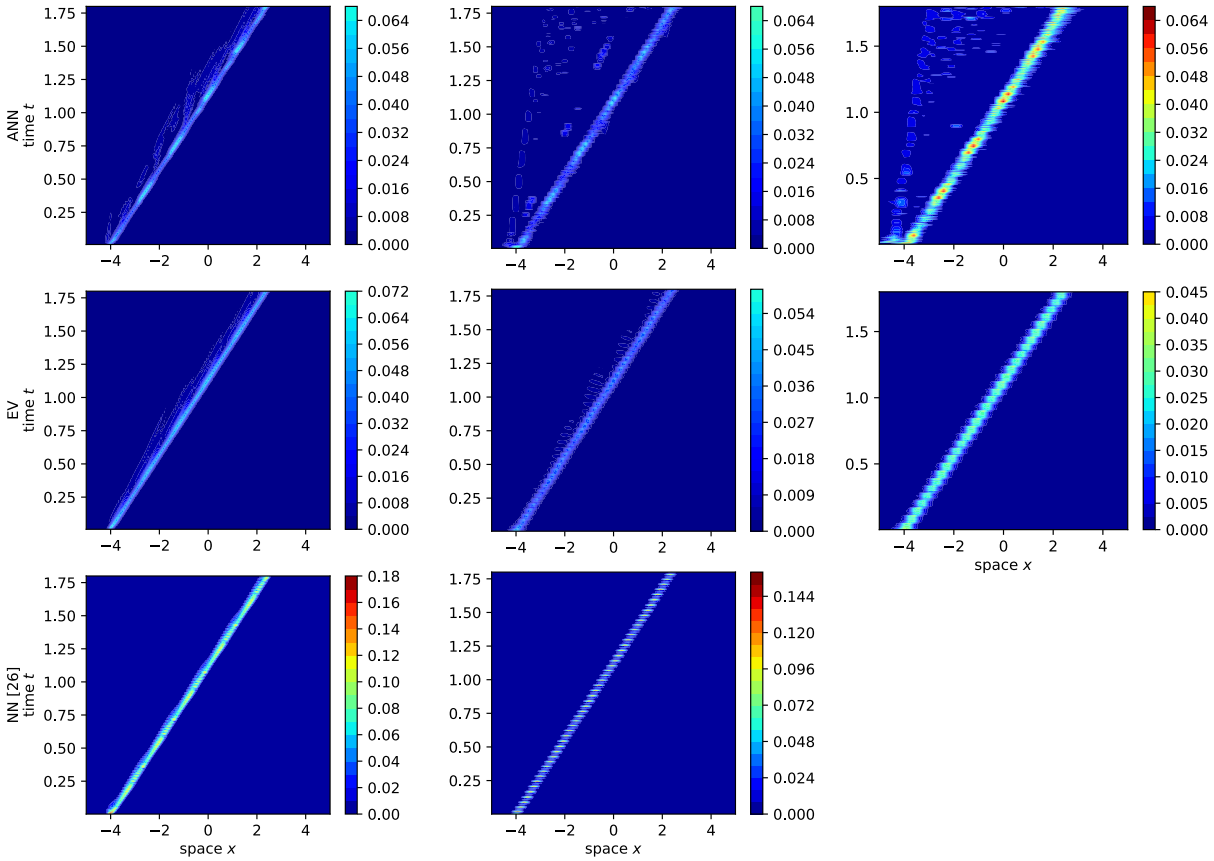

Figure 17: Test case 6: comparison of space-time artificial viscosity for $k = 1, 3, 5$ (from left to right).

by a union of smooth and discontinuous initial data. Namely, we consider the flux of Eq. (8), $\Omega = (-5, 5)$, $T = 1.8$ and initial condition

$$\boldsymbol{u}_0(x) = \begin{bmatrix} \rho_0 \\ v_0 \\ p_0 \end{bmatrix} = \begin{cases} [3.857143, 2.629369, 10.333333]^\top & x \in (-5, -4], \\ [1 + \frac{1}{2}\sin(5x), 0, 1]^\top & x \in (-4, 5). \end{cases}$$

The problem is completed with Dirichlet boundary conditions on the left and Neumann on the right part of the boundary (cfr. [56]). Since the solution exhibits a high frequency wave we employ a finer discretization, namely $h = \frac{1}{150}, \frac{1}{75}, \frac{1}{50}$ with $k = 1, 3, 5$, and CFL $= 0.12, 0.3, 0.4$, respectively. The Courant number is $C = 0.2707, 0.1337, 0.0762$. From Figure 16, we can see that for the case $k = 1$ the NN and the EV model and the NN of [16] produce a rather similar solution. However, the quantitative analysis reported in Table 5 shows that the neural viscosity model is slightly more accurate in the L^1 norm, and when considering overshoots and undershoots. Instead, the cases $k = 3, 5$ show that the EV model and the NN of [16] smooths the solution too much, and thus it is not able to capture the high frequencies. On the other hand, our NN shows much better results, namely the solution correctly represents the high-frequency wave pack present between $x = 0.5$ and $x = 2.5$. This also proves that the neural viscosity model is able to exploit the high-order nature of the method, indeed the three considered cases have the same number of degrees of freedom. In particular, for $k = 3$ we have a reduction of the L^1 error of 43% and of 51% of overshoot/undershoot. The dissipation injected by the NN, as shown in Figure 17, is concentrated on the shock and, differently

