Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2023-0026

Vol. **34**, No. 4, pp. 907-933
October 2023

Neural Networks with Local Converging Inputs (NNLCI) for Solving Conservation Laws, Part II: 2D Problems

Haoxiang Huang[1], Vigor Yang[2] and Yingjie Liu[3,*]

[1] *Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA.*
[2] *Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA.*
[3] *School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332, USA.*

Received 26 January 2023; Accepted (in revised version) 9 August 2023

Abstract. In our prior work [10], neural networks with local converging inputs (NNLCI) were introduced for solving one-dimensional conservation equations. Two solutions of a conservation law in a converging sequence, computed from low-cost numerical schemes, and in a local domain of dependence of the space-time location, were used as the input to a neural network in order to predict a high-fidelity solution at a given space-time location. In the present work, we extend the method to two-dimensional conservation systems and introduce different solution techniques. Numerical results demonstrate the validity and effectiveness of the NNLCI method for application to multi-dimensional problems. In spite of low-cost smeared input data, the NNLCI method is capable of accurately predicting shocks, contact discontinuities, and the smooth region of the entire field. The NNLCI method is relatively easy to train because of the use of local solvers. The computing time saving is between one and two orders of magnitude compared with the corresponding high-fidelity schemes for two-dimensional Riemann problems. The relative efficiency of the NNLCI method is expected to be substantially greater for problems with higher spatial dimensions or smooth solutions.

AMS subject classifications: 65L99, 65M99, 65N99, 68T99, 76L05

Key words: Neural network, neural networks with local converging inputs, physics informed machine learning, conservation laws, differential equation, multi-fidelity optimization.

1 Introduction

Artificial neural networks [9] are an important tool for computations in science and engineering. Many approaches have recently been developed that incorporate artificial

*Corresponding author. *Email addresses:* hcwong@gatech.edu (H. Huang), vigor.yang@aerospace.gatech.edu (V. Yang), yingjie@math.gatech.edu (Y. Liu)

neural networks for solving partial differential equations. For example, Sirignano and Spiliopoulos [29] introduced the Deep Galerkin Method to approximate the unknown solution as a mapping from a space-time location to the solution value there with a deep neural network, incorporating the finite difference residue error and initial and boundary constraints in the loss function. E and Yu [7] introduced the Deep Ritz Method, which incorporates the Ritz energy of a finite element method into the loss function. Raissi *et al.* [24] developed physics-informed neural networks (PINN) by employing an automatic differentiation [3] to define the residue error in the loss function. Much success has been achieved in predicting a variety of flow problems with given governing equations, including the Navier-Stokes system [23–26], hypersonic flow [18], electro-convection [21] and others. In [20], the Rankine-Hugoniot jump conditions were added as a constraint to the loss function of the neural network for solving the Riemann problems. In [14], a specially designed neural network was used to approximate the mapping from all known information, such as initial and boundary values, to the unknown solution, and many existing solutions have been used to train such a neural network. In [5, 19], finite expansions of neural networks that can be trained off-line were introduced to form a mapping from the initial value and a spatial location to a later high-fidelity solution at the same location.

Neural networks have also been trained off-line to predict key parameters of a numerical scheme. In [2, 6, 27], neural networks were used to detect discontinuities. An appropriate slope limiter or artificial viscosity was then determined to treat discontinuities using a local solution as the input. Another approach is to use a low-cost numerical solution computed on a coarse grid as input to predict a high-fidelity solution [15, 22].

In our earlier work [10], a novel neural network method (NNLCI) was introduced to solve conservation laws whose solutions may contain shock and contact discontinuities. In NNLCI, local low-cost solutions are employed as the input to a neural network to predict a high-fidelity solution at a given space-time location. To enable the neural network to distinguish a numerically smeared discontinuity from a smooth solution with large gradient in its input, the input is created by solving the conservation laws twice in sequence, with approximate solutions of converging accuracy, with low-cost numerical schemes and in a local domain of dependence of the space-time location. Because a numerical discontinuity becomes increasingly steeper in a converging sequence in the input, while a smooth solution does not, the neural network then can accurately identify flow attributes in its input and make the correct prediction. Such inputs can be generated in different ways, including schemes with two different grids (with one grid coarser than the other), with two different numerical diffusion coefficients on the same grid, or with two schemes of different orders of accuracy on the same grid. All inputs and high-fidelity solutions for all cases studied throughout the paper are computed by a first-order or fourth-order numerical scheme, using Dual Intel Xeon Gold 6226 processors. The NNLCI approach works effectively, not only for discontinuities, but also for smooth regions of the solution. It has broad application to a wide variety of differential equations. The computational cost is modest because it is a local post-processing-type solver, and

low-cost schemes on coarse grids can be used to generate inputs.

In the present work, we extend the NNLCI method [10] to two-dimensional conservation systems. Several different 2D Riemann problems adopted from [13] are studied. The results demonstrate the effectiveness of this method for multi-dimensional flows containing shock and contact discontinuities. TensorFLow [1] is used for the neural networks in the numerical experiments. Compared to several widely used numerical methods, such as MUSCL [30], ENO [8, 28], and WENO [11, 16], the proposed neural network method appears to be particularly useful for applications that need repetitive computations with varying parameters.

The paper is structured as follows. Section 2 describes the development of the proposed neural network with local converging inputs (NNLCI). Section 3 introduces several variants of NNLCI. Conclusions are presented in Section 4.

2 NNLCI for 2D Riemann problems

Consider the 2D scalar conservation law

$$\frac{\partial U}{\partial t}+\frac{\partial f(U)}{\partial x}+\frac{\partial g(U)}{\partial y}=0, \quad (x,y)\in\Omega\subset\mathcal{R}, \quad t\in[0,T]. \tag{2.1}$$

The Euler equations take the form

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ E\end{pmatrix}+\frac{\partial}{\partial x}\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ u(E+p)\end{pmatrix}+\frac{\partial}{\partial y}\begin{pmatrix}\rho v\\ \rho vu\\ \rho v^2+p\\ v(E+p)\end{pmatrix}=0, \quad (x,y)\in\Omega\subset\mathcal{R}, \quad t\in[0,T], \tag{2.2}$$

where Ω is a 2D spatial domain, and ρ, u, v and p are the density, x and y components of velocity, and pressure, respectively, with the equation of state specified in Section 2.2.

2.1 Input, output and loss function

The novel neural network method (NNLCI) is fully described in [10], and is only briefly described here. In NNLCI, local low-cost solutions are used as inputs, and a high-fidelity solution at a given space-time location is predicted. The neural network functions like a local post-processor that scans two low-cost numerical solutions and updates them to a high-fidelity solution. It takes local patches of the two low-cost numerical solutions as its input and yields a high-fidelity solution at a corresponding space-time location as its output. The two low-cost solutions must be converging to the exact solution, so one is closer to the exact solution than the other. The localness provides great efficiency and flexibility in training. In fact, one set of numerical solutions (two low-cost numerical solutions for inputs plus one fine-grid numerical solution used as the reference solution in training) provides a large number of local patches for training the neural network.

In our experiments a dozen or even fewer fine-grid numerical simulations are sufficient to provide good training of the neural network. The local (low-cost) solution patches for input in the training process can be cropped around prediction target times, or at all numerical time levels. They can also be cropped sparsely where the solution has little change. Because of the local input design, simple standard neural network structures are adequate for NNLCI to make good predictions.

Let $\Omega=[a,b]\times[c,d]$ be partitioned with the coarsest uniform rectangular grid $a=x_0<x_1<\cdots<x_M=b$ and $c=y_0<y_1<\cdots<y_N=d$. The spatial grid size is $\Delta x=x_1-x_0$ and $\Delta y=y_1-y_0$, and the time step size is Δt. We refine the spatial grid to $\frac{\Delta x}{2}$ and $\frac{\Delta y}{2}$, and time step size $\frac{\Delta t}{2}$. Let L be a low-cost scheme for computing (2.1) on both grids. For predicting the solution at (x,y,t), that is (i',j',n') in the coarsest grid, we choose the coarsest grid solution (computed by L) at 9 points $(x_{i'-1},y_{j'-1})$, $(x_{i'-1},y_{j'})$, $(x_{i'-1},y_{j'+1})$, $(x_{i'},y_{j'-1})$, $(x_{i'},y_{j'})$, $(x_{i'},y_{j'+1})$, $(x_{i'+1},y_{j'-1})$, $(x_{i'+1},y_{j'})$, and $(x_{i'+1},y_{j'+1})$, at time level $t_{n'-1}$, along with point $(x_{i'},y_{i'},t_{n'})$, as the first part of the input of the neural network. The finer grid solution (also computed by L) at the same space-time locations is used as the second part of input. The chosen 10 space-time locations on both grids enclose a local (space-time) domain of dependence of the exact solution at $(x_{i'},y_{i'},t_{n'})$.

Denote the first part of the 2D input as

$$w^{n'-1}_{i'-1,j'-1},w^{n'-1}_{i'-1,j'},w^{n'-1}_{i'-1,j'+1},w^{n'-1}_{i',j'-1},w^{n'-1}_{i',j'},w^{n'-1}_{i',j'+1},w^{n'-1}_{i'+1,j'-1},w^{n'-1}_{i'+1,j'},w^{n'-1}_{i'+1,j'+1},w^{n'}_{i',j'}, \tag{2.3}$$

and the second part of the 2D input as

$$w^{n''-2}_{i''-2,j''-2},w^{n''-2}_{i''-2,j''},w^{n''-2}_{i''-2,j''+2},w^{n''-2}_{i'',j''-2},w^{n''-2}_{i'',j''},w^{n''-2}_{i'',j''+2},w^{n''-2}_{i''+2,j''-2},w^{n''-2}_{i''+2,j''},w^{n''-2}_{i''+2,j''+2},w^{n''}_{i'',j''}. \tag{2.4}$$

Note that the space-time indices (i',j',n') on the coarsest grid refers to the same location as (i'',j'',n'') on the finer grid, $(i'-1,j'-1,n'-1)$ refers to the same location as $(i''-2,j''-2,n''-2)$ does, and so on. The input of NNLCI

$$\begin{aligned}\{&w^{n'-1}_{i'-1,j'-1},w^{n'-1}_{i'-1,j'},w^{n'-1}_{i'-1,j'+1},w^{n'-1}_{i',j'-1},w^{n'-1}_{i',j'},w^{n'-1}_{i',j'+1},w^{n'-1}_{i'+1,j'-1},w^{n'-1}_{i'+1,j'},w^{n'-1}_{i'+1,j'+1},w^{n'}_{i',j'},\\ &w^{n''-2}_{i''-2,j''-2},w^{n''-2}_{i''-2,j''},w^{n''-2}_{i''-2,j''+2},w^{n''-2}_{i'',j''-2},w^{n''-2}_{i'',j''},w^{n''-2}_{i'',j''+2},\\ &w^{n''-2}_{i''+2,j''-2},w^{n''-2}_{i''+2,j''},w^{n''-2}_{i''+2,j''+2},w^{n''}_{i'',j''}\},\end{aligned} \tag{2.5}$$

is now called the "input of w." The corresponding output of NNLCI is the predicted solution of (2.1) at (x,y,t) on the coarsest grid. See Figs. 1 and 2 for an illustration of NNLCI. For the Euler system, the input and output of NNLCI are made up of corresponding inputs and outputs for each prime variable. For example, the input can be the vector

$$\{\text{input of}\,\rho,\text{input of}\,u,\text{input of}\,v,\text{input of}\,p\} \tag{2.6}$$

with $20\times4=80$ elements, and the corresponding output will be $\{\rho,u,v,p\}$ at (x,y,t), or (i',j',n') on the coarsest grid with 4 elements.

