Eleventh International Conference on Computational Fluid Dynamics (ICCFD11), Barcelona,Spain, July 11-15, 2022

ICCFD11-2022-4202

A Large Time Step Numerical Method for the Euler Equations using Deep Learning

O. Ovadia*, A. Kahana* and E. Turkel*,**
Corresponding author: odedovadia@mail.tau.ac.il

Applied Mathematics Department, Tel Aviv University, Israel.

Abstract: In this work we present a large time step method to numerically approximate the Euler equations using deep learning. Classical explicit numerical methods such as finite difference or finite volume are heavily constrained in their choice of time step by the Courant–Friedrichs–Lewy (CFL) condition. When the condition is violated, solutions can quickly "blow up" and exhibit severe instability. We empirically show that convolutional neural networks (CNNs) are able to accurately approximate the Euler equations and remain stable over time, even in such difficult cases that would otherwise cause instability.

Keywords: Machine learning, deep learning, Euler equations, numerical stability.

1 Introduction

We present a large time-step numerical method to solve the one-dimensional Euler equations using deep learning (DL). The Euler system of partial differential equations (PDEs) is one of the most well-studied equations in the field of fluid dynamics. In this paper, we focus on the one-dimensional compressible case, which describes the conservation of mass, balance of momentum, and balance of energy for an inviscid flow [1, 2]. The Euler equations are a special case of the more general Navier-Stokes equations that are applicable in practically all fluid dynamics applications [3].

It is often highly impractical or even impossible to find an exact solution for the Euler equations, except for some well known cases such as Sod's problem [4]. As a result, it is far more common to solve them numerically, by finding an accurate approximation of the analytical solution. There is a plethora of different numerical methods to model the Euler equations, including finite differences and finite volumes methods. However, standard numerical methods often face difficulties due to the non-linear behavior of the equations. Analytical solutions often involve discontinuous shock waves that are hard to model accurately. Moreover, even smooth initial conditions can induce shocks [5] which increases the difficulty of the problem. Hence, the Euler equations require special numerical methods, that are suited to these problems.

Most numerical methods are grid-based [6, 7]. It means that the approximation of the physical problem is performed on a discrete set of points in space and time. For increased accuracy it is vital to refine the spatial grid and use a large amount of nodes. This is often the case for the Euler equations with sharp discontinuities or high wave speeds. However, refining the spatial grid without properly modifying the temporal one could cause the solution of the numerical scheme to become unstable, and "blow up" after a few iterations. One necessary condition to avoid that phenomenon, is the Courant–Friedrichs–Lewy (CFL) condition [8], which is discussed in detail in this paper. This condition adds a bound on the choice of time-step in relation to the spatial grid. Consequently, when using a very refined spatial grid it is necessary to increase the number of time-steps as well to keep the scheme stable [9].

In recent years, the usage of machine learning (ML) and especially deep learning (DL) for PDEs has become increasingly common. Powered by advances in hardware and computing capabilities, it has become feasible to train models on large datasets, a task that was incredibly difficult before. In particular, the new idea of Physics-Informed Neural Networks (PINNs) was proposed in [10], and since then evolved into

an area of research of its own. The main idea is to convert the PDE problem from a numerical one to a data-driven one, and then combine DL architectures and physical domain knowledge to create powerful PDE approximators. Standard PINNs have been applied to the Euler equations before. Some examples include forward and inverse modelling [11], supersonic flows [12], and for conservation laws in general [13, 14]. Recently, approximating non-linear operators using DeepONets [15] has also been shown to be applicable to the Euler equations [16].

In this work, we utilize the recent advances in the study of PDEs and DL to build an explicit non-linear numerical solver for the Euler equations that is able to overcome violations of the CFL condition. We generate numerical solutions synthetically from randomly chosen initial conditions using a stable scheme and sub-sample them in such a way that they are unstable (the CFL condition is not met). We proceed by training on this dataset a set of three neural networks corresponding to the three elements of the governing Euler equations. After the training process is done, the model is ready to be used as a numerical solver. While the training part can take a long time, inference is immediate (takes only milliseconds), making this method applicable for real-time scenarios. Similar work has been done before for the wave equation [17], but the transition to a system of non-linear equations is not trivial, due to the behavior of the Euler equations described above.

The outline for the remainder of this paper is as follows. Section 2 discusses the physical problem and classical numerical methods to approximate it. Section 3 reviews key concept of deep learning and describes the model and its training process. Section 4 presents the numerical experiments conducted and the findings using the proposed methods. Concluding remarks are found in section 5.