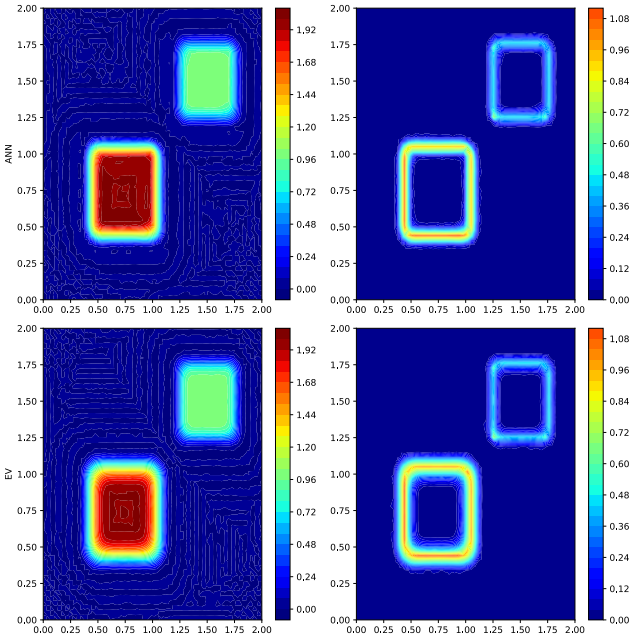

Figure 18: Test case 7: solution and error at final time $T = 0.25$ for $k = 1$.

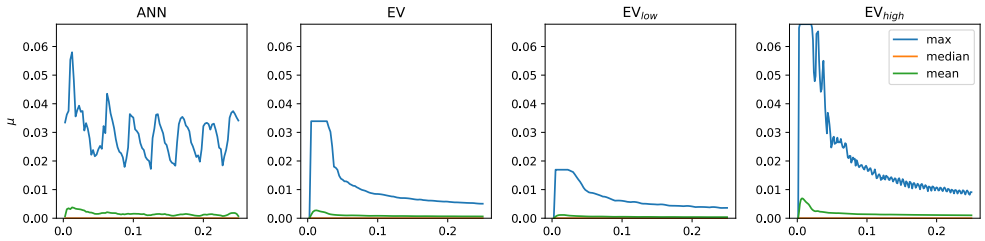

Figure 19: Test case 7: maximum value of the artificial viscosity μ in time for $k = 1$.

from the EV model, adds also other small patches of viscosity at the boundary of the sinusoidal wave pack.

5.3 Two-dimensional problems

We analyze now two-dimensional problems. Notice that, to pass to 2D, we do not have to change anything about the training algorithm or the neural network architecture: everything is exactly the same as in the one-dimensional case. However, the physics that governs these problems is significantly different from before. This requires to train a new neural network, which in turn entails a re-tuning of the hyperparameters. Indeed, in two dimensions more complex structures appear as a result of the interactions among one-dimensional waves. Hence, the differences among the models are more prominent, requiring an NN able to generalize better.

We choose the hyperparameter with a procedure that exactly mimics what was done in the one-dimensional case, as described in Subsection 4.3. Let us remark that this expensive operation of training and hyperparameters tuning must be done just once for all the two-dimensional problems, meanwhile, traditional models must be tuned problem by problem. At the end of this process, we selected a NN with a width of 160 and a depth of eight (the other hyperparameters remain unchanged). As we have done for the one-dimensional test cases, we start by showcasing the results on scalar problems and then pass to the Euler system of equations.

5.3.1 Test case 7: linear advection

The first 2D test case we consider is linear advection $\boldsymbol{f}(u) = \boldsymbol{\beta} u$ with coefficient $\boldsymbol{\beta} = [1, 1]^\top$. This simple test case is non-trivial when dealing with discontinuous initial condition, namely, we employ

$$u_0(x_1, x_2) = \begin{cases} 2 & x_i \in (\frac{1}{5}, \frac{4}{5}),\ i \in \{1, 2\}, \\ 1 & x_i \in (1, \frac{3}{2}),\ i \in \{1, 2\}, \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, it gives us the possibility to check the mass conservation properties of the neural viscosity model. We solve the problem in $\Omega = (0, 2)^2$ with a final time $T = 0.25$ on a structured grid with $7200, 1800, 450$

	$k=1$		$k=3$		$k=5$	
	NN	EV	NN	EV	NN	EV
ϵ	1.0374e+05	1.2616e+05	4.7333e+04	6.6665e+04	7.3585e+04	8.7204e+04
$\nabla\epsilon$	1.0383e+05	1.0916e+05	3.0168e+05	3.4178e+05	8.6089e+05	8.7469e+05
$[\![\epsilon]\!]$	9.5542e+03	1.0781e+04	4.0133e+03	4.7451e+03	2.4605e+03	2.3712e+03
o/u	8.9123e+02	1.6955e+03	2.0373e+03	1.1930e+03	5.0116e+02	6.5465e+02
mv	1.2287e-15	1.3516e-14	8.9032e-14	9.8162e-13	2.3507e-13	2.5857e-13

Table 6: Test case 7: comparison of cumulative L^1 error metrics Eq. (17) - (21) over all timesteps from 0 to T.