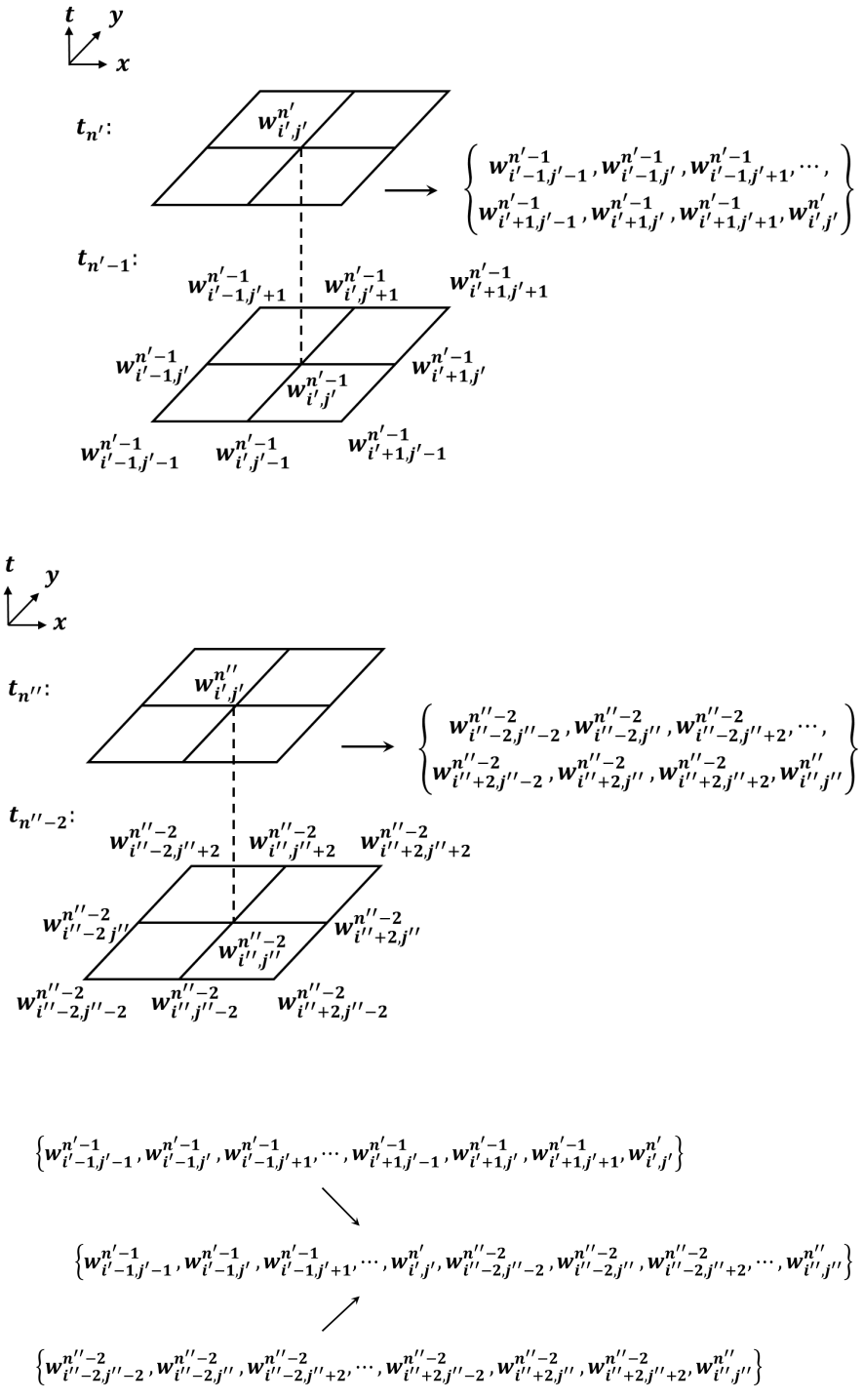

Figure 1: Procedure for formatting the input for NNLCI.

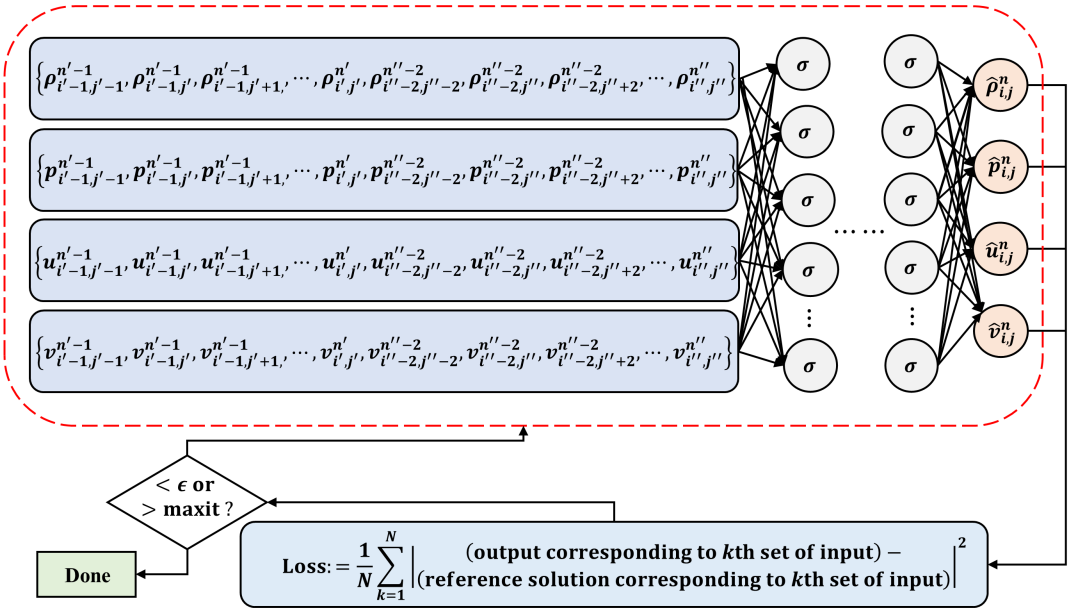

Figure 2: Training procedure for NNLCI.

The loss function measures the difference between the output and the reference solution corresponding to the input, and is defined as follows.

$$\begin{aligned}\text{Loss} = \frac{1}{N}\sum_{k=1}^{N} |&(\text{output corresponding to } k^{th} \text{ set of input})\\ &- (\text{reference solution corresponding to } k^{th} \text{ set of input})|^2,\end{aligned} \tag{2.7}$$

where $|\cdot|$ is the 2-norm measuring the distance between the output vector and the reference solution vector at the same space-time location. The summation includes every set of input (corresponding to different space-time location or initial condition) in the training data. Note that in the summation every output and its corresponding reference solution must be at the same space-time location. However, there may be terms (output and corresponding reference solution) in the summation at different time levels, because the neural network only makes a local prediction which we can take advantage of. Suppose we want to predict the solution at the final time. The summation can include input (2.6) at the final or intermediate times for multiple initial conditions used in training. The latter is more costly but yields similar predictions in our numerical tests.

2.2 Generation of input and training data

In this section, the NNLCI developed in [10] is extended from one- to two-dimensional problems. Six different cases of the Riemann problems for the Euler equations, adopted from [13], are considered. We consider the Euler equations (2.2) for an ideal gas with

$$E=\frac{p}{\gamma-1}+\frac{1}{2}\rho(u^2+v^2), \tag{2.8}$$

and $\gamma=1.4$. The initial values are constant in each quadrant of the spatial domain $[0,1]\times[0,1]$, where Quadrant $1=(0.5,1)\times(0.5,1)$; Quadrant $2=(0,0.5)\times(0.5,1)$; Quadrant $3=(0,0.5)\times(0,0.5)$; and Quadrant $4=(0.5,1)\times(0,0.5)$. Table 1 lists the initial conditions.

The initial values of the training and prediction cases are perturbed from the original [13] data. Fig. 3 shows a representative example (Case 6).

The NNLCI method is used to predict the solutions of several Riemann problems described in [13], with the appropriate neural network being used for each case. (See Table 1.) Different neural networks are required for NNLCI for different cases. For Cases 1, 2, and 3, the corresponding neural network consists of 8 hidden layers, each with 320 neurons. For Cases 6 and 8, it also consists of 8 hidden layers. Case 4 needs a neural network with 9 hidden layers, each with 360 neurons. The activation function has the form of *tanh*. During the training process, the neural network minimizes the difference between outputs and a reference solution by using first an Adam optimizer then an L-BFGS optimizer in TensorFLow. The number of iterations for each optimization procedure is less than 50000. Upon completion of the training, the neural network is used to predict

Table 1: Initial conditions for 2D Euler system, adopted from [13].

	2-D Riemann Problems		
	Case 1	Case 2	Case 3
Quadrant	Initial Values (ρ,u,v,p)		
1	$(1.00,0.00,0.00,1.00)$	$(1.00,0.00,0.00,1.00)$	$(1.50,0.00,0.00,1.50)$
2	$(0.52,-0.73,0.00,0.40)$	$(0.52,-0.73,0.00,0.40)$	$(0.53,1.21,0.00,0.30)$
3	$(0.11,-0.73,-1.40,0.04)$	$(1.00,-0.73,-0.73,1.00)$	$(0.14,1.21,1.21,0.03)$
4	$(0.26,0.00,-1.40,0.15)$	$(0.52,0.00,-0.73,0.40)$	$(0.53,0.00,1.21,0.30)$

	2-D Riemann Problems		
	Case 4	Case 6	Case 8
Quadrant	Initial Values (ρ,u,v,p)		
1	$(1.10,0.00,0.00,1.10)$	$(1.00,0.75,-0.50,1.00)$	$(0.52,0.10,0.10,0.40)$
2	$(0.51,0.89,0.00,0.35)$	$(2.00,0.75,0.50,1.00)$	$(1.00,-0.63,0.10,1.00)$
3	$(1.10,0.89,0.89,1.10)$	$(1.00,-0.75,0.50,1.00)$	$(0.80,0.10,0.10,1.00)$
4	$(0.51,0.00,0.89,0.35)$	$(3.00,-0.75,-0.50,1.00)$	$(1.00,0.10,-0.63,1.00)$

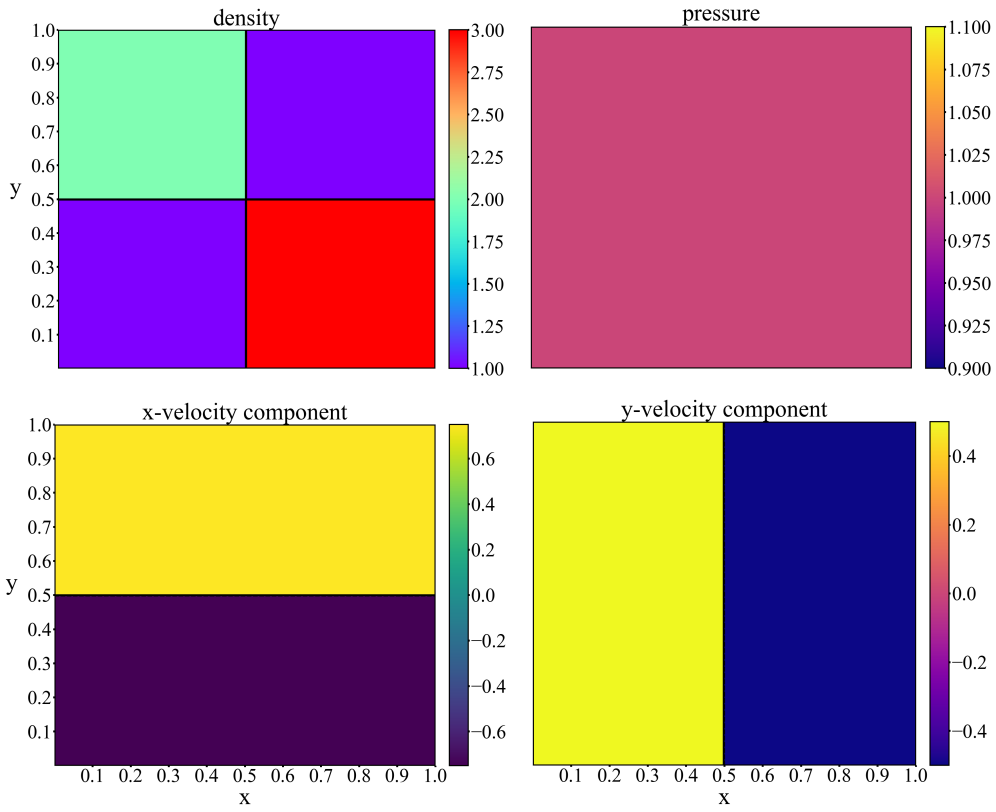

Figure 3: Initial values of Case 6: density, pressure, x-velocity, and y-velocity.

solutions, with inputs computed by the same low-cost schemes and grids as the training data.