2 Numerical modelling

2.1 Mathematical model

The Euler equations for compressible flow describe the conservation of mass, momentum, and energy. Their conservative formulation as an initial value problem with periodic boundary conditions in the one-dimensional case of is given by [18]:

$$\begin{cases} \partial_t U + \partial_x F(U) = 0 & x \in \Omega, \quad t \in (0, T] \\ U(x, 0) = U_0(x) & x \in \Omega \\ U(L, t) = U(R, t) & t \in [0, T] \\ \nabla U(L, t) = \nabla U(R, t) & t \in [0, T] \end{cases}, \tag{1}$$

where $\Omega := [L, R]$ is the physical domain and T is the final time. U and F are given by:

$$U = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}, \; F(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ u(E + p) \end{pmatrix}, \tag{2}$$

where ρ is the density, p is the pressure, u is the velocity, and E is the total energy. For an ideal gas we also have a closed form expression for the pressure p:

$$p = (\gamma - 1)(E - \frac{1}{2}\rho u^2) \; , \tag{3}$$

where $\gamma = 1.4$ is the adiabatic index.

2.2 Numerical approximation

One of the most common methods to approximate the Euler equations is the finite volume method. When discretizing the PDE it is often beneficial to use its integral form in order to admit discontinuous solutions. This is given by:

$$\int_{x_1}^{x_2} U(x,t_2)dx = \int_{x_1}^{x_2} U(x,t_1)dx + \int_{t_1}^{t_2} F(U(x_1,t))dt - \int_{t_1}^{t_2} F(U(x_2,t))dt \tag{4}$$

for any $[x_1,x_2]\times[t_1,t_2] \in \Omega \times [0,T]$.

We discretize the spatial domain into N_x finite volumes, also called cells. Denote the i-th cell by I_i. It is defined by:

$$I_i := [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}],\ i = 1...N_x \tag{5}$$

where the cell boundaries $x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}$ and its center x_i are given by:

$$x_{i-\frac{1}{2}} := (i-1)\Delta x, \qquad x_{i+\frac{1}{2}} := i\Delta x, \qquad x_i := (i-\frac{1}{2})\Delta x, \qquad i = 1...N_x, \tag{6}$$

where the cell size is $\Delta x := \frac{|R-L|}{N_x}$.

Similarly, the temporal domain $[0,T]$ is discretised using N_t parts $[t_{n-1},t_n]$, for $n = 1,\dots,N_t$. In the constant time-step size case we have $t_n := n\Delta t$ and the time-step is defined by $\Delta t := \frac{T}{N_t}$. However, in practice many methods use an adaptive step-size approach in time due to the time-varying nature of velocities in non-linear PDEs.

Classical approaches, such as Godunov-type methods [19] define the cell-average as:

$$U_i^n := \frac{1}{\Delta x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} U(x,t_n)dx \tag{7}$$

which means that $U(x,t_n) = U_i^n$, $\forall x \in I_i$. Using these notations, an explicit conservative numerical is given by:

$$U_i^{n+1} = U_i^n + \frac{\Delta t}{\Delta x}[F_{i-\frac{1}{2}} + F_{i+\frac{1}{2}}] \tag{8}$$

where $F_{i\pm\frac{1}{2}}$ is called the numerical flux. There are many ways to calculate the numerical flux, and they correspond to different numerical schemes. Among them are the original Godunov scheme by [19], and later schemes by Osher [20], Roe [21], and Harten-Lax-van Leer (HLL) [22].

2.3 Numerical stability

To ensure stability, one must appropriately choose the discretization parameters Δt and Δx. Usually, Δx is determined by the physical problem, and Δt is chosen in a way that allows the scheme to remain stable. A necessary condition for stability of an explicit scheme is the Courant–Friedrichs–Lewy (CFL) condition [8]:

$$\left|\frac{\Delta t}{\Delta x} S_{max}^n\right| \leq C_{max} \tag{9}$$

where C_{max} is the maximal allowed value for the Courant number $\left|\frac{\Delta t}{\Delta x} S_{max}^n\right|$, and S_{max}^n is the maximal wave speed at time t_n, given by [6]:

$$S_{max}^n := \max_i\{|u_i^n| + a_i^n\}$$

where the speed of sound a for an ideal gas is $a^2 = \frac{\gamma p}{\rho}$.

Note, that due to the non-linearity of the Euler equations, the wave speed varies in time and space. Hence, we take the maximal value in space in each time-step. Furthermore, the threshold C_{max} is determined by the choice of numerical scheme. For the scope of this paper, we use $C_{max} = 1$, though for practical purposes the more strict choice of $C_{max} = 0.9$ is often used.

2.4 Data-driven formulation

The numerical problem can also be formulated as a supervised machine learning problem. In supervised learning, each sample has a label and a chosen learning algorithm learns the connection between each data sample and its label. In our case, both the samples and the labels are defined using solutions of the Euler equations.

We create a dataset of accurate solutions of the Euler equations. We achieve this by choosing discretization parameters for which the CFL condition holds. Then we pick a set of initial conditions and numerically evolve each one of them until we reach the final time T using an accurate numerical scheme. Then, we subsample each numerical solution in time by a constant factor, which amounts to increasing Δt. Consequently, we get the same set of solutions, only with a coarser temporal discretization. We choose a subsampling rate for which the CFL condition does not hold. This is our final generated dataset. More details about the creation of the dataset can be found in the results section 4.

