Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2024-0260

Vol. **39**, No. 1, pp. 1-28
January 2026

First-Principle-Like Reinforcement Learning of Nonlinear Numerical Schemes for Conservation Laws

Hao-Chen Wang[1], Meilin Yu[2,*] and Heng Xiao[1]

[1]*Stuttgart Center for Simulation Science (SC SimTech), University of Stuttgart, Stuttgart, Germany.*
[2]*University of Maryland Baltimore County (UMBC), Baltimore, MD, USA.*

Received 24 October 2024; Accepted (in revised version) 9 July 2025

Abstract. In this study we present a universal nonlinear numerical scheme design method for nonlinear conservation laws, enabled by multi-agent reinforcement learning (MARL). Unlike contemporary approaches based on supervised learning or reinforcement learning, our method does not rely on reference data or empirical design. Instead, a first-principle-like approach using fundamental computational fluid dynamics (CFD) principles, including total variation diminishing (TVD) and k-exact reconstruction, is employed to design nonlinear numerical schemes. The third-order finite volume scheme is employed as the workhorse to test the performance of the MARL-based nonlinear numerical scheme design method. Numerical results demonstrate that the new MARL-based method can strike a balance between accuracy and numerical dissipation in nonlinear numerical scheme design, and outperforms the third-order MUSCL (Monotonic Upstream-centered Scheme for Conservation Laws) with the van Albada limiter for shock capturing. Furthermore, we demonstrate for the first time that a numerical scheme trained from one-dimensional (1D) Burgers' equation simulations can be directly used for numerical simulations of both 1D and 2D (two-dimensional constructions using the tensor product operation) Euler equations. The framework of the MARL-based numerical scheme design concepts can incorporate, in general, all types of numerical schemes as simulation machines.

AMS subject classifications: 65M08, 68T07, 76M12
Key words: Reinforcement learning, first-principle-like rewards, nonlinear numerical scheme, conservation law, shock capturing.

1 Introduction

Hyperbolic conservation laws, governed by nonlinear partial differential equations (PDEs), have extensive applications across numerous fields of science and engineer-

*Corresponding author. *Email addresses:* `haochen.wang@itlr.uni-stuttgart.de` (H.-C. Wang), `mlyu@umbc.edu` (M. Yu), `heng.xiao@simtech.uni-stuttgart.de` (H. Xiao) ing, such as aero-hydrodynamics, astrophysics, plasma physics, advanced manufacture and transportation engineering [1]. One feature of nonlinear hyperbolic conservation laws is that their solutions admit singularities (e.g., shock waves and contact discontinuity), which can be developed in finite time from smooth initial data. This poses grand challenges on numerical simulations as nonlinear numerical schemes need to be developed to take both scheme stability and numerical resolution into consideration; see Godunov's pioneering work on numerical methods for shock capturing [2]. As a result, many nonlinear numerical scheme construction methods have been developed during the last half century, such as high-resolution schemes with TVD slope limiters [3], weighted essentially non-oscillatory (WENO) methods [4–6], total variation bounded (TVB) discontinuous Galerkin methods [7], hierarchical multi-dimensional limiting process (MLP) [8, 9], moving discontinuous Galerkin finite element method with interface condition enforcement (MDG-ICE) [10, 11] and localized artificial viscosity and diffusivity methods [12–14], just to name a few. However, many nonlinear numerical schemes developed so far have to introduce empirical components, such as the use of limiter functions in the MUSCL scheme and smoothness indicators in the WENO schemes.

1.1 Supervised learning of shock capturing schemes

To reduce the dependence on empirical designs in numerical methods, researchers have increasingly turned to machine learning techniques to develop data-driven models. These models differ from traditional approaches by replacing many empirical components with neural networks. For instance, Ray et al. [15, 16] developed a data-driven troubled-cell indicator by training an artificial neural network (ANN) and tested it on 1D grids and 2D unstructured grids. Beck et al. [17] developed a data-driven shock indicator by using image-based edge detection methods on 2D grids. Bezgin et al. [18] developed a data-driven nonlinear weight function for WENO3. Numerical results showed that these data-driven models can perform better than the empirical ones and do not need problem-dependent parameter tuning. However, one common issue shared by the aforementioned works is that special numerical treatments, such as certain auxiliary equations and their corresponding analytical solutions, needs to be used to encode desired numerical features into the machine learning model. The choice of specific equations is still largely based on the authors' experience and analytical solutions may not be available.