In the fully connected neural network, we explored different numbers of layers and neurons per layer to achieve the optimal setup. The prediction results were found to not be sensitive to network structure, so that we could adjust the structure by increments of at least 10%. In a case of excessive neurons, the minimization process could easily end up at a wrong local minimum. On the other hand, if the number of neurons is too small, the prediction error could be large because the neural network was not able to approximate the solution well. Other types of neural networks, such as the convolutional neural network, will be explored to reduce the number of parameters. The results of prediction by NNLCI, however, are not expected to vary significantly when using different types of neural networks. Different initial values were considered to validate the NNLCI method, including the original initial value of the configuration, and $\pm 3\%$ and $\pm 5\%$ perturbations of the initial value.

To generate the training data, we use a first-order scheme on the coarse and finer uniform grids (200 and 400 cells in each spatial coordinate, respectively) to compute the

input data for several different initial values for each case, including $\pm 2\%$, $\pm 4\%$, $\pm 6\%$, $\pm 8\%$ and $\pm 10\%$ perturbations of the original initial value. High-resolution reference solutions for the training data are computed on a uniform grid with 400 cells in each spatial coordinate. A 4th-order central scheme on overlapping cells with hierarchical reconstruction (HR) limiting [17] is employed to compute reference solutions for all 2D Riemann problems. The low-cost scheme used for computing inputs in all cases is the first-order leapfrog and diffusion splitting scheme.

$$\begin{cases} \frac{\tilde{U}_{i,j}-U_{i,j}^{n-1}}{2\Delta t}+\frac{f(U)|_{i+1,j}^{n}-f(U)|_{i-1,j}^{n}}{2\Delta x}+\frac{g(U)|_{i,j+1}^{n}-g(U)|_{i,j-1}^{n}}{2\Delta y}=0, \\ \frac{U_{i,j}^{n+1}-\tilde{U}_{i,j}}{\Delta t}-\alpha\big[\frac{\tilde{U}_{i+1,j}-2\cdot\tilde{U}_{i,j}+\tilde{U}_{i-1,j}}{\Delta x^2}+\frac{\tilde{U}_{i,j+1}-2\cdot\tilde{U}_{i,j}+\tilde{U}_{i,j-1}}{\Delta y^2}\big]=0, \end{cases} \tag{2.9}$$

where $\alpha=\Delta x$, and $\Delta x=\Delta y$. The time step size remains fixed at Δt for the 200×200 grid and $\frac{\Delta t}{2}$ for the 400×400 grid, to satisfy the CFL condition.

2.3 Results and discussion of 2D Riemann problems

The first-order leapfrog and diffusion splitting scheme (2.9) is employed to compute the inputs for NNLCI for most of the cases considered here. We utilize input patches at the final time level and the high-fidelity solution at corresponding space-time locations for training cases with $\pm 2\%$,$\pm 4\%$,$\pm 6\%$,$\pm 8\%$,and$\pm 10\%$ perturbations of the original initial value. Once the neural network is trained, it can efficiently predict a high-fidelity solution in less than one second, with its initial value in the convex hull of those used in training. The spatial computational domain is $[0,1]\times[0,1]$ unless otherwise specified. Fig. 4 shows the predicted final-time flow solution of the 2D Euler system for Case 6. Fig. 5 shows the axial distribution of density at $y=0.34$, $y=0.50$, $y=0.60$, $y=0.70$, and $y=0.80$. Fig. 6 shows the prediction with the initial value perturbed by $+5\%$ from Case 6. Fig. 7 shows the density profiles at various vertical locations. Excellent agreement between the NNLCI prediction and the high-fidelity reference solution is achieved.

For Case 6, the inputs for each simulation with perturbed initial condition take ~ 1 hour (walltime) for Dual Intel Xeon Gold 6226 to compute, to reach the time step corresponding to the final time step in the high-fidelity solution. The high-fidelity solution takes ~ 24 hours (walltime) per simulation in the same computing environment. Since the time of prediction is around 1s, which is negligible compared to data generation for inputs and solutions, NNLCI extrapolates the high-fidelity results ~ 24 times faster than the 4th-order finite volume scheme [17] for each initial condition simulation. For rest of the cases presented here, with two coarse grid inputs computed by the leapfrog and diffusion splitting scheme (2.9), NNLCI predicts high-fidelity results with similar high time savings, consistently showing the same high accuracy of prediction.

To study the prediction accuracy when the training data are more sparse, we increase the spacing (defined as relative distance between two neighboring data points) in the training data from 2-4% to about 10% for Case 6. The training data now consist of solutions of cases with original initial value and $\pm 10\%$ perturbations from the initial value of

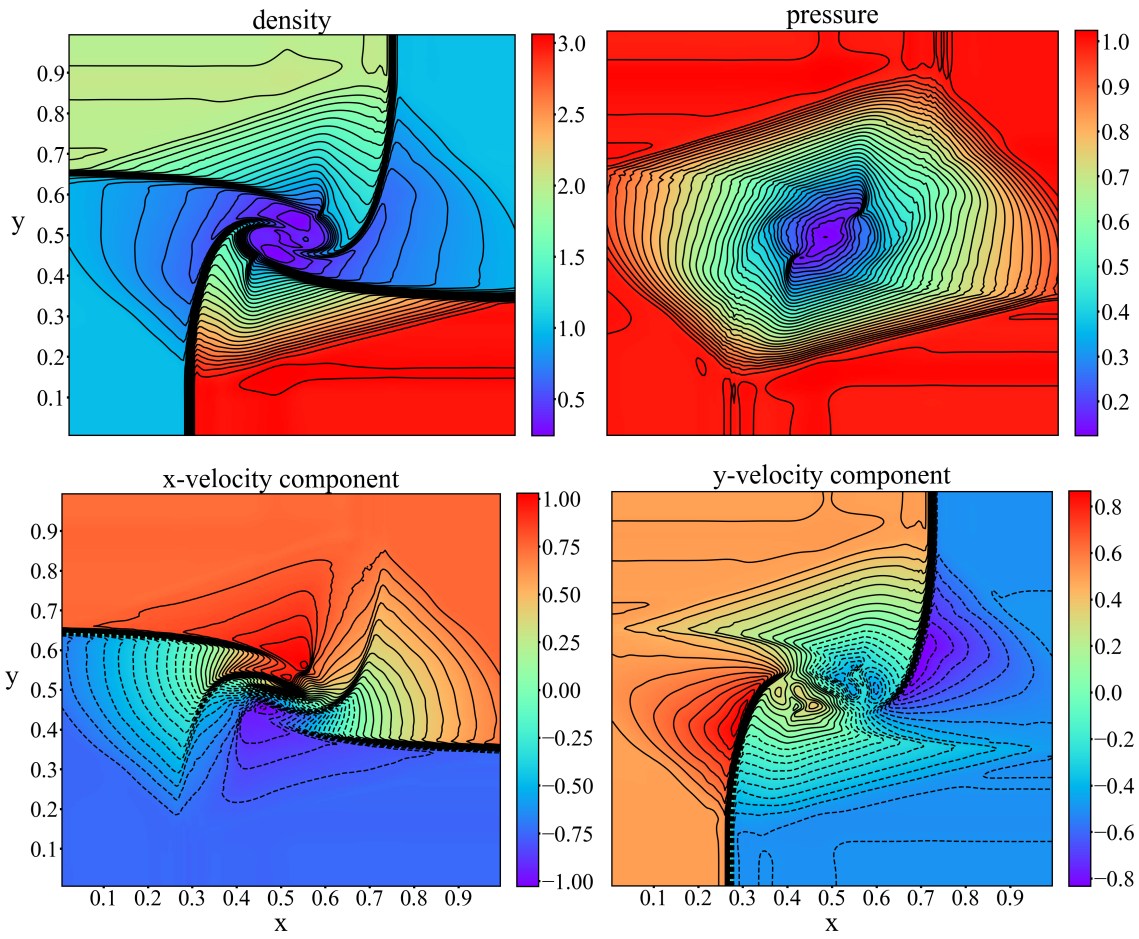

Figure 4: NNLCI prediction of the final-time $(t=0.3)$ solution of Case 6: density, pressure, x-velocity, and y-velocity.

Case 6. After training, the neural network is used to predict high-fidelity solutions for problems with other initial values. Fig. 8 shows the prediction results for the case with initial value perturbed 8% from Case 6. The inputs are computed by the leapfrog and diffusion splitting scheme (2.9).

Cases 1, 2, 3, 4 and 8 were also carefully examined. Fig. 9 shows the NNLCI-predicted density field for Case 1 at the final-time $t=0.2$. The spatial computational domain is $[0.25,0.95]\times[0.25,0.95]$. Fig. 10 shows the predicted density distribution with the initial value perturbed by $+5\%$ from that of Case 1. The low-fidelity inputs to the neural network and high-fidelity reference solution are also included for comparison.

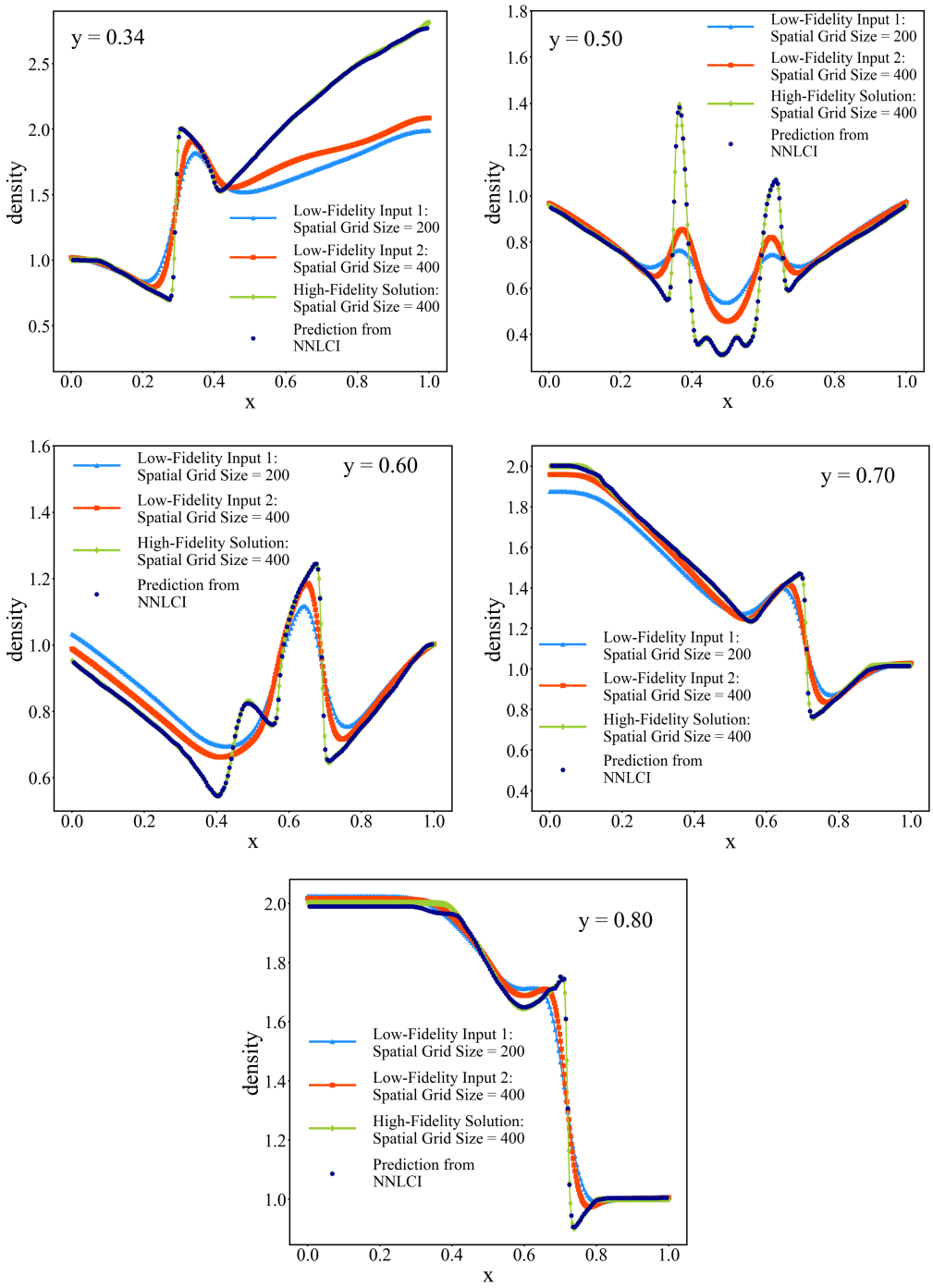

Figure 5: Axial distributions of density along $y=0.34$, $y=0.50$, $y=0.60$, $y=0.70$, and $y=0.80$ of the NNLCI prediction of the final-time $(t=0.3)$ solution of Case 6. Predicted density (dark blue), low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids respectively, and reference solution (green).