We can then proceed to describe the numerical problem as a data-driven problem. Let U^n be the numerical solution obtained in the dataset creation process for some initial condition at time $t_n < T$. Then U^n can be written as follows:

$$U^n = \begin{pmatrix} \rho^n \\ (\rho u)^n \\ E^n \end{pmatrix} = \begin{pmatrix} \rho(x_1,t_n) & \rho(x_2,t_n) & \dots & \rho(x_{N_x},t_n) \\ \rho u(x_1,t_n) & \rho u(x_2,t_n) & \dots & \rho u(x_{N_x},t_n) \\ E(x_1,t_n) & E(x_2,t_n) & \dots & E(x_{N_x},t_n) \end{pmatrix}. \tag{10}$$

For each sample U^n our goal is to predict U^{n+1}. Using standard ML notations, we divide the data into samples X and corresponding targets (labels) Y. Let N_{IC} be the number of generated initial conditions. Let $l < N_{IC}$ be the index related to the l-th initial condition. We denote the relevant numerical solution at time $n < N_t$ by:

$$U^{n,l} = \begin{pmatrix} \rho^{n,l} \\ (\rho u)^{n,l} \\ E^{n,l} \end{pmatrix}.$$

Using this notation, we define the mapping between X and Y as pairs of the form $(x_{nl}, y_{n,l})$ where $x_{n,l} = U^{n,l}$ and $y_{n,l} = U^{n+1,l}$. The full mapping is given by:

$$\{(x_{n,l}, y_{n,l}) : \ \forall l = 0 \dots N_{IC} - 1, \ n = 0 \dots N_t - 1\}. \tag{11}$$

Thus, given some $x_{n,l}$ our goal is to predict $y_{n,l}$. The main difficulty is that the Courant number induced by the discretizations of $x_{n,l}$ and $y_{n,l}$ does not satisfy the CFL condition. We overcome this difficulty using deep learning.

3 Deep Learning Approach

We use a deep learning approach to solve the data-driven problem formulated in section 2.4. We use a fully convolutional neural-network [23]. Each layer has trainable parameters (weights and bias terms) that are initialized randomly. All hidden layers are followed by a non-linear activation. To learn these parameters we iterate using the Stochastic Gradient Descent (SGD) algorithm to minimize a predefined target function (loss function). After experimenting with a variety of network architectures by varying the depth, number of convolutions, size of convolution kernel, and more hyper-parameters, the chosen final architecture is described below.

3.1 Network architecture

Convolutions are well known for their ability to capture spatial dependencies in datasets. Since each sample in our dataset is a snapshot of $(\rho, \rho u, E)$ at different spatial locations, convolutions are the natural choice. We use four one-dimensional hidden convolutional layers, followed by one last convolutional output layer. Each hidden layer is composed of 32 trainable convolutional filters of size 5, followed by a non-linear ReLU

activation function. This leads to a very light model with about 16,000 weights. Having a low number of weights is beneficial in terms of computational complexity of the optimization process and inference speed.

Due to the different nature of each component of the equations, we trained three separate neural networks. Each neural network is trained to solve the equations for a different component: density, momentum, and energy. Using one component exclusively for each neural network would not work, since they interact with one another. We address this by using a shared input for all three networks. More specifically, the input to all networks can be described as U^n, while the three separate outputs are ρ^{n+1}, $(\rho u)^{n+1}$, and E^{n+1}. A schematic of the model architecture is presented in figure 1.

3.2 Loss function

In the process of training the neural network, we aim to find the best weights that optimize a certain loss function, depending on the problem at hand. In our case, we have a machine learning regression problem where it is common to use the mean squared error (MSE) metric. We define it using the notation from section 2.4. We write the prediction of the model for $x_{n,l}$ as

$$\hat{y}_{n,l} := \begin{pmatrix} \rho_{pred}^{n+1,l} \\ (\rho u)_{pred}^{n+1,l} \\ E_{pred}^{n+1,l} \end{pmatrix}.$$

Using this, we directly write the overall MSE in terms of $y_{n,l}$ and $\hat{y}_{n,l}$:

$$MSE := \frac{1}{N_{IC} N_t} \sum_{l=0}^{N_{IC}-1} \sum_{n=0}^{N_t-1} \|y_{n,l} - \hat{y}_{n,l}\|_2^2, \tag{12}$$

and we decompose it into the MSE for each variable:

$$\begin{aligned} MSE_\rho &= \frac{1}{N_{IC} N_t} \sum_{l=0}^{N_{IC}-1} \sum_{n=0}^{N_t-1} \left(\rho^{n+1,l} - \rho_{pred}^{n+1,l}\right)^2, \\ MSE_{\rho u} &= \frac{1}{N_{IC} N_t} \sum_{l=0}^{N_{IC}-1} \sum_{n=0}^{N_t-1} \left((\rho u)^{n+1,l} - (\rho u)_{pred}^{n+1,l}\right)^2, \\ MSE_E &= \frac{1}{N_{IC} N_t} \sum_{l=0}^{N_{IC}-1} \sum_{n=0}^{N_t-1} \left(E^{n+1,l} - E_{pred}^{n+1,l}\right)^2. \end{aligned} \tag{13}$$

We train each network according to its relevant loss function.