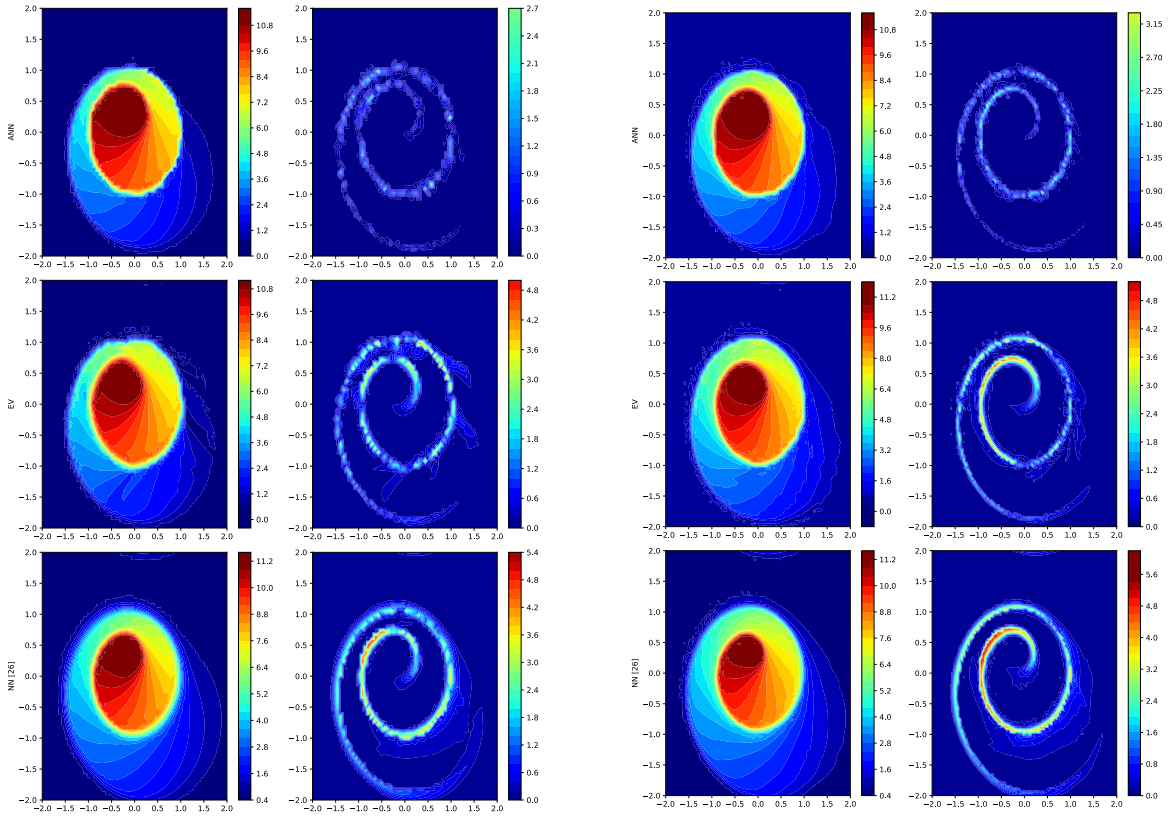

Figure 20: Test case 8: solution and error at final time $T = 0.25$ for $k = 1, 3$.

triangular elements and $k = 1, 3, 5$, CFL $= 0.075, 0.3, 0.32$, respectively. The Courant number is $C = 0.1061, 0.0471, 0.0181$. We focus on $k = 1$ since the other two cases are analogous. As done in the previous test cases, we compare the neural viscosity model with a tuned EV model ($c_K = 1.0, c_{\max} = 0.5$) and two other EV models that inject a sub-optimal amount of dissipation (too large $c_K = 2.0, c_{\max} = 1.0$ and too small $c_K = 0.5, c_{\max} = 0.25$). From Figure 18 we clearly see that both the NN and the EV model smooth the discontinuous profile of the initial condition in order to avoid oscillations. The NN solution is symmetric and visually exhibits a sharper wavefront. This is confirmed by the quantitative analysis of the error reported in Table 6. In particular, it is interesting to observe that the NN model injects more viscosity than a classical EV model but has a smaller L^1 error. This is not an intuitive behavior when comes to the EV model since from both Figure 19 and Table 6 it is clearly observable that EV models that inject larger dissipation have smaller overshoots/undershoots but larger L^1 error. Moreover, the solution obtained with the NN conserves the mass better than the one obtained with the EV model, showing that also physics is respected. Therefore, in this case, the neural viscosity model is effectively learning a new model that has better properties than the EV one.