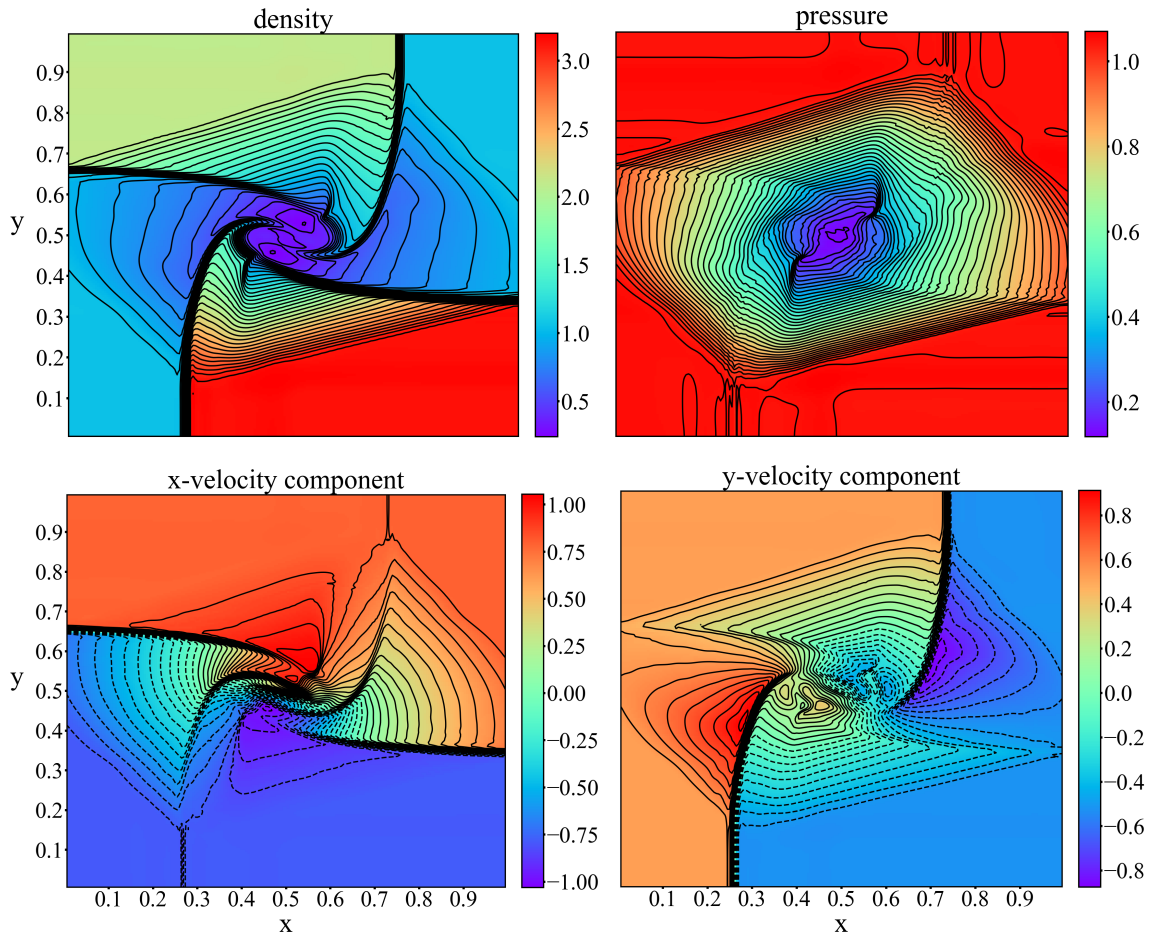

Figure 6: NNLCI prediction of the final-time $(t=0.3)$ solution of the 2D Euler system, with initial value $+5\%$ perturbed from that of Case 6: density, pressure, x-velocity, and y-velocity.

Fig. 11 shows the predicted final-time density field of Case 2. The spatial computational domain is $x,y\in[0,0.85]$. The low-fidelity inputs were calculated using 200×200 and 400×400 grids in sequence. Fig. 12 shows the NNLCI-predicted final-time density solution of the 2D Euler system with the initial value perturbed by $+5\%$ from that of Case 2. Figs. 13 and 14 show the situations with Case 3. The spatial computational domain is $x,y\in[0,0.9]$.

In certain areas, the low-cost input solutions of Case 3 computed using the first-order scheme (2.9)may not be qualitatively similar to the high-fidelity solution for NNLCI to make an accurate prediction. To improve the quality of the inputs, therefore, we computed two input solutions on 100×100 and 200×200 grids by means of the 4th-order

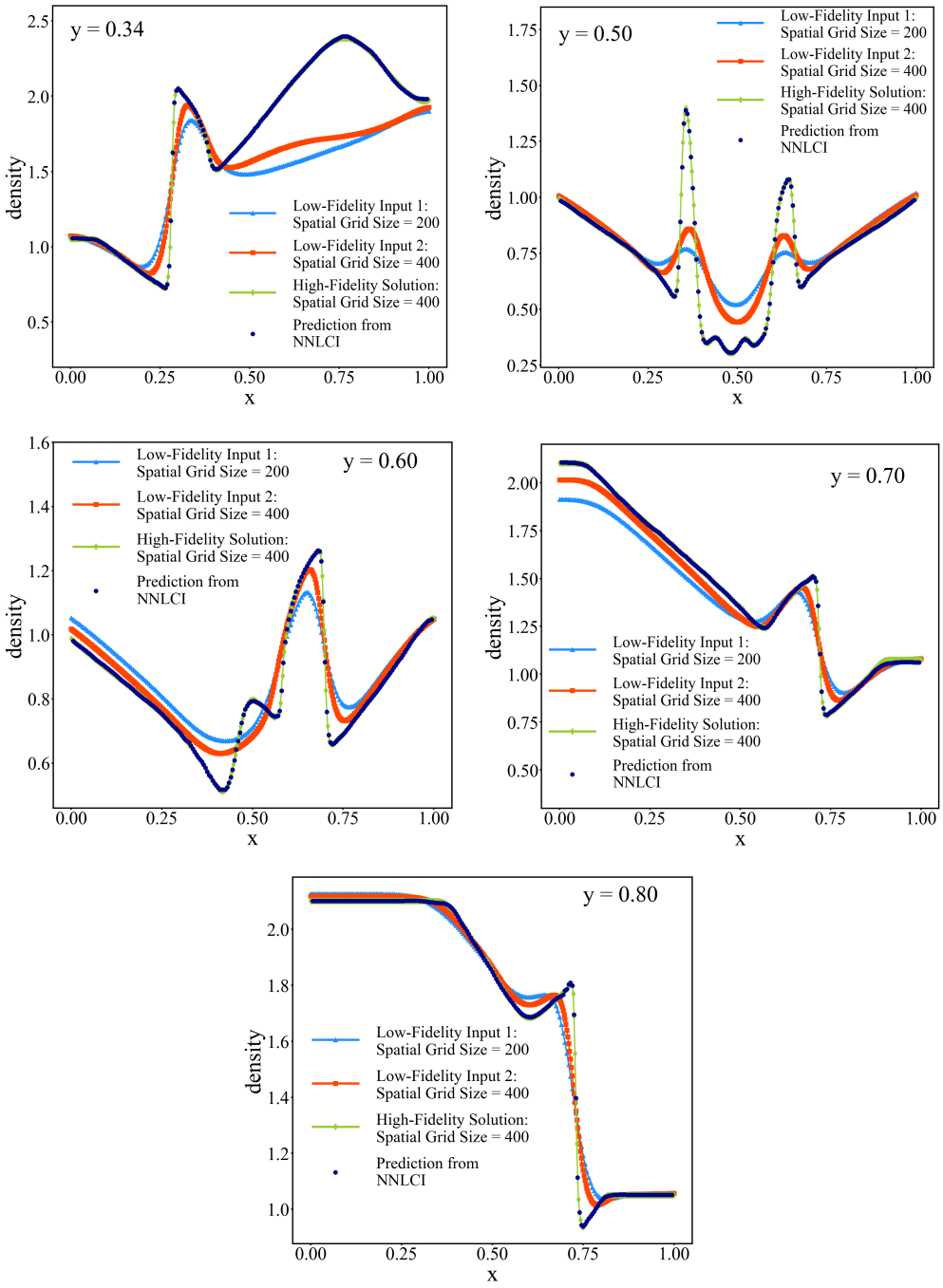

Figure 7: Axial distributions of density along $y=0.34$, $y=0.50$, $y=0.60$, $y=0.70$, and $y=0.80$ of the NNLCI prediction of the final-time $(t=0.3)$ solution of the 2D Euler system, with initial value $+5\%$ perturbed from that of Case 6. Predicted density (dark blue), low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids respectively, and reference solution (green).

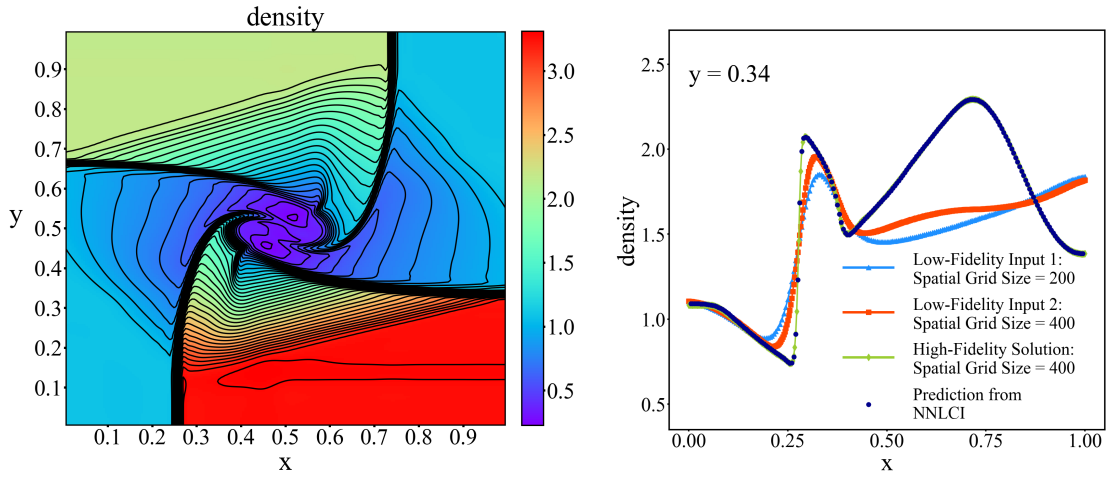

Figure 8: NNLCI prediction of final-time $(t=0.3)$ density distribution of 2D Euler system with initial value perturbed by $+8\%$ from that of Case 6 (see Table 1), and axial distribution of density (dark blue) along $y=0.34$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