3.3 Physics-informed simultaneous training

The architecture proposed in section 3.1 combined with the loss in section 3.2 yields three separate networks that learn independently. We seek to improve this method by adding a physics-informed (PI) component that utilizes physical knowledge about the Euler equations. We achieve this by adding a PI loss that enables us to connect the three networks and form one unified model.

3.3.1 Simultaneous training

The most straightforward method to solve the Euler equations as a system with DL is to use a single model. However, after running multiple experiments with different architectures we have seen that a single model that outputs the three variables together does not produce good results. A more fitting solution would be to train the three proposed models together.

In general, training a neural network requires a few well knows steps. We begin by dividing the data into small random batches. For each batch we apply the model on the given samples. Then, we compare the predicted outputs to the ground truth using a predefined loss function (MSE in our case). We continue by computing the gradients of the loss function with respect to the weights of the model. Finally, we apply the

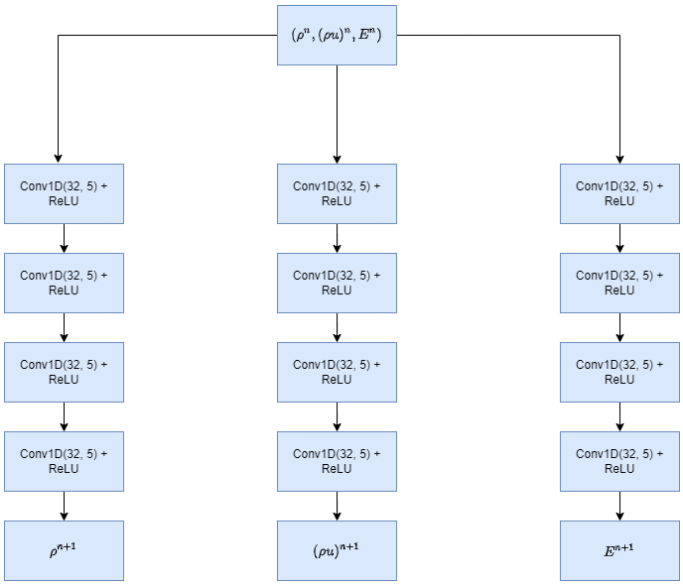

Figure 1: Neural networks architecture.

SGD optimization algorithm on these gradients. A full iteration on the entire data-set (iterating over all the batches), is called an epoch. This process can be repeated for multiple epochs until convergence of the SGD algorithm is reached.

We extend this process to include three separate models. The full flow can be seen in algorithm 1. Denote the combined loss of all three networks by $\mathcal{L}_U$, and their combined weights by W_U. Since the models are independent from one another, we compute their gradients separately. In that case, computing SGD using $\mathcal{L}_U$, w.r.t. W_U has no additional value. Hence, while the models train on the same batch together, they still do not affect one another. So, we need to find a way to connect their gradients. We achieve this by adding a physics-informed (PI) loss function, described in the following subsection.

3.3.2 Physics-informed loss

We create a PI loss term to enhance the performance of the proposed method by adding iterative marching steps into the loss, similar to the ones used to create the dataset. We achieve this by taking the three outputs of the model, concatenating them together, and then applying the model on them again. Finally, we compare the results to the ground truth of the next time-step. We formulate this process mathematically using notation from section 2.4. Let $M_\rho, M_{\rho u}, M_E$ be our three models. For some initial condition index l, and time-step $n < N_t$ we define the model predictions as:

$$\begin{aligned} \rho_{pred}^{n+1,l} &:= M_\rho(U^{n,l}), \\ (\rho u)_{pred}^{n+1,l} &:= M_{\rho u}(U^{n,l}), \\ E_{pred}^{n+1,l} &:= M_E(U^{n,l}), \\ U_{pred}^{n+1,l} &:= \begin{pmatrix} \rho_{pred}^{n+1,l} \\ (\rho u)_{pred}^{n+1,l} \\ E_{pred}^{n+1,l} \end{pmatrix}. \end{aligned} \tag{14}$$

Algorithm 1 Simultaneous training process

Require:

Three models $M_\rho, M_{\rho u}, M_E$ with corresponding weights $W_\rho, W_{\rho u}, W_E$ (initialized randomly).

Loss function $\mathcal{L}$.