5.3.2 Test case 8: KPP rotating wave

We consider now nonlinear scalar conservation equations characterized by the non-convex flux $[\sin u, \cos u]^\top$. Specifically, we examine a two-dimensional scalar conservation equation introduced in [24, 35]. This particular test poses a challenge to numerous high-order numerical schemes due to its intricate two-dimensional composite wave structure in the solution. Namely, we have $\Omega = (-2, 2)^2$, $T = 1$ and initial condition

$$u_0(\boldsymbol{x}) = \begin{cases} \frac{7}{2}\pi & \text{if } \|\boldsymbol{x}\|_2 < 1, \\ \frac{1}{4}\pi & \text{otherwise.} \end{cases}$$

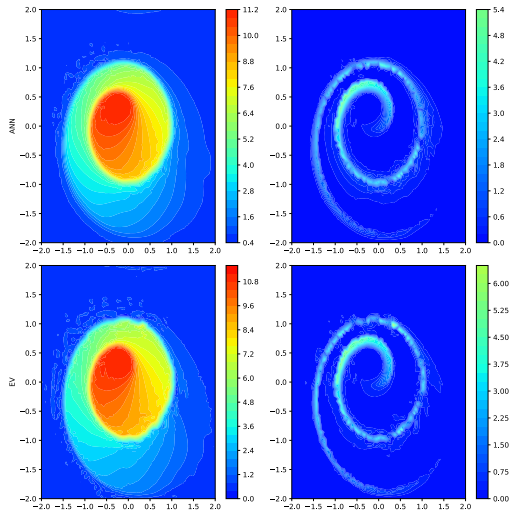

Figure 21: Test case 8: solution and error at final time $T = 0.25$ for $k = 5$.

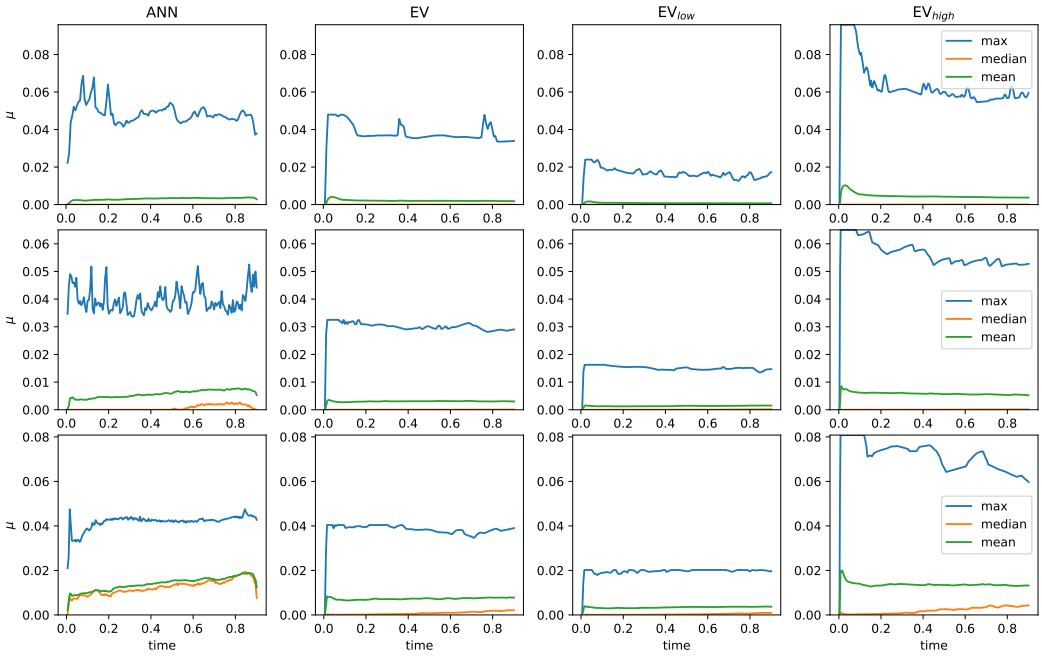

Figure 22: Test case 8: maximum value of the artificial viscosity μ in time for $k = 1, 3, 5$ (from top to bottom).

The problem is endowed with periodic boundary conditions. For discretization, we employ a structured triangular mesh with $7200, 1800, 450$ elements and $k = 1, 3, 5$, CFL $= 0.11, 0.4, 0.48$, respectively. The Courant number is $C = 0.11, 0.0444, 0.0192$. We compare the neural viscosity model with EV models tuned like in the previous test case and the NN of [16]. The results reported in Figure 20 and show that the neural viscosity model produces more accurate solutions with respect to the EV one. In particular, it presents less wriggles and a sharper wave front. To understand the behavior of the model we report also in Figure 22 the maximum, mean, and median of the injected viscosity. As was the case in 1D, the NN model injects more viscosity than the tuned EV model, but retains a lower L^1 error, as shown in Table 7. On the other hand, the method proposed in [16], is the one that produces the smoothest solution, featuring far less oscillations than the other two methods. However, it is also the less accurate solution from the point of view of the L^1 norm.