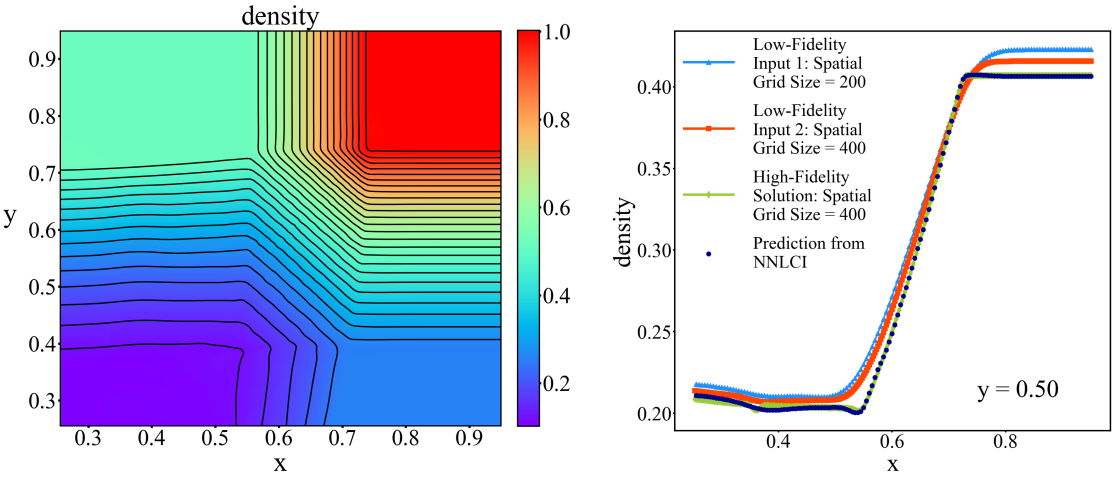

Figure 9: NNLCI prediction of final-time $(t=0.2)$ density distribution of Case 1 (see Table 1), and axial distribution of density (dark blue) along $y=0.50$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

scheme [17] used in the reference solution. Figs. 15 and 16 show improved predictions as compared with those in Figs. 13 and 14, especially in the center region. Inputs generation by the 4th-order scheme take $\sim$2 hours (walltime) to compute, while high-fidelity solution take $\sim$16 hours (walltime) for each simulation. That is, NNLCI predicts high-fidelity

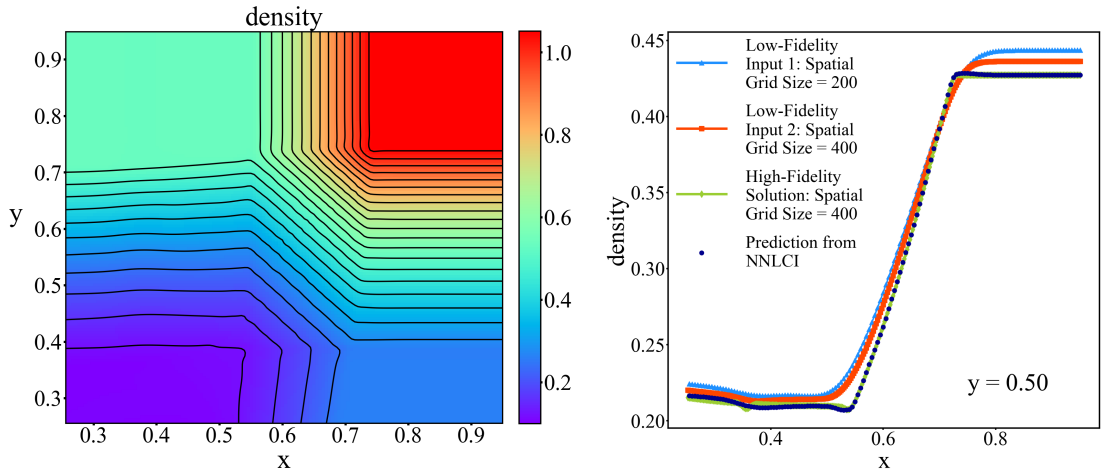

Figure 10: NNLCI prediction of final-time $(t=0.2)$ density distribution of 2D Euler system with initial value perturbed by $+5\%$ from that of Case 1 (see Table 1), and axial distribution of density (dark blue) along $y=0.50$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

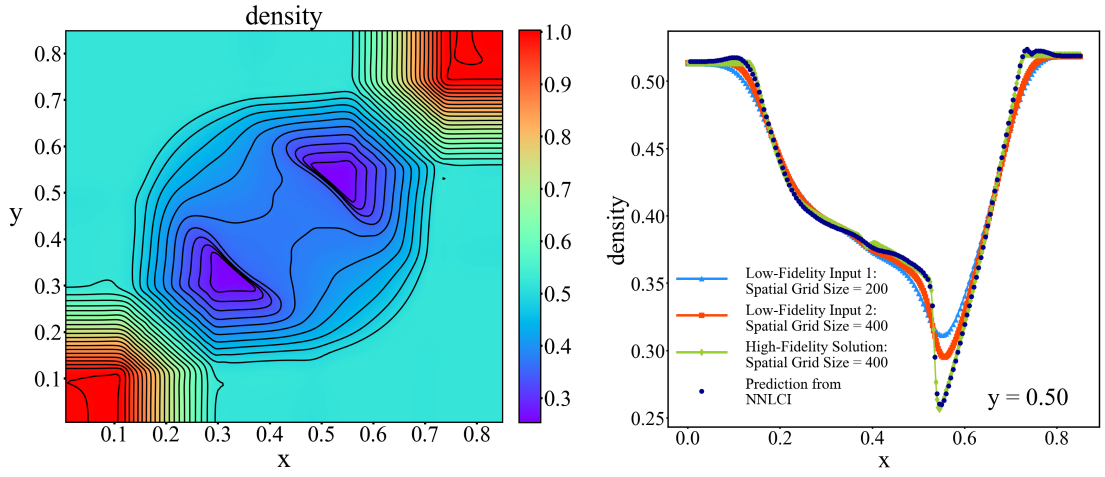

Figure 11: NNLCI prediction of final-time $(t=0.2)$ density distribution of Case 2 (see Table 1), and axial distribution of density (dark blue) along $y=0.50$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

results in about $1/8$ of the time of the corresponding high-fidelity simulation.

Fig. 17 show the NNLCI- predicted final-time solution of Case 4. The situation with the initial value perturbed by $+5\%$ from that of Case 4 is given in Fig. 18. The spatial computational domain is $x,y\in[0.22,0.98]$.

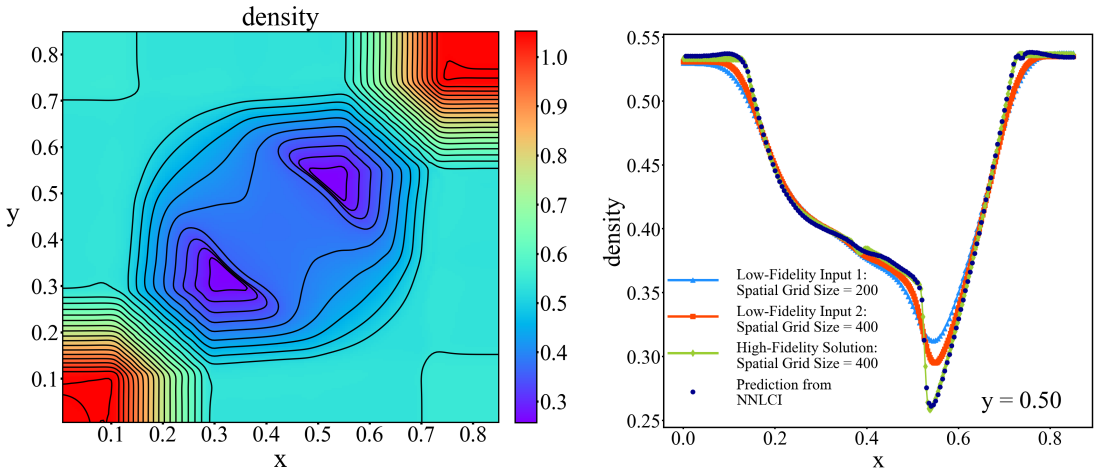

Figure 12: NNLCI prediction of final-time $(t=0.2)$ density distribution of 2D Euler system with initial value perturbed by $+5\%$ from that of Case 2 (see Table 1), and axial distribution of density (dark blue) along $y=0.50$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

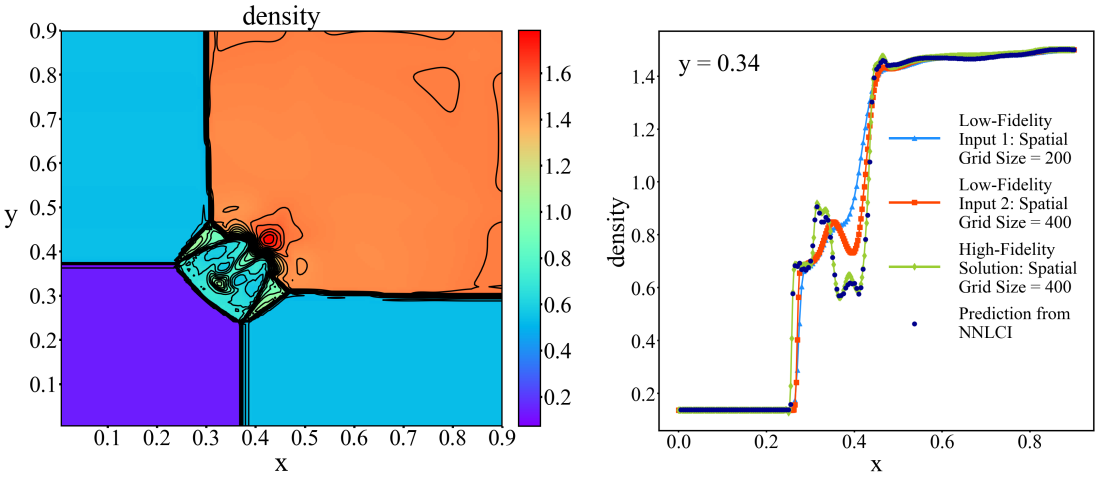

Figure 13: NNLCI prediction of final-time $(t=0.3)$ density distribution of Case 3 in (see Table 1), and axial distribution of density (dark blue) along $y=0.34$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

Fig. 19 show the NNLCI- predicted final-time solution of Case 8. The situation with the initial value perturbed by $+5\%$ from that of Case 8 is given in Fig. 20. The spatial computational domain is $x,y\in[0,0.9]$.

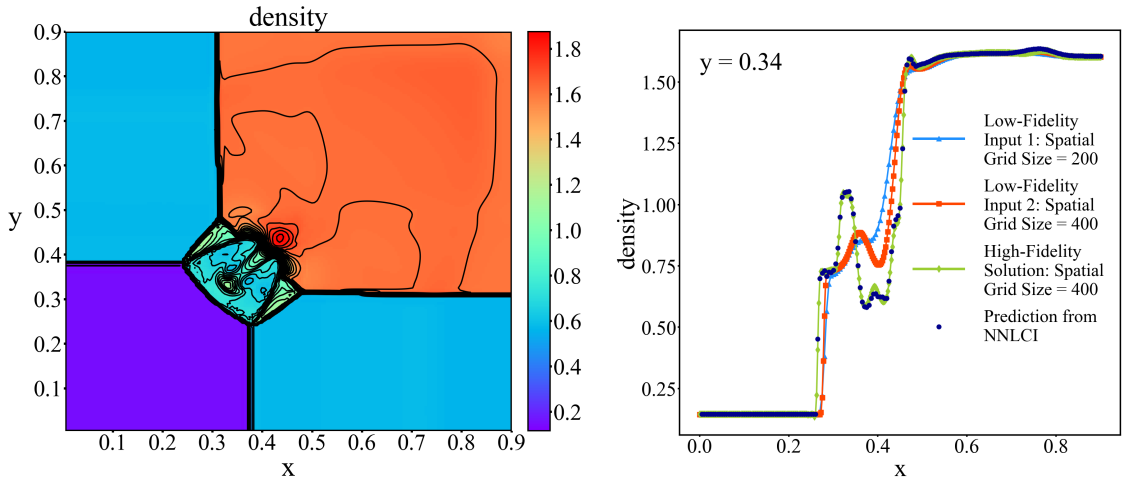

Figure 14: NNLCI prediction of final-time $(t=0.3)$ density distribution of 2D Euler system with initial value perturbed by $+5\%$ from that of Case 3 (see Table 1), and axial distribution of density (dark blue) along $y=0.34$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

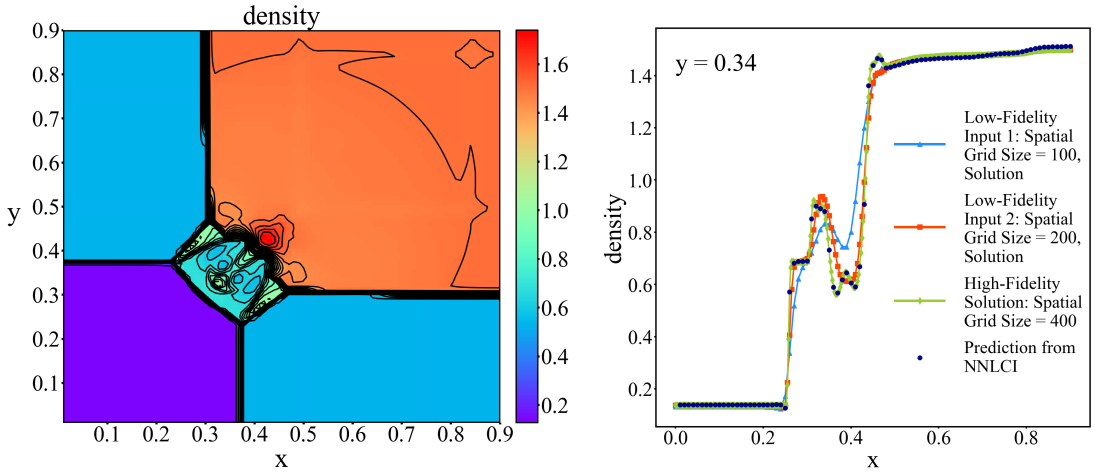

Figure 15: NNLCI prediction of final-time $(t=0.3)$ density distribution of Case 3 (see Table 1), and axial distribution of density (dark blue) along $y=0.34$, compared to low-fidelity input solutions (blue and red) on 100×100 and 200×200 grids, respectively, and reference solution (green).