Dataset divided into $\{B_n := (X_n, Y_n)\}_{n=1}^{N_{batch}}$ batches of equal size, where Y_n can be decomposed as $Y_n = (\boldsymbol{\rho}_n, (\boldsymbol{\rho u})_n, \boldsymbol{E}_n)$.

while $n \leq N_{batch}$ **do**

Predict:

$\boldsymbol{\rho}_{pred} \leftarrow M_\rho(X_n)$

$(\boldsymbol{\rho u})_{pred} \leftarrow M_{\rho u}(X_n)$

$\boldsymbol{E}_{pred} \leftarrow M_E(X_n)$

Compute loss:

$\mathcal{L}_\rho \leftarrow \mathcal{L}(\boldsymbol{\rho}_{pred}, \boldsymbol{\rho}_n)$

$\mathcal{L}_{\rho u} \leftarrow \mathcal{L}((\boldsymbol{\rho u})_{pred}, (\boldsymbol{\rho u})_n)$

$\mathcal{L}_E \leftarrow \mathcal{L}(\boldsymbol{E}_{pred}, \boldsymbol{E}_n)$

$\mathcal{L}_U \leftarrow \mathcal{L}_\rho + \mathcal{L}_{\rho u} + \mathcal{L}_E$

Compute gradients:

$\nabla_\rho \leftarrow \frac{\partial \mathcal{L}_\rho}{\partial W_\rho}$

$\nabla_{\rho u} \leftarrow \frac{\partial \mathcal{L}_{\rho u}}{\partial W_{\rho u}}$

$\nabla_E \leftarrow \frac{\partial \mathcal{L}_E}{\partial W_E}$

$\nabla_U \leftarrow \frac{\partial \mathcal{L}_U}{\partial W}$, where $W = W_\rho \cup W_{\rho u} \cup W_E$

Apply SGD on gradients

end while

Then, the iterative PI predictions are given by:

$$\begin{aligned} \rho_{PI}^{n+2,l} &:= M_\rho(U_{pred}^{n+1,l}), \\ (\rho u)_{PI}^{n+2,l} &:= M_{\rho u}(U_{pred}^{n+1,l}), \\ E_{PI}^{n+2,l} &:= M_E(U_{pred}^{n+1,l}). \end{aligned} \tag{15}$$

We compare these values to the ground truth at time-step $n+2$ using a regular MSE loss, as in (13):

$$\begin{aligned} PIMSE_\rho &:= \frac{1}{N_{IC} N_t} \sum_{l=0}^{N_{IC}-1} \sum_{n=0}^{N_t-2} (\rho^{n+2,l} - \rho_{PI}^{n+2,l})^2, \\ PIMSE_{\rho u} &:= \frac{1}{N_{IC} N_t} \sum_{l=0}^{N_{IC}-1} \sum_{n=0}^{N_t-2} ((\rho u)^{n+2,l} - (\rho u)_{PI}^{n+2,l})^2, \\ PIMSE_E &:= \frac{1}{N_{IC} N_t} \sum_{l=0}^{N_{IC}-1} \sum_{n=0}^{N_t-2} (E^{n+2,l} - E_{PI}^{n+2,l})^2. \end{aligned} \tag{16}$$

We set the overall loss of the model to be a linear combination of the regular MSEs and PIMSEs. It is possible to give these losses different weights in the overall sum. However, in our case all the losses are roughly of the same order of magnitude, so we choose to average them. It follows directly from these definitions that each PIMSE is dependent on all three models, unlike the regular MSEs. With the addition of the PIMSE loss terms it makes sense to use SGD on $\mathcal{L}_U$ w.r.t. W_U, since the gradients of the models are now connected to one another.

3.4 Evaluation metric

After the model is done training, we need to evaluate its performance. As mentioned above, we are interested in using the model as an iterative numerical scheme. It follows that using a standard MSE alone is not a sufficient metric. Instead, we look at the average relative L^2 error over time for each initial condition.

Let $\{U_n\}_{n=0}^{N_t}$ be an exact solution of the Euler equation. We define an iterative process starting from U_0, similarly to (15). We use U_0 to predict $\hat{U}_1$, and use the predicted value as input to make the next prediction $\hat{U}_2$, and so on. We repeat this iteratively until we reach the final time-step. We then compare the $\{\hat{U}_n\}_{n=1}^{N_t}$ predicted values to the exact $\{U_n\}_{n=1}^{N_t}$ values. Let $\{\hat{\rho^n}\}_{n=0}^{N_t}, \{\rho^n\}_{n=0}^{N_t}$ be the density values corresponding to $\{\hat{U}_n\}_{n=1}^{N_t}$, $\{U_n\}_{n=0}^{N_t}$. Then the average relative L^2 error in after N_t time-steps is defined as:

$$\frac{1}{N_t}\sum_{n=1}^{N_t}\frac{\|\hat{\rho^n}-\rho^n\|_2}{\|\rho^n\|_2} \tag{17}$$

and the errors for ρu and E are defined equivalently. The average error is defined as the average of all three errors.