5.3.3 Test case 9: Euler equation, Riemann problem

We now consider a two-dimensional Riemann problem for the Euler system. Namely, we consider the configuration 12 proposed in [36, 53]. The problem is particularly challenging in view of the presence of both contact waves and shocks. The physical domain is $\Omega = (-\frac{1}{2}, \frac{1}{2})^2$, the final time is $T = 0.25$ and the initial

	$k=1$			$k=3$			$k=5$	
	NN Ours	EV	NN [26]	NN	EV	NN [26]	NN	EV
ϵ	1.6196e5	3.9369e5	6.3515e5	1.6956e5	3.8346e5	5.6881e5	2.5583e5	2.8887e5
$\nabla\epsilon$	2.4551e5	4.3755e5	5.4866e5	1.2625e6	1.7127e6	1.8699e6	2.6152e6	2.8368e6
$[\![\epsilon]\!]$	1.0973e5	1.4006e5	9.5575e4	8.2736e4	9.7502e4	6.4618e4	4.4259e4	6.2624e4
o/u	8.0091e3	1.9289e4	4.8742e3	6.4779e3	6.8667e3	2.5480e3	4.6155e3	5.0963e3

Table 7: Test case 8: comparison of cumulative L^1 error metrics Eq. (17) - (20) over all timesteps from 0 to T.

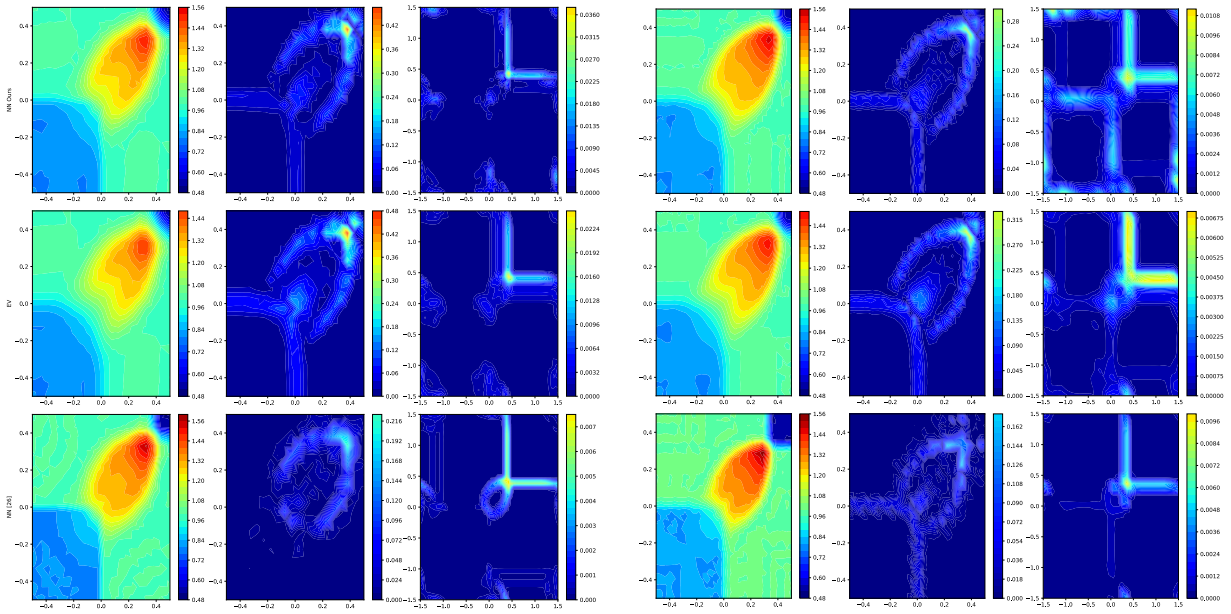

Figure 23: Test case 9: solution and error at final time $T = 0.25$ for $k = 1, 3$ (from left to right).

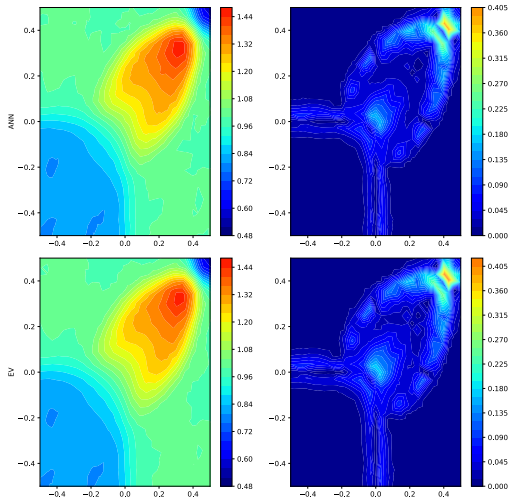

Figure 24: Test case 9: solution and error at final time $T = 0.25$ for $k = 5$.

conditions are:

$$\boldsymbol{u}_0(x_1, x_2) = \begin{bmatrix} \rho_0 \\ v_{1,0} \\ v_{2,0} \\ e_0 \end{bmatrix} = \begin{cases} [0.5313, 0, 0, 0.4]^\top & \text{if } x_1 > 0, x_2 > 0, \\ [1, 0.7276, 0, 1]^\top & \text{if } x_1 < 0, x_2 > 0, \\ [0.8, 0, 0, 1]^\top & \text{if } x_1 < 0, x_2 < 0, \\ [1, 0, 0.7276, 1]^\top & \text{if } x_1 > 0, x_2 < 0. \end{cases}$$