Tables 2 and 3 summarize the overall results for the 2-D Riemann problems, using the NNLCI method, in terms of the relative l_2 errors.

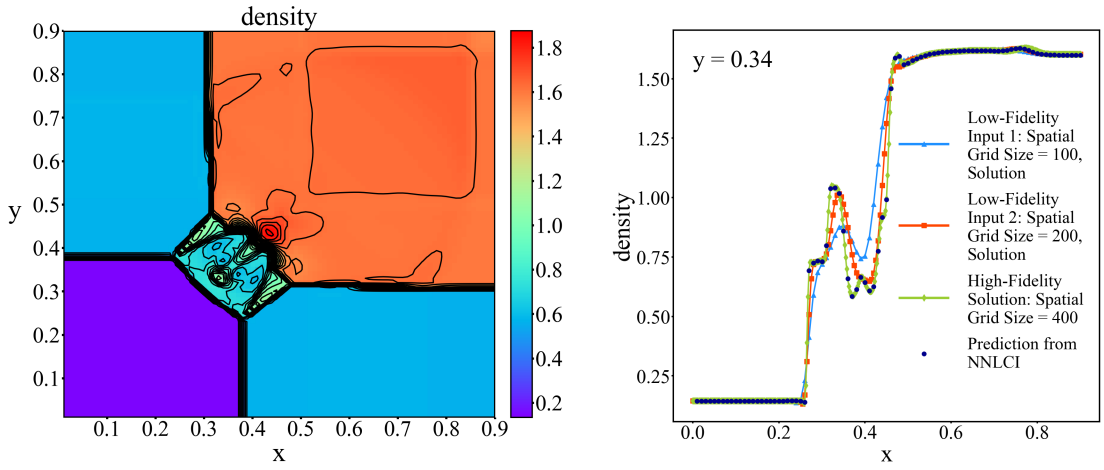

Figure 16: NNLCI prediction of final-time $(t=0.3)$ density distribution of 2D Euler system with initial value $+5\%$ perturbed from that of Case 3 (see Table 1), and axial distribution of density (dark blue) along $y=0.34$, compared to low-fidelity input solutions (blue and red) on 100×100 and 200×200 grids, respectively, and reference solution (green).

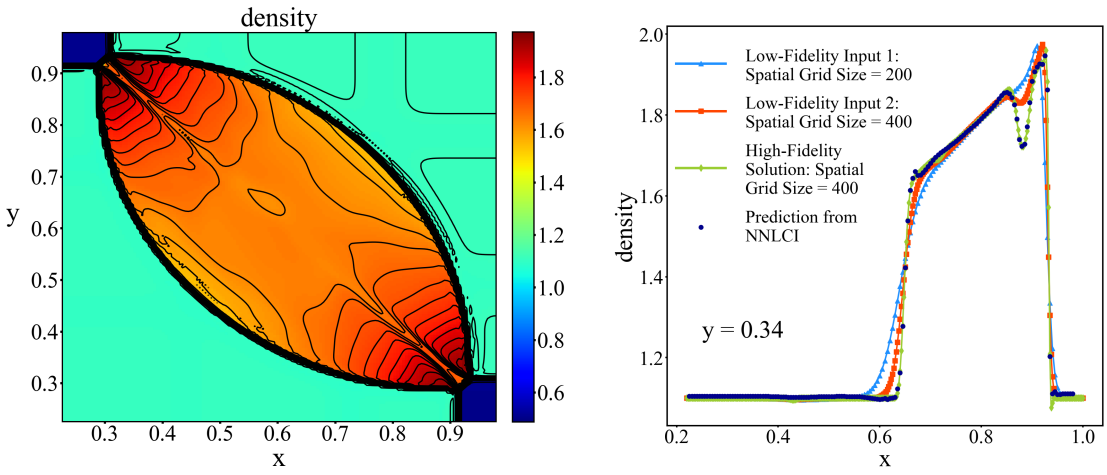

Figure 17: NNLCI prediction of final-time $(t=0.25)$ density distribution of Case 4 (see Table 1), and axial distribution of density (dark blue) along $y=0.34$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

3 NNLCI with input on single grid

Instead of using two different grids, as seen in the cases above, the input for NNLCI can be generated on a single grid, by introducing small variations to the governing equations.

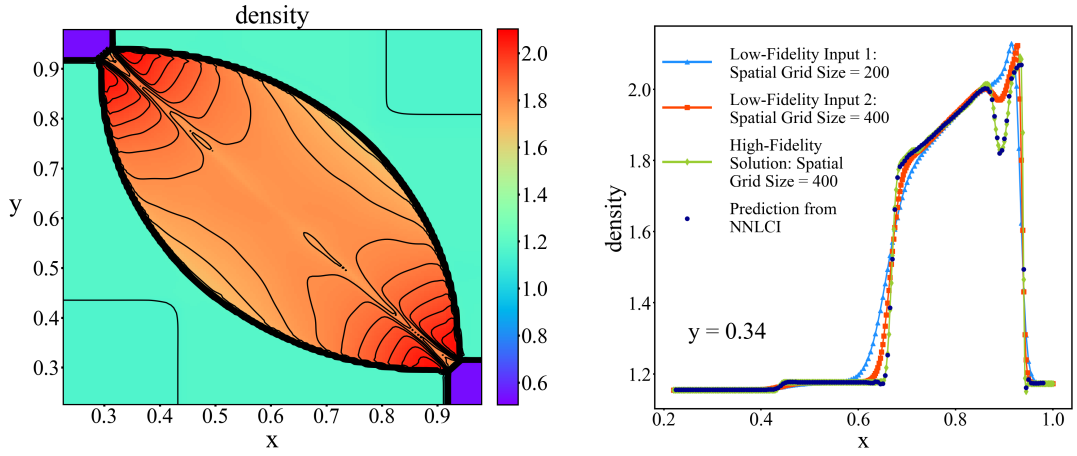

Figure 18: NNLCI prediction of final-time $(t=0.25)$ density distribution of 2d Euler system with initial value $+5\%$ perturbed from that of Case 4 (see Table 1), and axial distribution of density (dark blue) along $y=0.34$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

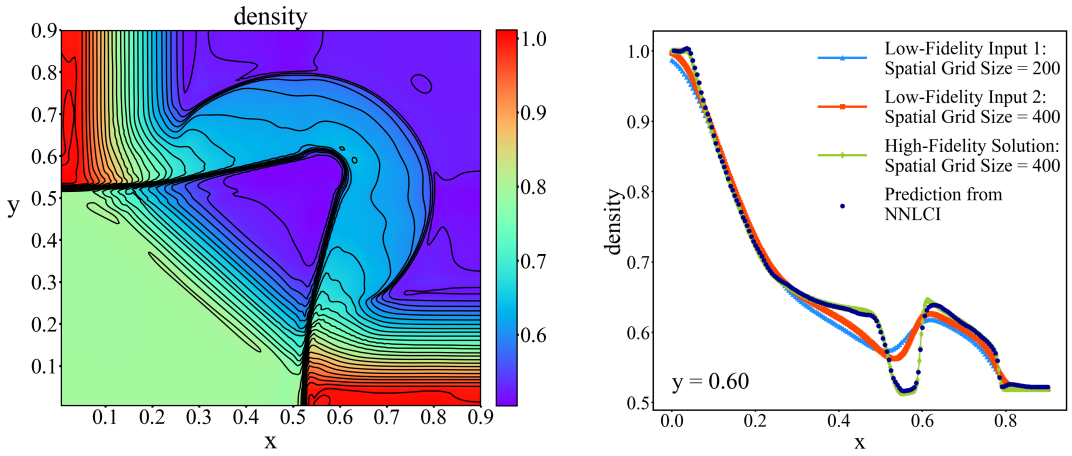

Figure 19: NNLCI prediction of final-time $(t=0.25)$ density distribution of Case 8 (see Table 1), and axial distribution of density (dark blue) along $y=0.60$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

This can be achieved, for example, by means of the vanishing viscosity approach [4, 12, 31]. See [10] in 1D case. We approximate (2.1) using the leapfrog and diffusion splitting scheme (2.9) with two different α (i.e., $\alpha=\Delta x$ and $c\Delta x$), as follows

$$\begin{cases} \frac{\tilde{U}_{i,j}-U_{i,j}^{n-1}}{2\Delta t}+\frac{f(U)|_{i+1,j}^{n}-f(U)|_{i-1,j}^{n}}{2\Delta x}+\frac{g(U)|_{i,j+1}^{n}-g(U)|_{i,j-1}^{n}}{2\Delta y}=0, \\ \frac{U_{i,j}^{n+1}-\tilde{U}_{i,j}}{\Delta t}-\Delta x\left[\frac{\tilde{U}_{i+1,j}-2\cdot\tilde{U}_{i,j}+\tilde{U}_{i-1,j}}{\Delta x^2}+\frac{\tilde{U}_{i,j+1}-2\cdot\tilde{U}_{i,j}+\tilde{U}_{i,j-1}}{\Delta y^2}\right]=0, \end{cases} \tag{3.1}$$

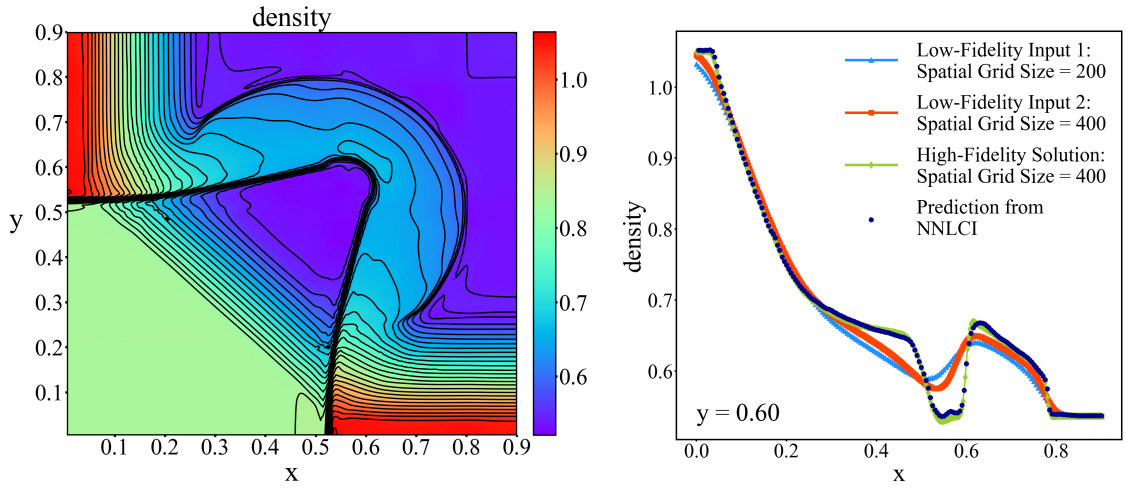

Figure 20: NNLCI prediction of final-time $(t=0.25)$ density distribution of 2D Euler system with initial value $+5\%$ perturbed from that of Case 8 (see Table 1), and axial distribution of density (dark blue) along $y=0.60$, compared to low-fidelity input solutions (blue and red) on 200×200 and 400×400 grids, respectively, and reference solution (green).