4 Experiments and results

We tested the proposed method on multiple scenarios with different initial conditions. In each case we generated accurate and stable solutions using $CFL \leq 0.9$ and sub-sampled them in time. As a result, the new coarse temporal discretization did not satisfy the CFL condition, as described in 2.4. All solutions were generated by the Harten-Lax-van Leer-Contact numerical scheme (HLLC) [24] with periodic boundary conditions in the physical domain $\Omega = [-1, 1]$. The solver was implemented in the Python programming language as part of the Clawpack software [25, 26, 27]. In each scenario we train two models: a regular separately trained model, and a simultaneously trained PI model. We use the exact same architecture for both models, described in 3.1. All models were trained for 500 epochs using a batch size of 32. Finally, we compare the results of the two models as described in section 3.4.

4.1 Scenario I

We begin with a simple case where we have smooth solutions. We consider initial conditions of the form

$$\begin{cases}\rho(x,0) = 1 + c\sin(\pi x)\\ \rho u(x,0) = 1 + c\sin(\pi x)\\ E(x,0) = 2.5 + 0.5(1 + c\sin(\pi x))\end{cases} \tag{18}$$

where $c \in [0.1, 0.5]$ is randomly uniformly sampled. We set the discretization parameters to $N_t = 100$, $N_x = 100$, and the final simulation time to $T = 4$. For the HLLC scheme we used a refined temporal discretization with $N_t = 600$. The resulting initial CFL in this case is $CFL \approx 4.5 > 1$.

As presented in figure (2) and table (1), both the non-PI (separate training) model and the PI one perform well here. However, the latter is more accurate by an order of magnitude, though in this simple case the difference is barely noticeable to the human eye.

4.2 Scenario II

We now consider a slightly more difficult case. We still have smooth solutions, but we increase the frequencies of the sines in the initial conditions. This creates a more diverse dataset, given by:

$$\begin{cases}\rho(x,0) = 1 + c_1\sin(c_2\pi x)\\ \rho u(x,0) = 1 + c_1\sin(c_2\pi x)\\ E(x,0) = 2.5 + 0.5(1 + c_1\sin(c_2\pi x))\end{cases}, \tag{19}$$

where $c_1 \sim Uniform(0.1, 0.5)$ and c_2 is randomly sampled from $[1, 2, ..., 6]$. We keep the discretization parameters $N_t = 100$ and $N_x = 100$, but increase the final simulation time to $T = 5$. For the HLLC scheme we used a refined temporal discretization with $N_t = 700$. The resulting initial CFL in this case is $CFL \approx 6 > 1$.

Figure (3) and table 2 show that the difference between the two models is far more significant in this case. The PI model remains close to the exact small time-step solution, while the non-PI model fails to approximate the solution well. This is especially visible near the left boundary.

4.3 Scenario III

In this scenario we also test the capabilities of the proposed method for smooth initial conditions. However, unlike the previous scenarios, we now set different initial conditions for the density and momentum. Now, the solutions exhibit shock behavior at later time-steps. The initial conditions are given by:

$$\begin{cases} \rho(x,0) = 1 + c\sin(2\pi x) \\ \rho u(x,0) = c\sin(4\pi x) \\ E(x,0) = 0.5 + c\cos(2\pi x) \end{cases} \tag{20}$$

where $c \sim Uniform(0.1, 0.2)$. We increase the spatial discretization to $N_x = 800$, and keep $N_t = 100$. We set the final simulation time as $T = 1$. For the HLLC scheme we used a refined temporal discretization with $N_t = 400$. The resulting initial CFL in this case is $CFL \approx 3 > 1$.

The sharp shock behavior is observed in (4). The non-PI model completely fails when trying to approximate the solution. It creates non-physical oscillations while also completely missing the shock locations. However, the PI model is able to accurately approximate the solution, even near the shocks.

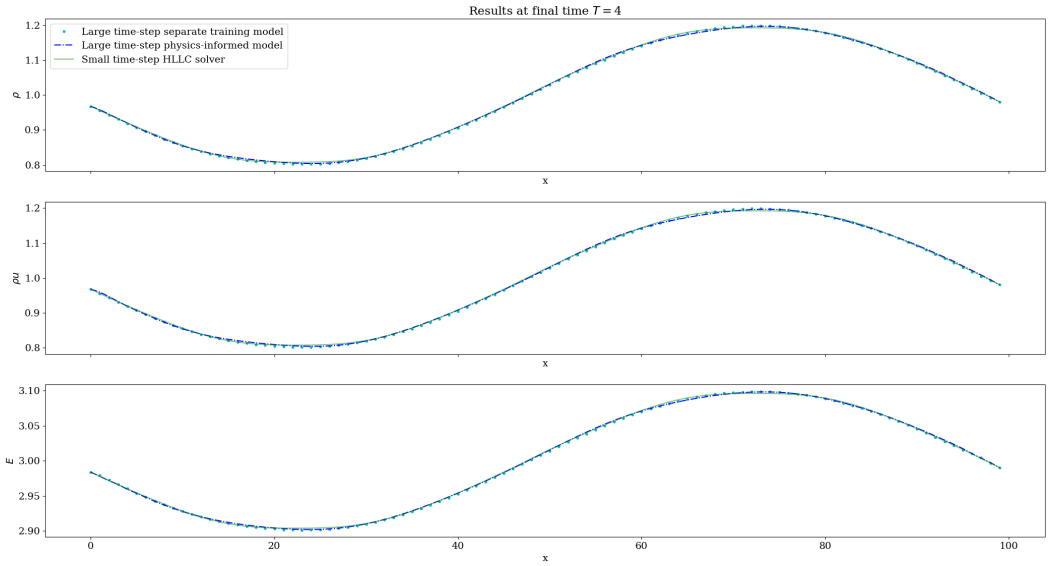

Figure 2: A comparison of the solutions at final time $T = 4$ obtained by the models in scenario 4.1.