Researchers have also leveraged the tool of machine learning to design new flux limiters [19] and learn discretizations for PDEs directly [20]. Nguyen et al. [19] designed a framework to derive an optimal flux limiter for the coarse-grained Burgers' equation by learning from high-resolution data. Numerical results demonstrated that the trained flux limiter achieves better results than standard limiters, but only in Burgers' equation simulations. Therefore, the model's generalizability to different physics is questionable. Bar et al. [20] designed a data-driven discretization method to learn the optimal approximations to PDEs on a coarse grid directly from the solutions on a finer grid. Numerical results demonstrated that their proposed method outperforms the standard numerical scheme on a coarse grid. However, the generalizability of the method was not discussed. To address the intrinsic limitations associated with supervised learning, reinforcement learning provides an alternative way to learn strategies to build nonlinear numerical schemes possibly without reference (labeled) data.

1.2 Reinforcement learning of shock capturing schemes

Reinforcement learning aims at optimizing a control policy by maximizing the cumulative (discounted/delayed) reward and has a broad range of applications ranging from games [21,22], robotics [23], disease treatment [24], and healthcare [25]. In computational fluid dynamics, reinforcement learning has been successfully applied to flow control [26], large eddy simulation (LES) [27], and wall-modeled LES [28]. In contrast to supervised learning, this approach does not require labeled data. Instead, agents in reinforcement learning continuously interact with the environment to generate samples from these interactions. In this manner, the algorithm incorporates the feedback of the environment into its decision-making process. This feature is of great importance because the long-term properties can be largely promised through the guide of a reward function, which is carefully designed using expert or domain knowledge [29].

Progress has been made to use reinforcement learning to design numerical schemes. Deep reinforcement learning was first used to learn WENO solvers for 1D scalar conservation laws via treating numerical PDE solvers as a Markov Decision Process (MDP) by Wang et al. [30]. In their study, the authors incorporated high-resolution solution data into the criteria for evaluating performance, known as the reward design. As a result, the agents were trained more or less in a supervised fashion. No generalization of the trained numerical scheme to more complex equations, such as Euler equations, was demonstrated. The same limitation applies to the follow-up works [31, 32]. Later, reinforcement learning was used to optimize the parameters of the fifth-order targeted ENO (TENO5) [33] for compressible flow simulations by Feng et al. [34]. They designed a reward function that aims to optimize numerical dissipation and dispersion of the learned TENO5 scheme with reference to the fifth-order WENO and high-order central schemes. However, the scheme must be retrained for each new equation, which restricts its generalizability.

Inspired by the idea of a-posteriori optimization [35], Beck and co-workers [36, 37] designed slope limiters for 1D and 2D second-order finite volume schemes by using reinforcement learning without using high-resolution data for model training. The novelty of their works lies in that they included reference limiters in the reward design to bound the permissive slope in the time-stepping process. Specifically, they designed the reward function by following empirical rules of flux limiter design: (1) penalize near zero or negative density or pressure, (2) penalize oscillatory solutions, (3) reward solutions that are already the most non-oscillatory (corresponding to solution from the Minmod limiter), and (4) detect the upwind direction in nonoscillatory solutions and utilize information derived from this. They have demonstrated that the trained limiter can be generalized to solve the same governing equations with different initial conditions and grid sizes. However, a possible limitation of this design is that it relies on specific types of existing flux limiters, and thus, the trained nonlinear numerical schemes may inherit empirical elements therein. As such, it is not clear how the learned scheme can be generalized to problems with different physics (e.g., from Burgers' equation to Euler equations) and spatial dimensions (e.g., from one dimension to two and three dimensions), while standard numerical schemes in CFD typically have such generalization capability. The machine-learned numerical schemes should achieve a similar universality as in the standard CFD schemes across physics (i.e., governing equations), grid, and dimensionality.