Table 2: Relative l_2 errors of NNLCI predictions based on low-fidelity inputs computed by leapfrog and diffusion splitting scheme on 200×200 and 400×400 grids.

	2-D Riemann Problem				
NNLCI Input Method	**leapfrog and diffusion splitting scheme**				
	Case 1	Case 2	Case 4	Case 6	Case 8
Initial Value	Relative l_2 Errors				
original	0.18E−2	0.33E−2	0.57E−2	0.66E−2	0.40E−2
$+3\%$	0.17E−2	0.30E−2	0.49E−2	0.39E−2	0.39E−2
-3%	0.17E−2	0.31E−2	0.39E−2	0.37E−2	0.41E−2
$+5\%$	0.17E−2	0.31E−2	0.39E−2	0.41E−2	0.37E−2
-5%	0.16E−2	0.31E−2	0.41E−2	0.40E−2	0.49E−2

and

$$\begin{cases} \frac{\tilde{V}_{i,j}-V_{i,j}^{n-1}}{2\Delta t}+\frac{f(V)|_{i+1,j}^{n}-f(V)|_{i-1,j}^{n}}{2\Delta x}+\frac{g(V)|_{i,j+1}^{n}-g(V)|_{i,j-1}^{n}}{2\Delta y}=0, \\ \frac{V_{i,j}^{n+1}-\tilde{V}_{i,j}}{\Delta t}-c\Delta x\Big[\frac{\tilde{V}_{i+1,j}-2\cdot\tilde{V}_{i,j}+\tilde{V}_{i-1,j}}{\Delta x^2}+\frac{\tilde{V}_{i,j+1}-2\cdot\tilde{V}_{i,j}+\tilde{V}_{i,j-1}}{\Delta y^2}\Big]=0. \end{cases} \tag{3.2}$$

Table 3: Relative l_2 of NNLCI predictions based on low-fidelity inputs computed by (1) leapfrog and diffusion splitting scheme on 200×200 and 400×400 grids; (2) 4th order scheme on 100×100 and 200×200 grids.

	2-D Riemann Problem, Case 3	
NNLCI Input Method	**leapfrog and diffusion splitting scheme**	**4th-order scheme**
Initial Value	Relative l_2 Errors	
original	1.71E−2	0.78E−2
+3%	2.62E−2	0.30E−2
−3%	2.77E−2	0.20E−2
+5%	4.02E−2	0.38E−2
−5%	1.55E−2	0.39E−2

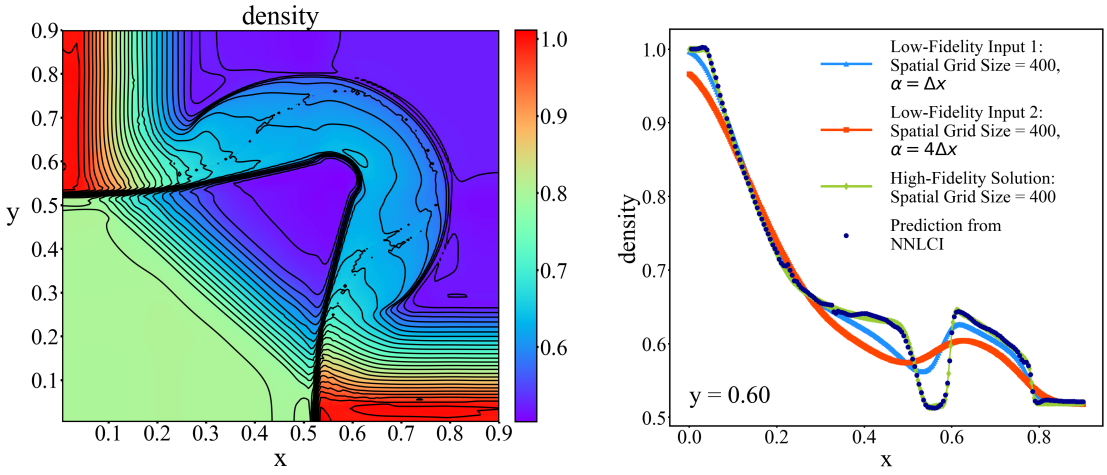

Figure 21: NNLCI prediction of final-time $(t=0.25)$ density distribution of Case 8 (see Table 1), and axial distribution of density (dark blue) along $y=0.60$, compared to low-fidelity input solutions (blue and red) from leapfrog and diffusion splitting schemes (3.1) and (3.2) ($c=4$), respectively on the 400×400 grid, and reference solution (green).

The input computed on a single uniform grid can be written as

$$\begin{aligned}&\{U^{n-1}_{i-1,j-1},U^{n-1}_{i-1,j},U^{n-1}_{i-1,j+1},U^{n-1}_{i,j-1},U^{n-1}_{i,j},U^{n-1}_{i,j+1},U^{n-1}_{i+1,j-1},U^{n-1}_{i+1,j},U^{n-1}_{i+1,j+1},U^{n}_{i,j},\\&V^{n-1}_{i-1,j-1},V^{n-1}_{i-1,j},V^{n-1}_{i-1,j+1},V^{n-1}_{i,j-1},V^{n-1}_{i,j},V^{n-1}_{i,j+1},V^{n-1}_{i+1,j-1},V^{n-1}_{i+1,j},V^{n-1}_{i+1,j+1},V^{n}_{i,j}\}.\end{aligned}\tag{3.3}$$

Figs. 21 and 22 show the results obtained by means of this approach on a 400×400 uniform grid and $c=4$. Note that for the second equation (3.2), the larger diffusion coefficient may require a smaller time step than the first equation (3.1). The second equation of (3.2) is thus computed in two steps, using the time step $\frac{\Delta t}{2}$ for each step.

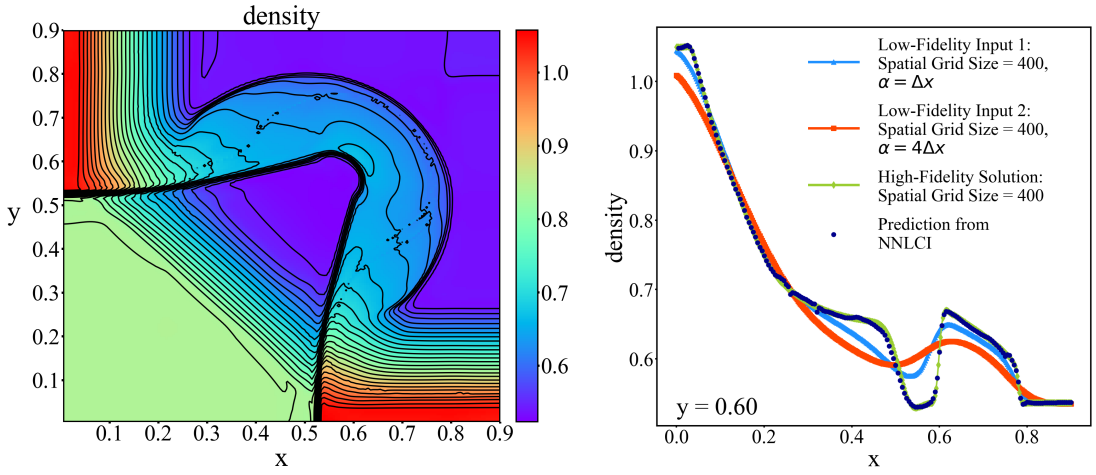

Figure 22: NNLCI prediction of final-time $(t=0.25)$ density distribution of 2D Euler system with initial value $+5\%$ perturbed from that of Case 8 (see Table 1), and axial distribution of density (dark blue) along $y=0.60$, compared to low-fidelity input solutions (blue and red) from leapfrog and diffusion splitting schemes (3.1) and (3.2) $(c=4)$, respectively on the 400×400 grid, and reference solution (green).

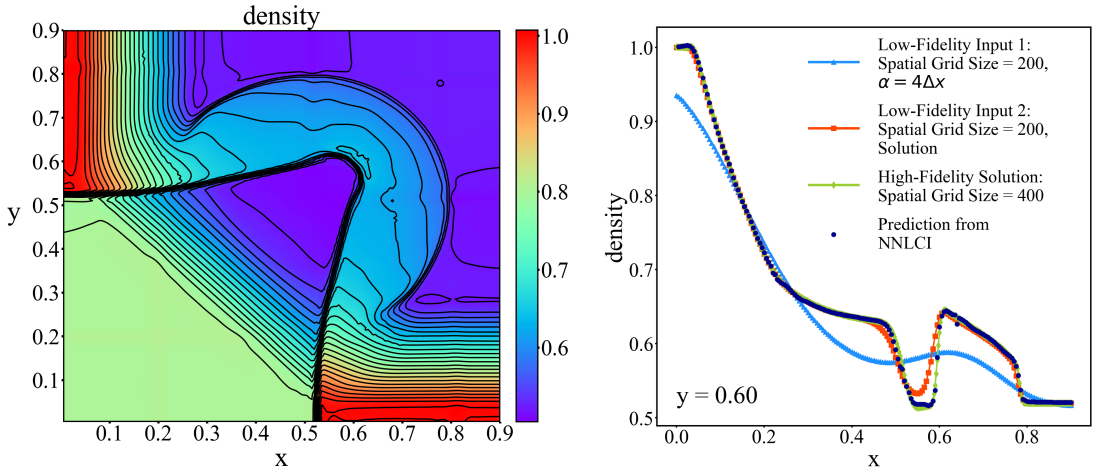

Figure 23: NNLCI prediction of final-time $(t=0.25)$ density distribution of Case 8 (see Table 1), and axial distribution of density (dark blue) along $y=0.60$, compared to low-fidelity input solutions (blue and red) from leapfrog and diffusion splitting scheme (2.9) $(\alpha=4\Delta x)$ on 400×400 grid and 4th-order scheme on 200×200 grid, respectively, and reference solution (green).

The input for NNLCI can also be provided by means of numerical schemes of different orders of accuracy [10]. For example, the first part of the input can be computed using a first-order leapfrog and diffusion splitting scheme (2.9) on 400×400 grid with $\alpha=4\Delta x$,

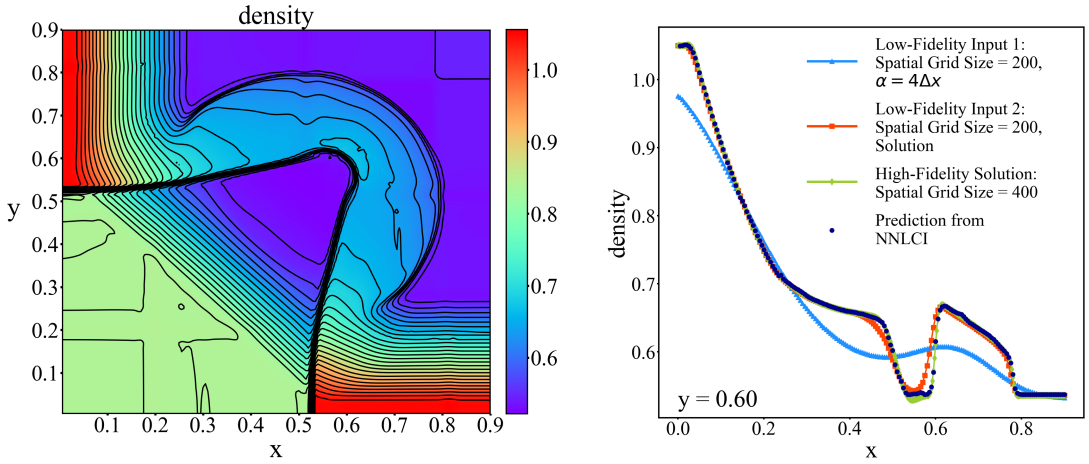

Figure 24: NNLCI prediction of final-time $(t=0.25)$ density distribution of 2D Euler system with initial value $+5\%$ perturbed from that of Case 8 (see Table 1), and axial distribution of density (dark blue) along $y=0.60$, compared to low-fidelity input solutions (blue and red) by leapfrog and diffusion splitting scheme (2.9) $(\alpha=4\Delta x)$ on 400×400 grid and 4th-order scheme on 200×200 grid, respectively, and reference solution (green).