Since the problem is completed with periodic boundary conditions, we enlarge the domain to $\Omega = (-\frac{3}{2}, \frac{3}{2})^2$ to avoid contaminations of the solution. The discretization is done with a structured grid with $7200, 1800, 450$ triangular elements and $k = 1, 3, 5$, CFL $= 0.05, 0.18, 0.21$, respectively. The Courant number is $C = 0.1098, 0.0443, 0.0179$. We compare our NN with a tuned EV model and the NN of [16]. The results are reported in Figure 23. A qualitative way to evaluate the model is to check the error done in representing the fine structures in $\boldsymbol{x} = [0, 0]^\top$ and $\boldsymbol{x} = [0.4, 0.4]^\top$. It is possible to notice that the NN model captures better the contact wave, especially for $k = 3$. This is confirmed by a quantitative analysis of the errors, as shown in Table 8. In this case, the model of [16] slightly outperforms our NN, producing more accurate

	$k=1$			$k=3$			$k=5$	
	NN Ours	EV	NN [26]	NN Ours	EV	NN [26]	NN Ours	EV
ϵ	1.8409e4	3.1180e4	1.4657e4	1.2387e4	2.1667e4	1.1576e4	1.1578e4	1.7845e4
$\nabla\epsilon$	2.0193e4	2.9233e4	1.6856e4	7.0429e4	9.1563e4	6.6923e4	1.3498e5	1.6098e5
$[\![\epsilon]\!]$	5.5618e3	4.5949e3	7.2912e3	2.3759e3	1.7313e3	4.4055e3	1.4495e3	9.8379e2
o/u	7.4932e1	1.7049e2	6.5821e2	6.5138e1	1.3598e2	5.0583e2	3.9249e1	7.5978e1
mv	5.911e-14	5.570e-14	5.656e-14	6.366e-14	6.480e-14	4.768e-14	7.731e-14	7.162e-14

Table 8: Test case 9: comparison of cumulative L^1 error metrics Eq. (17) - (20) over all timesteps from 0 to T.

results under several metrics. Our NN model injects viscosity greater than 10^{-8} in 3730, 1023, and 316 cells for $k = 1, 3, 5$, respectively.

6 Conclusions

In this work, we proposed a novel algorithm to train a NN surrogating an artificial viscosity model in a non-supervised way. Seeking to surpass the performance of classical viscosity models, we introduced a new approach inspired by RL that, by interacting with a DG solver, is able to discover new viscosity models without the necessity of explicitly building a dataset.

First, we presented a NN model with a renewed architecture with respect to the one present in literature [16]. The modification we have introduced proved to increase the flexibility and accuracy of the network. Then, we tested the model trained with the proposed training algorithm on several test cases, ranging from problems with smooth to discontinuous solutions and from scalar to vectorial equations. The trained NN exhibits favorable generalization properties despite being trained in an environment consisting of relatively simple problems. The resulting model proves to maintain the expected order of convergence for smooth problems. Furthermore, it outperforms the optimally tuned classical models under the considered metrics. Indeed, we have demonstrated that a viscosity model with consolidated reliability, such as EV, may falter in generating accurate results unless optimal parameters are carefully selected. The trained model exhibits effective shock-capturing properties and identifies areas requiring dissipation of numerical oscillations. This feature is more apparent in two-dimensional problems, where the NN can accurately capture multidimensional waves, fine patterns, and intricate spatial configurations.

This algorithm also opens the possibility of training models by incorporating data that comes from the physical measurement of quantities of interest, such as the velocity or the pressure of a flow field. In future works, we aim to test that this integration effectively enhances model reliability by incorporating fundamental principles and constraints, leading to more accurate predictions and improved generalization to real-world scenarios. For instance, integrating in the loss a regularization term based on the entropy production could potentially improve the model's adherence to physical laws. Moreover, we plan to apply the proposed algorithm to more domain-specific test cases such as the variational multiscale stabilization methods for the approximation of the incompressible Navier-Stokes equations.

Declaration of competing interests. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding. M.C., P.F.A. and L.D. are members of the INdAM Research group GNCS. The authors have been partially funded by the research projects PRIN 2020 (n. 20204LN5N5) funded by Italian Ministry of University and Research (MUR). P.F.A. is partially funded by the European Union (ERC SyG, NEMESIS, project number 101115663). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. L.D. acknowledges the support by the FAIR (Future Artificial Intelligence Research) project, funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, investment 1.3, line on Artificial Intelligence), Italy. The present research is part of the activities of "Dipartimento di Eccellenza 2023-2027", MUR, Italy.

CRediT authorship contribution statement. P.F.A. and L.D.: Conceptualization, Methodology, Resources, Review and editing, Supervision, Project administration, Funding acquisition. M.C.: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Visualization, Original draft.