1.3 Objectives and contribution of the present work

To achieve the goal of significantly enhancing the generalization capability of machine-learned numerical schemes, in this work we present a universal, first-principle-like nonlinear numerical scheme design method enabled by multi-agent reinforcement learning. This MARL-based numerical scheme design framework uses neither reference solution data nor empirical flux limiter features in the reward design. Instead, a first-principle-like approach is adopted, where fundamental CFD principles are used to guide the agents in seeking a balance between accuracy and numerical dissipation automatically. Specifically, we design the reward function to promote stability by suppressing total variation increases while at the same time minimizing the modification of the baseline numerical scheme for smooth flow problems, such as the k-exact finite volume scheme adopted in this study. Note that the use of a limiter-free scheme as a reference in reward design is fundamentally different from referring to a particular limiter used in other works [37]. The trained nonlinear numerical scheme is largely free from ad hoc parameters and shows excellent generalizability over different physics, grid resolutions, and spatial dimensions. In doing so, numerical schemes learned from the 1D Burgers' equation are directly used to simulate both 1D and 2D Euler equations on grids with varying resolutions.

We chose to use the MUSCL scheme in our research because of its simplicity and wide applicability. This choice enables us to effectively showcase and validate the main features of the proposed MARL-based nonlinear numerical scheme design framework. Note that this reinforcement learning framework can be adapted to learn most, if not all, popular high-order numerical methods, such as discontinuous Galerkin, flux reconstruction, and WENO. The basic idea is to use the baseline high-order methods as reference in the reward design to ensure accuracy while to use bounded total variation to promote stability.

The rest of this paper is organized as follows. The MARL-based nonlinear numerical scheme is presented in Section 2. Therein, the design of the equation environment, agents, and reward in MARL, and the associated training framework are introduced. In Section 3, numerical properties of the MARL-based nonlinear numerical schemes are examined, and test results of model generalizability across different types of governing equations, initial conditions, grid resolutions, and problem dimensions are presented and discussed. Finally, conclusions of the present work and discussions of potential improvement of the MARL-based nonlinear numerical scheme design method are provided in Section 4.

2 Numerical methodology

2.1 Multi-agent reinforcement learning

Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. The major difference between reinforcement learning and supervised learning is that the former algorithm does not need labeled input/output pairs. At each discrete time step t, with a given state $\mathbf{s}$, the agent selects actions $\mathbf{a}$ with respect to its policy $\pi:\mathcal{S}\to\mathcal{A}$, receiving a reward r and the new state of the environment $\mathbf{s}'$. The return is defined as the discounted sum of rewards $R_t=\sum_{i=t}^{T}\gamma^{i-t}r(\mathbf{s}_i,\mathbf{a}_i)$, where γ is a discount factor determining the priority of short-term rewards and T is the final simulation time step. The focus of reinforcement learning is to find an optimal policy π_ψ that maximizes the expected return $J(\psi)=\mathbb{E}_{\tau\sim\pi}[R(\tau)]$ through exploration and exploitation, where ψ represents the parameters of the policy. There are three primary types of reinforcement learning algorithms: value-based methods, policy-based methods, and actor-critic methods.

Q-learning is a notable example of a value-based approach, recognized for its simplicity and popularity. In Q-learning, the algorithm has a function that calculates the quality of a state-action combination $Q:\mathcal{S}\times\mathcal{A}\to\mathbb{R}$. This quantity Q in the expression is usually called Q-value. Q-learning seeks to find a policy that maximizes the expected return $J(\psi)$ through iteratively updating the Q-function. The limitation of this algorithm is that it can only handle small-scale problems with a discrete action space, primarily due to its reliance on tabular methods. However, replacing the tabular methods with neural networks can help solve the issue of scalability and allows agents to make decisions from unstructured input data without the need for manual engineering of the state space [29].

In contrast, policy-based methods, such as the Policy Gradient algorithm, directly optimize the policy $\pi_\psi(\mathbf{a}|\mathbf{s})$ by adjusting parameters ψ in the direction that increases the expected return $J(\psi)$. The policy function, which can be replaced by the neural network architecture, is advantageous in environments with continuous action spaces. This flexibility allows the policy to handle a continuous range of actions naturally, making policy-based methods particularly suitable for such scenarios.