	Density error	Velocity error	Energy error	Average error
Non-PI model	7.00348E-03	6.99725E-03	1.16930E-03	5.05668E-03
PI model	1.55723E-03	1.57113E-03	2.53120E-04	1.12716E-03

Table 1: A comparison of relative L^2 errors between the models in scenario 4.1.

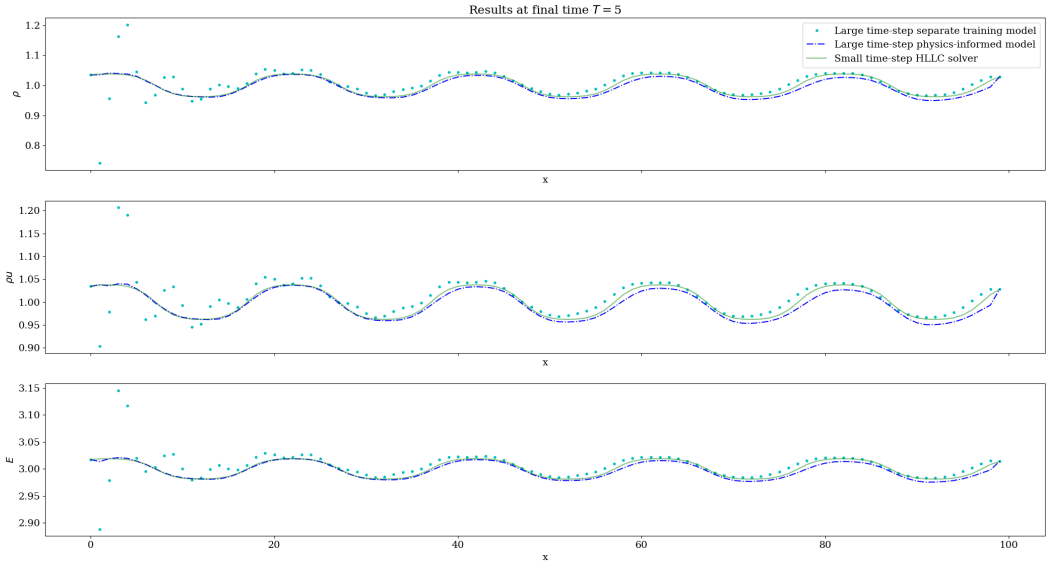

Figure 3: A comparison of the solutions at final time $T = 5$ obtained by the models in scenario 4.2.

	Density error	Velocity error	Energy error	Average error
Non-PI model	6.02502E-02	6.05010E-02	1.14910E-02	4.40807E-02
PI model	6.74203E-03	6.34023E-03	1.05918E-03	4.71381E-03

Table 2: A comparison of relative L^2 errors between the models in scenario 4.2.

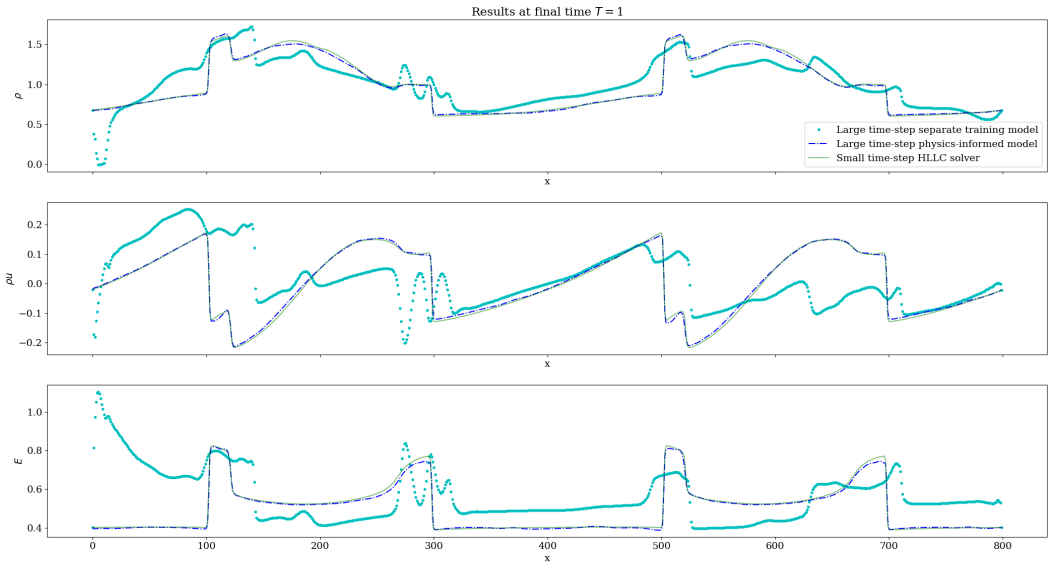

Figure 4: A comparison of the solutions at final time $T = 1$ obtained by the models in scenario 4.3.