References

[1] H. Abbassi, F. Mashayek, and G. B. Jacobs. Shock capturing with entropy-based artificial viscosity for staggered grid discontinuous spectral element method. *Computers & Fluids*, 98:152–163, 2014.

[2] P. F. Antonietti, A. Cangiani, J. Collis, Z. Dong, E. H. Georgoulis, S. Giani, and P. Houston. Review of discontinuous Galerkin finite element methods for partial differential equations on complicated domains. *Building bridges: connections and challenges in modern approaches to numerical partial differential equations*, pages 281–310, 2016.

[3] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM Journal on Numerical Analysis*, 39(5):1749–1779, 2002.

[4] B. Ayuso and L. D. Marini. Discontinuous Galerkin methods for advection-diffusion-reaction problems. *SIAM Journal on Numerical Analysis*, 47(2):1391–1420, 2009.

[5] C. Bardos, A.-Y. LeRoux, and J.-C. Nédélec. First order quasilinear equations with boundary conditions. *Communications in Partial Differential Equations*, 4(9):1017–1034, 1979.

[6] G. E. Barter and D. L. Darmofal. Shock capturing with PDE-based artificial viscosity for DGFEM: Part I. formulation. *Journal of Computational Physics*, 229(5):1810–1827, 2010.

[7] F. Bassi, A. Crivellini, A. Ghidoni, and S. Rebay. High-order discontinuous Galerkin discretization of transonic turbulent flows. In *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, page 180, 2009.

[8] F. Brezzi, L. D. Marini, and E. Süli. Discontinuous Galerkin methods for first-order hyperbolic problems. *Mathematical Models and Methods in Applied Sciences*, 14(12):1893–1903, 2004.

[9] A. Burbeau, P. Sagaut, and C.-H. Bruneau. A problem-independent limiter for high-order Runge–Kutta discontinuous Galerkin methods. *Journal of Computational Physics*, 169(1):111–150, 2001.

[10] M. Caldana, P. F. Antonietti, and L. Dede'. A deep learning algorithm to accelerate algebraic multigrid methods in finite element solvers of 3D elliptic PDEs. *arXiv preprint arXiv:2304.10832*, 2023.

[11] B. Cockburn, G. E. Karniadakis, and C.-W. Shu. *Discontinuous Galerkin methods: theory, computation and applications*, volume 11. Springer Science & Business Media, 2012.

[12] B. Cockburn, S.-Y. Lin, and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: One-dimensional systems. *Journal of Computational Physics*, 84(1):90–113, 1989.

[13] B. Cockburn and C.-W. Shu. The Runge–Kutta discontinuous Galerkin method for conservation laws v: multidimensional systems. *Journal of Computational Physics*, 141(2):199–224, 1998.

[14] C. M. Dafermos and C. M. Dafermos. *Hyperbolic conservation laws in continuum physics*, volume 3. Springer, 2005.

[15] D. A. Di Pietro and A. Ern. *Mathematical aspects of discontinuous Galerkin methods*, volume 69. Springer Science & Business Media, 2011.

[16] N. Discacciati, J. S. Hesthaven, and D. Ray. Controlling oscillations in high-order discontinuous Galerkin schemes using artificial viscosity tuned by neural networks. *Journal of Computational Physics*, 409:109304, 2020.

[17] M. Dumbser and M. Käser. An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes—II. The three-dimensional isotropic case. *Geophysical Journal International*, 167(1):319–336, 2006.

[18] M. Dumbser, O. Zanotti, R. Loubère, and S. Diot. A posteriori subcell limiting of the discontinuous Galerkin finite element method for hyperbolic conservation laws. *Journal of Computational Physics*, 278:47–75, 2014.

[19] D. R. Durran. *Numerical methods for fluid dynamics: With applications to geophysics*, volume 32. Springer Science & Business Media, 2010.

[20] M. Eichinger, A. Heinlein, and A. Klawonn. Stationary flow predictions using convolutional neural networks. In *Numerical Mathematics and Advanced Applications ENUMATH 2019: European Conference, Egmond aan Zee, The Netherlands, September 30-October 4*, pages 541–549. Springer, 2020.

[21] S. Fresca, L. Dede', and A. Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *Journal of Scientific Computing*, 87:1–36, 2021.

[22] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT press, 2016.

[23] D. Gottlieb and C.-W. Shu. On the Gibbs phenomenon and its resolution. *SIAM Review*, 39(4):644–668, 1997.

[24] J.-L. Guermond, R. Pasquetti, and B. Popov. Entropy viscosity method for nonlinear conservation laws. *Journal of Computational Physics*, 230(11):4248–4267, 2011.

[25] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[26] Z. Hao, S. Liu, Y. Zhang, C. Ying, Y. Feng, H. Su, and J. Zhu. Physics-informed machine learning: A survey on problems, methods and applications. *arXiv preprint arXiv:2211.08064*, 2022.

[27] R. Hartmann. Adaptive discontinuous Galerkin methods with shock-capturing for the compressible Navier–Stokes equations. *International Journal for Numerical Methods in Fluids*, 51(9-10):1131–1156, 2006.