The actor-critic method is a hybrid approach that combines elements of both value-based and policy-based methods, leveraging the strengths of each [38]. In its framework, the 'actor' is responsible for selecting actions according to a policy $\pi_\psi(\mathbf{a}|\mathbf{s})$, which is parameterized by ψ. The 'critic', on the other hand, evaluates the chosen actions by estimating the value function, usually represented as the action-value function $Q^\pi(\mathbf{s},\mathbf{a})$. The critic's role is to provide feedback on the quality of the actions taken by the actor, helping to reduce variance in the policy gradient estimates and stabilize training. The actor updates the policy parameters in the direction suggested by the critic, typically using the advantage function to measure how much better or worse an action is compared to the expected value of the state.

In this work, we use Twin-Delayed Deep Deterministic Policy Gradient [39] (TD3), which is a variant of the modern actor-critic method, to train the agents in a centralized training decentralized execution (CTDE) manner. TD3 is particularly advantageous for problems involving continuous action spaces, making it well-suited for our application, which requires handling high-dimensional and continuous environments. Moreover, the algorithm is able to address the problem of overestimation of Q-value by introducing three numerical treatments: clipped double-Q learning, delayed policy updates, and target policy smoothing. More details about the algorithm can be found in [39].

To simplify the problem, all the agents share one single policy (i.e., the actor neural network). This simplification from multi-agent setting to single-agent setting is consistent with the traditional design of a numerical scheme. The reward design will be discussed in Section 2.3. In our implementation, the reinforcement learning processes are executed with the open-source machine learning framework PyTorch [40], while the environments for training and testing equations are set up using the open-source Python library Gym [41].

The proposed workflow of the MARL-based training framework is illustrated in Fig. 1. The reinforcement learning method adopted for learning numerical schemes consists of the following steps: (i) select the action based on the input states. (ii) interact with the equation environment to generate numerous transition tuples, which are composed of current states $\mathbf{s}$, current actions a, rewards r, and next states $\mathbf{s}'$. Then store these transition tuples in the replay buffer. (iii) samples are drawn from the replay buffer to first train and update the critic neural networks, followed by the training and update of the actor neural network.

2.2 MARL-based nonlinear numerical scheme

In this section, we describe how a nonlinear numerical scheme can be represented using a neural network. Our nonlinear numerical scheme builds upon the finite volume (FV) method with the MUSCL-type construction. For smooth problems (e.g., linear advection with a sinusoidal wave as the initial condition), the MUSCL scheme can achieve third-order accuracy by reconstructing the left state $u^L_{i+\frac{1}{2}}$ at the interface $i+\frac{1}{2}$, and the right state $u^R_{i-\frac{1}{2}}$ at the cell edge $i-\frac{1}{2}$ as follows:

$$u^L_{i+\frac{1}{2}}=\bar{u}_i+\frac{\phi_i}{4}\left[\left(1-\frac{1}{3}\right)(\bar{u}_i-\bar{u}_{i-1})+\left(1+\frac{1}{3}\right)(\bar{u}_{i+1}-\bar{u}_i)\right], \tag{2.1a}$$

$$u^R_{i-\frac{1}{2}}=\bar{u}_i-\frac{\phi_i}{4}\left[\left(1-\frac{1}{3}\right)(\bar{u}_{i+1}-\bar{u}_i)+\left(1+\frac{1}{3}\right)(\bar{u}_i-\bar{u}_{i-1})\right], \tag{2.1b}$$

where $\phi_i=1$ denotes the local parameter for the ith cell, and $\bar{u}$ is the cell-averaged value.