	Density error	Velocity error	Energy error	Average error
Non-PI model	2.56729E-01	1.02613E+00	3.37565E-01	5.40141E-01
PI model	1.53220E-02	5.36622E-02	1.46112E-02	2.78651E-02

Table 3: A comparison of relative L^2 errors between the models in scenario 4.3.

5 Conclusions and Future Work

In this work we presented a numerical method to explicitly solve the Euler equations while violating the CFL condition. We achieved this by building a deep learning model composed of three distinct neural networks. We added a physics-informed loss that enabled us to simultaneously train the three connected neural networks. Finally, we tested the model on different scenarios, and saw that the addition of the physics-informed loss greatly increased the accuracy of our method.

We plan on expanding this method to other difficult scenarios that involve strong discontinuities, such as Sod's shock tube [4], the Shu-Osher problem [28], and Woodward-Colella blast wave [29]. Another expansion plan is to reproduce similar results for the two-dimensional case.

References

[1] Richard Courant and Kurt Otto Friedrichs. *Supersonic flow and shock waves*, volume 21. Springer Science & Business Media, 1999.

[2] Hans Wolfgang Liepmann and Anatol Roshko. *Elements of gasdynamics*. Courier Corporation, 2001.

[3] David J Acheson. Elementary fluid dynamics, 1991.

[4] Gary 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.

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

[6] Eleuterio F Toro. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. Springer, 2013.

[7] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.

[8] Richard Courant, K. Friedrichs, and Hans Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100:32–74, 1928.

[9] Peter D Lax and Robert D Richtmyer. Survey of the stability of linear finite difference equations. *Communications on pure and applied mathematics*, 9(2):267–293, 1956.

[10] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[11] Zhiping Mao, Ameya D. Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[12] Ameya D. Jagtap, Zhiping Mao, Nikolaus Adams, and George Em Karniadakis. Physics-informed neural networks for inverse problems in supersonic flows, 2022.

[13] Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5):2002–2041, 2020.

[14] Ameya D Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[15] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[16] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

[17] Oded Ovadia, Adar Kahana, Eli Turkel, and Shai Dekel. Beyond the courant-friedrichs-lewy condition: Numerical methods for the wave problem using deep learning. *Journal of Computational Physics*, 442:110493, 2021.

[18] Courant and Hilbert. Methods of mathematical physics, vol. ii. partial differential equations. *Journal of Applied Mechanics*, 30, 1963.

[19] Sergei K. Godunov and I. Bohachevsky. Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics. *Matematičeskij sbornik*, 47(89)(3):271–306, 1959.

[20] Björn Engquist and S. Osher. One-sided difference approximations for nonlinear conservation laws. *Journal of Computational Physics*, 36:321–351, 1981.

[21] P.L Roe. Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357–372, 1981.

[22] Amiram Harten, Peter D. Lax, and Bram van Leer. On upstream differencing and godunov-type schemes for hyperbolic conservation laws. *SIAM Review*, 25(1):35–61, 1983.

[23] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, Los Alamitos, CA, USA, 2015. IEEE Computer Society.

[24] E. F. Toro, M. Spruce, and W. Speares. Restoration of the contact surface in the HLL-Riemann solver. *Shock Waves*, 4(1):25–34, July 1994.

[25] Clawpack Development Team. Clawpack software, 2020. Version 5.7.1.

[26] Kyle T Mandli, Aron J Ahmadia, Marsha Berger, Donna Calhoun, David L George, Yiannis Hadjimichael, David I Ketcheson, Grady I Lemoine, and Randall J LeVeque. Clawpack: building an open source ecosystem for solving hyperbolic pdes. *PeerJ Computer Science*, 2:e68, 2016.

[27] David I. Ketcheson, Kyle T. Mandli, Aron J. Ahmadia, Amal Alghamdi, Manuel Quezada de Luna, Matteo Parsani, Matthew G. Knepley, and Matthew Emmett. PyClaw: Accessible, Extensible, Scalable Tools for Wave Propagation Problems. *SIAM Journal on Scientific Computing*, 34(4):C210–C231, November 2012.

[28] Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of computational physics*, 77(2):439–471, 1988.

[29] Paul Woodward and Phillip Colella. The numerical simulation of two-dimensional fluid flow with strong shocks. *Journal of computational physics*, 54(1):115–173, 1984.