[28] A. Heinlein, A. Klawonn, M. Lanser, and J. Weber. Combining machine learning and domain decomposition methods for the solution of partial differential equations—a review. *GAMM-Mitteilungen*, 44(1):e202100001, 2021.

[29] D. Hendrycks and K. Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

[30] J. S. Hesthaven. *Numerical methods for conservation laws: From analysis to algorithms*. SIAM, 2017.

[31] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

[32] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[33] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven. Nodal discontinuous Galerkin methods on graphics processors. *Journal of Computational Physics*, 228(21):7863–7882, 2009.

[34] A. Klöckner, T. Warburton, and J. S. Hesthaven. Viscous shock capturing in a time-explicit discontinuous Galerkin method. *Mathematical Modelling of Natural Phenomena*, 6(3):57–83, 2011.

[35] A. Kurganov, G. Petrova, and B. Popov. Adaptive semidiscrete central-upwind schemes for nonconvex hyperbolic conservation laws. *SIAM Journal on Scientific Computing*, 29(6):2381–2401, 2007.

[36] A. Kurganov and E. Tadmor. Solution of two-dimensional Riemann problems for gas dynamics without Riemann problem solvers. *Numerical Methods for Partial Differential Equations: An International Journal*, 18(5):584–608, 2002.

[37] D. Kuzmin. A vertex-based hierarchical slope limiter for p-adaptive discontinuous Galerkin methods. *Journal of Computational and Applied Mathematics*, 233(12):3077–3085, 2010.

[38] B. Landmann, M. Kessler, S. Wagner, and E. Krämer. A parallel, high-order discontinuous Galerkin code for laminar and turbulent flows. *Computers & Fluids*, 37(4):427–438, 2008.

[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[40] I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[41] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[42] M. Lutter, J. Silberbauer, J. Watson, and J. Peters. Differentiable physics models for real-world offline model-based reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4163–4170. IEEE, 2021.

[43] A. Mani, J. Larsson, and P. Moin. Suitability of artificial bulk viscosity for large-eddy simulation of turbulent flows with shocks. *Journal of Computational Physics*, 228(19):7368–7374, 2009.

[44] G. Novati, H. L. de Laroussilhe, and P. Koumoutsakos. Automating turbulence modelling by multi-agent reinforcement learning. *Nature Machine Intelligence*, 3(1):87–96, 2021.

[45] G. Orlando. A filtering monotonization approach for DG discretizations of hyperbolic problems. *Computers & Mathematics with Applications*, 129:113–125, 2023.

[46] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in Pytorch. *NIPS 2017 Workshop on Autodiff*, 2017.

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[48] P.-O. Persson and J. Peraire. Sub-cell shock capturing for discontinuous Galerkin methods. In *44th AIAA Aerospace Sciences Meeting and Exhibit*, page 112, 2006.

[49] J. Qiu and C.-W. Shu. Hermite WENO schemes and their application as limiters for Runge–Kutta discontinuous Galerkin method: one-dimensional case. *Journal of Computational Physics*, 193(1):115–135, 2004.

[50] J. Qiu and C.-W. Shu. Runge–Kutta discontinuous Galerkin method using WENO limiters. *SIAM Journal on Scientific Computing*, 26(3):907–929, 2005.

[51] D. Ray and J. S. Hesthaven. Detecting troubled-cells on two-dimensional unstructured grids using a neural network. *Journal of Computational Physics*, 397:108845, 2019.

[52] V. V. Rusanov. The calculation of the interaction of non-stationary shock waves and obstacles. *USSR Computational Mathematics and Mathematical Physics*, 1(2):304–320, 1962.

[53] C. W. Schulz-Rinne. Classification of the Riemann problem for two-dimensional gas dynamics. *SIAM Journal on Mathematical Analysis*, 24(1):76–88, 1993.

[54] L. Schwander, D. Ray, and J. S. Hesthaven. Controlling oscillations in spectral methods by local artificial viscosity governed by neural networks. *Journal of Computational Physics*, 431:110144, 2021.

[55] C.-W. Shu. High-order finite difference and finite volume WENO schemes and discontinuous Galerkin methods for CFD. *International Journal of Computational Fluid Dynamics*, 17(2):107–118, 2003.

[56] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77(2):439–471, 1988.

[57] G. A. Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1–31, 1978.

[58] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[59] T. Tassi, A. Zingaro, et al. A machine learning approach to enhance the SUPG stabilization method for advection-dominated differential problems. *Mathematics in Engineering*, 5(2):1–26, 2023.

[60] P. Wesseling. *Principles of computational fluid dynamics*, volume 29. Springer Science & Business Media, 2009.

[61] J.-L. Wu, H. Xiao, and E. Paterson. Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework. *Physical Review Fluids*, 3(7):074602, 2018.

[62] J. Yu and J. S. Hesthaven. A study of several artificial viscosity models within the discontinuous Galerkin framework. *Communications in Computational Physics*, 27(5):1309–1343, 2020.

[63] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023.

[64] V. Zingan, J.-L. Guermond, J. Morel, and B. Popov. Implementation of the entropy viscosity method with the discontinuous Galerkin method. *Computer Methods in Applied Mechanics and Engineering*, 253:479–490, 2013.