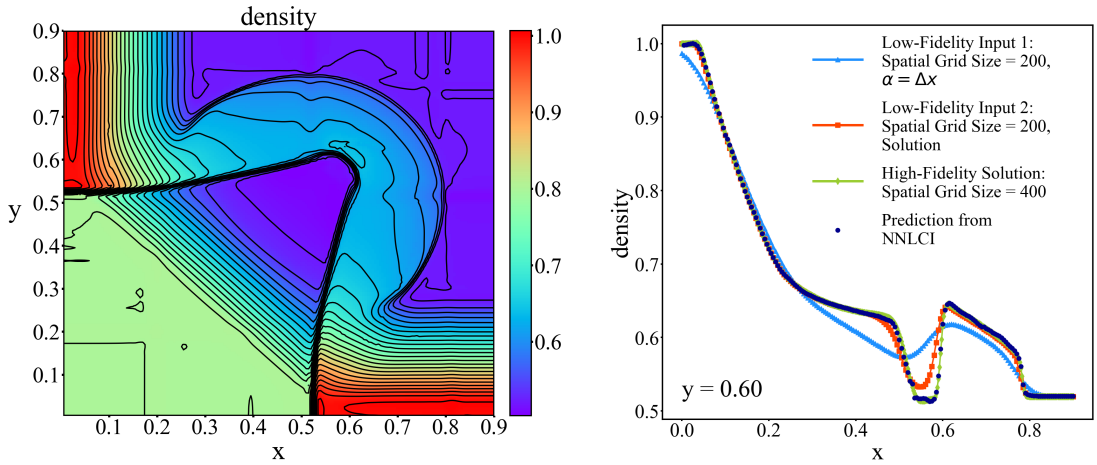

Figure 25: NNLCI prediction of final-time $(t=0.25)$ density distribution of Case 8 (see Table 1), and axial distribution of density (dark blue) along $y=0.60$, compared to low-fidelity input solutions (blue and red) from leapfrog and diffusion splitting scheme (2.9) $(\alpha=\Delta x)$ on 400×400 grid and 4th-order scheme on 200×200 grid, respectively, and reference solution (green).

and the second part using a 4th-order scheme on a 200×200 grid. The overall input is formatted on the 200×200 grid. Figs. 23 and 24 show the prediction from this approach.

The first part of the input to NNLCI can be slightly improved by using $\alpha=\Delta x$ in (2.9) on 400×400 grid. Figs. 25 and 26 show improved predictions from this minor adjustment.

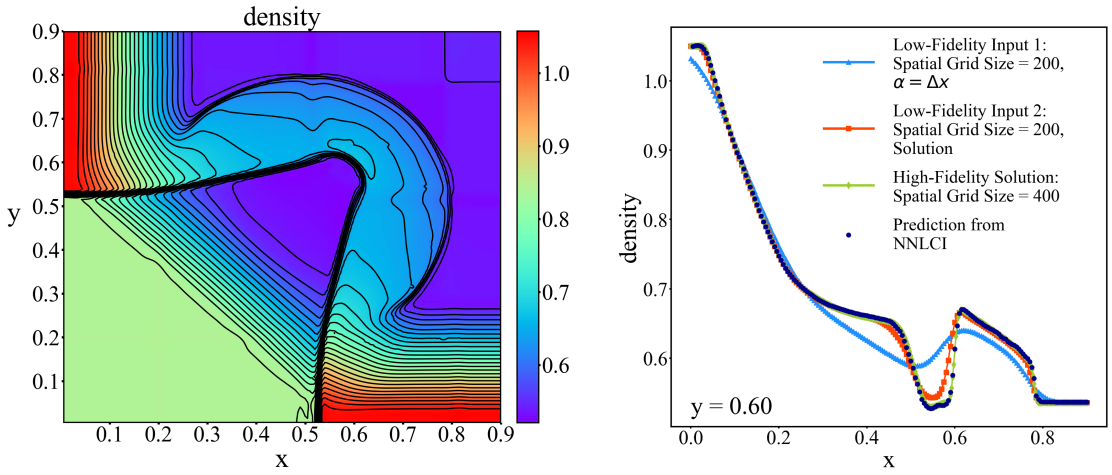

Figure 26: NNLCI prediction of final-time $(t=0.25)$ density distribution of 2D Euler system with initial value $+5\%$ perturbed from that of Case 8 (see Table 1), and axial distribution of density (dark blue) along $y=0.60$, compared to low-fidelity input solutions (blue and red) from leapfrog and diffusion splitting scheme (2.9) $(\alpha=\Delta x)$ on 400×400 grid and 4th-order scheme on 200×200 grid, respectively, and reference solution (green).

Table 4: Relative l_2 of NNLCI predictions with low-fidelity inputs computed by (1) leapfrog and diffusion splitting scheme with diffusion coefficients Δx and $4\Delta x$ respectively on 400×400 grid; (2) leapfrog and diffusion splitting scheme with diffusion coefficient $4\Delta x$ on 400×400 grid and a 4th-order scheme on 200×200 grid; (3) leapfrog and diffusion splitting scheme with diffusion coefficient Δx on 400×400 grid and a 4th-order scheme on 200×200 grid respectively.

	2-D Riemann Problem, Case 8		
NNLCI Input Method	**leapfrog and diffusion splitting scheme**	**leapfrog and diffusion splitting scheme with diffusion coefficient $4\Delta x$ and a 4th-order finite volume scheme**	**leapfrog and diffusion splitting scheme with diffusion coefficient Δx and a 4th-order finite volume scheme**
Initial Value	Relative l_2 Errors		
original	0.40E−2	0.31E−2	0.20E−2
+3%	0.32E−2	0.20E−2	0.20E−2
−3%	0.30E−2	0.26E−2	0.19E−2
+5%	0.30E−2	0.18E−2	0.19E−2
−5%	0.31E−2	0.22E−2	0.20E−2

The performance of the NNLCI approach using different inputs was assessed. Table 4 summarizes the relative l_2 errors in comparison with high-fidelity reference solutions.

Among prediction results shown in Table 4, the best one is seen with inputs computed by the leapfrog and diffusion splitting scheme (2.9) with diffusion coefficient Δx and a 4th-order finite volume scheme [17].

4 Conclusion

We have extended the neural network method presented in previous work [10] to two-dimensional conservation laws. The method was validated against several Riemann problems and demonstrated superior performance, robustness, and accuracy. The predicted results, even with smeared input profiles, agree with high-fidelity solutions for the entire domain, including regions with abrupt changes of state, such as shock waves and contact discontinuities. Further, the method is not sensitive to the low-cost schemes used to compute inputs. It can also accommodate inputs calculated with different numerical schemes and grids. For the two-dimensional Riemann problems considered in the present study, the computational savings of the NNLCI method is between one and two orders of magnitude compared with the corresponding high-fidelity solution schemes. The computational efficiency advantage is expected to be substantially greater for problems with higher dimensions or smooth solutions. Future work will include extension of the NNLCI method to three-dimensional conservation systems or other complex systems.

References

[1] ABADI, M., AGARWAL, A., AND P. BARHAM, E. A. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv: 1603.04467* (2016).

[2] ABGRALL, R., AND VEIGA, M. Neural network-based limiter with transfer learning. *Communications on Applied Mathematics and Computation* (2020), 1–41.

[3] BAYDIN, A. G., PEARLMUTTER, B. A., RADUL, A. A., AND SISKIND, J. M. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research 18*, 1 (2017), 5595–5637.

[4] BIANCHINI, S., AND BRESSAN, A. Vanishing viscosity solutions of nonlinear hyperbolic systems. *Annals of Mathematics* (2005), 223–342.

[5] CHEN, T., AND CHEN, H. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks 6*, 4 (1995), 911–917.

[6] DISCACCIATI, N., HESTHAVEN, J., AND RAY, D. Controlling oscillations in high-order discontinuous Galerkin schemes using artificial viscosity tuned by neural networks. *Journal of Computational Physics 409* (2020), 109304.

[7] E, W., AND YU, B. The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics 6* (2018), 1–12.

[8] HARTEN, A., ENGQUIST, B., OSHER, S., AND CHAKRAVARTHY, S. R. Uniformly high order accuracy essentially non-oscillatory schemes III. *Journal of Computational Physics 71*, 2 (1987), 231–303.

[9] HORNIK, K., STINCHCOMBE, M., AND WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 5 (1989), 359–366.
[10] HUANG, H., YANG, V., AND LIU, Y. Neural networks with local converging inputs (NNLCI) for solving conservation laws, part I: 1D problems. *Communications in Computational Physics (accepted)* (2023).
[11] JIANG, G.-S., AND SHU, C.-W. Efficient implementation of weighted ENO schemes. *Journal of Computational Physics 126* (1996), 202–228.
[12] LAX, P. Hyperbolic systems of conservation laws II. *Communications on Pure and Applied Mathematics, 10*, 4 (1957), 537–566.
[13] LAX, P., AND LIU, X. Solution of two-dimensional Riemann problems of gas dynamics by positive schemes. *SIAM Journal on Scientific Computing 19*, 2 (1998), 319–340.
[14] LI, Z., KOVACHKI, N., AZIZZADENESHELI, K., LIU, B., BHATTACHARYA, K., STUART, A., AND ANANDKUMAR, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895* (2020).
[15] LIU, D., AND WANG, Y. Multi-fidelity physics-constrained neural network and its application in materials modeling. *Journal of Mechanical Design 141*, 12 (2019), 121403.
[16] LIU, X. D., OSHER, S., AND CHAN, T. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics 115*, 1 (1994), 200–212.
[17] LIU, Y., SHU, C.-W., TADMOR, E., AND ZHANG, M.-P. Non-oscillatory hierarchical reconstruction for central and finite volume schemes. *Communications in Computational Physics 2* (2007), 933–963.
[18] LOU, Q., MENG, X., AND KARNIADAKIS, G. Physics-informed neural networks for solving forward and inverse flow problems via the Boltzmann-BGK formulation. *Journal of Computational Physics 447* (2021), 110676.
[19] LU, L., JIN, P., PANG, G., ZHANG, Z., AND KARNIADAKIS, G. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence 3*, 3 (2021), 218–229.
[20] MAGIERA, J., RAY, D., HESTHAVEN, J., AND ROHDE, C. Constraint-aware neural networks for Riemann problems. *Journal of Computational Physics 409* (2020), 109345.
[21] MAHMOUDABADBOZCHELOU, M., CAGGIONI, M., SHAHSAVARI, S., AND ET AL. Data-driven physics-informed constitutive metamodeling of complex fluids: A multifidelity neural network (MFNN) framework. *Journal of Rheology 65*, 2 (2021), 179–198.
[22] NGUYEN, H., AND TSAI, R. Numerical wave propagation aided by deep learning. *arXiv: 2107.13184* (2021).
[23] RAISSI, M., BABAEE, H., AND GIVI, P. Deep learning of turbulent scalar mixing. *Physical Review Fluids 4* (2019), 124501.
[24] RAISSI, M., PERDIKARIS, P., AND KARNIADAKIS, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics 378* (2019), 686–707.
[25] RAISSI, M., WANG, Z., TRIANTAFYLLOU, M., AND KARNIADAKIS, G. Deep learning of vortex-induced vibrations. *Journal of Fluid Mechanics 861* (2019), 119–137.
[26] RAISSI, M., YAZDANI, A., AND KARNIADAKIS, G. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science 367*, 6481 (2020), 1026–1030.
[27] SCHWANDER, L., RAY, D., AND HESTHAVEN, J. Controlling oscillations in spectral methods by local artificial viscosity governed by neural networks. *Journal of Computational Physics 431* (2021), 110144.
[28] SHU, C.-W., AND OSHER, S. Efficient implementation of essentially non-oscillatory shock-

capturing schemes. *Journal of Computational Physics 77* (1988), 439–471.

[29] SIRIGNANO, J., AND SPILIOPOULOS, K. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics 375* (2018), 1339–1364.

[30] VAN LEER, B. Towards the ultimate conservative difference scheme V, a second-order sequel to Godunov's method. *Journal of Computational Physics 32* (1979), 101–136.

[31] VON NEUMANN, J., AND RICHTMYER, R. D. A method for the numerical calculation of hydrodynamic shocks. *Journal of Applied Physics 21*, 3 (1950), 232–237.