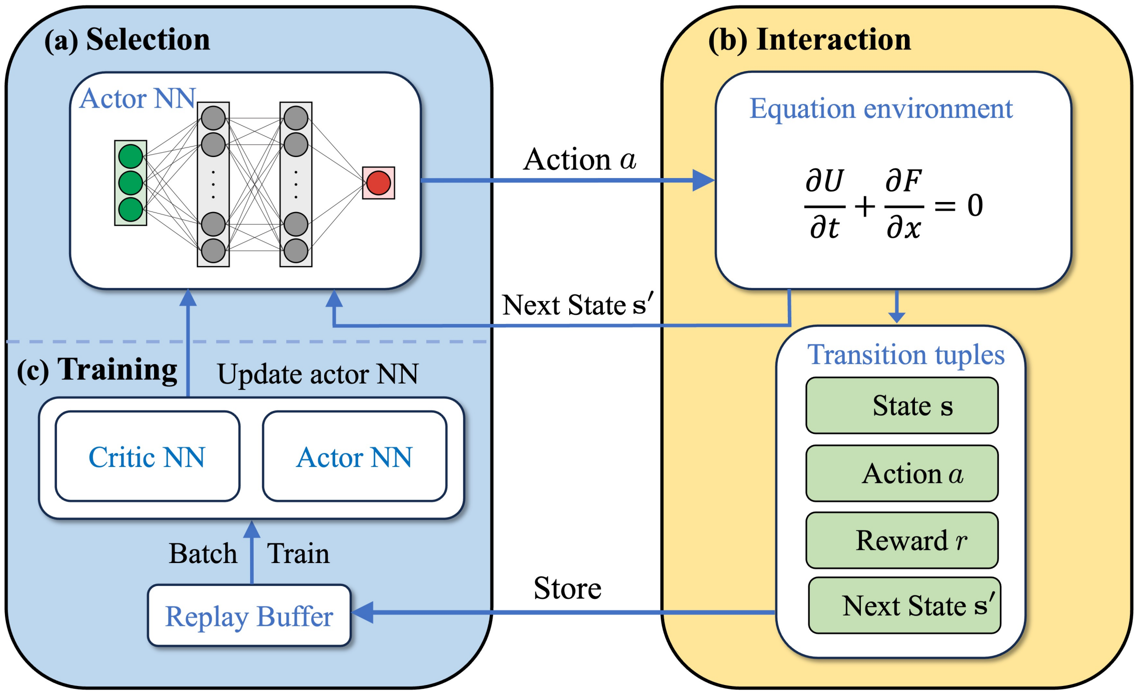


Figure 1: Workflow of the MARL-based training framework consisting of three steps: (a) input current states $\mathbf{s}$ to the actor neural network and output action a, (b) interact with the equation environment, generate numerous transition tuples, which are composed of current states $\mathbf{s}$, current actions a, rewards r, and next states $\mathbf{s}'$ and store transition tuples in the replay buffer, and (c) batch samples from the replay buffer to train and update neural networks.

However, for non-smooth problems, the optimal choice of ϕ_i remains unknown and may vary across different cells. Therefore, we define the action of ith agent, a_i, as a scalar value $\phi_i \in [-1,1]$. Agents are assigned to individual cells where this parameter must be determined. Once the local parameter ϕ_i has been assigned, we can use this value to calculate the left state $u^L_{i+\frac{1}{2}}$ and the right state $u^R_{i-\frac{1}{2}}$ within the ith cell. For a one-dimensional problem, the total number of agents is given by N_x, where N_x denotes the number of computational (physical) cells. The input state $\mathbf{s}_i$ for the ith agent is a three-cell stencil which comprises the cell-averaged values $\{\bar{u}_{i-1}, \bar{u}_i, \bar{u}_{i+1}\}$. Each agent's stencil is normalized independently using a variant of min-max normalization applied across its own stencil entries. Let the original stencil of agent i be denoted by $\mathbf{s}_i=[s_{i1},s_{i2},s_{i3}]$. The corresponding normalized stencil $\widehat{\mathbf{s}}_i=[\widehat{s}_{i1},\widehat{s}_{i2},\widehat{s}_{i3}]$ is computed as follows:

$$\widehat{s}_{ij}=\begin{cases}\dfrac{s_{ij}-\min\limits_k(s_{ik})}{\max\limits_k(s_{ik})-\min\limits_k(s_{ik})}, & \text{if } |\max_k(s_{ik})-\min_k(s_{ik})|>\epsilon_1,\\ 1, & \text{otherwise,}\end{cases} \quad \text{for } i=1,\cdots,N_x,\ j=1,2,3. \tag{2.2}$$

Here, s_{ij} denotes the jth value in the stencil of agent i, and the min and max are taken over all stencil positions $k=1,2,3$ for that agent. This ensures that each stencil $\widehat{\mathbf{s}}_i$ is scaled to the range $[0,1]$, while the small constant $\epsilon_1 = 10^{-8}$ prevents division by a near-zero denominator in the case of nearly constant stencils. For brevity, we omit the hat notation in the remainder of the paper, with the understanding that all stencil values are assumed to be normalized using this method. We treat nearly constant states that are caused by the numerical fluctuation as constant stencils and mark all the constant stencils. These are flagged accordingly, since varying the parameters within such stencils has negligible impact on the numerical simulation results.

Our current neural network–based parameterization employs a single parameter ϕ_i. When all agents choose $\phi_i = 1$, the scheme operates as a third-order MUSCL method, achieving high-order accuracy in smooth regions. Conversely, setting $\phi_i = 0$ reduces the scheme to the first-order Godunov method. This reduction to first-order accuracy near discontinuities is essential due to Godunov's theorem [2], which states any scheme that ensures monotone solutions can achieve at most first-order accuracy. Therefore, adapting the scheme to a first-order scheme in the vicinity of discontinuities ensures the preservation of monotonicity and prevents the introduction of non-physical oscillations.

We mention that learning ϕ_i as a function of the input states $\mathbf{s}$ is fundamentally different from the traditional empirical design of slope limiters, where the ratios of successive solution slopes serve both as shock sensors and as inputs to a predefined limiter function, such as van Albada limiter. In contrast, our multi-agent reinforcement learning approach employs a neural network that takes a three-cell stencil as its input. By leveraging reinforcement learning, we enable the model to autonomously identify and learn the optimal mappings from input states to corresponding actions. This strategy enhances the performance of the learned nonlinear numerical scheme, in terms of its generalizability, robustness, stability, and accuracy, as demonstrated in Section 3.

The algorithm of the MARL-based nonlinear numerical scheme design framework is shown in Algorithm 1 and the hyperparameters are listed in Table 1. The actor neural network (policy) π_ψ as well as two critic networks Q_{θ_1}, Q_{θ_2} all have 256 neurons in the hidden layers. There are also three corresponding target neural networks. A target neural network is a fixed neural network used as a stable reference during training in reinforcement learning to update the value estimates of another network, helping to stabilize learning. All six networks have two hidden layers and are connected with the activation function ReLU [42].

We briefly explain Algorithm 1 here. As a first step, the algorithm initializes two critic networks and one actor network, as well as three target networks, with random parameters. A replay buffer $\mathcal{B}$ is initialized for the storage of different transition tuples. In the first while loop, the environment is reset and normalized initial conditions of the equation are assigned as the input states $\mathbf{s}_i$ for each agent. In the second while loop, each agent selects a random action to encourage exploration of the environment. Then a joint action is formed and used to advance the environment by one time step using a user-defined time marching method. In this study, the second-order Runge–Kutta method is employed. The environment then outputs the joint normalized next state and a joint reward. A transition tuple $(\mathbf{s}_i,a_i,r_i,\mathbf{s}_i^{'})$ for each agent is stored in the replay buffer $\mathcal{B}$ afterwards. Then the algorithm batches N_B transitions from $\mathcal{B}$ to train the critic networks and the actor network. The target networks are updated every d steps. The actor neural network will output new action a_i' based on $\mathbf{s}_i{}'$. Then the iteration continues until reaching the designed stop time of the environment (**done** flag). After that, the environment resets and executes another training loop until meeting the final training step $n_{\max}$.

Algorithm 1 Reinforcement Learning Scheme (RLS) Framework

```
Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, and actor network $\pi_\psi$ with random parameters $\theta_1$,
$\theta_2$, $\psi$.
Initialize target networks $\ddot{\theta}_1 \leftarrow \theta_1$, $\ddot{\theta}_2 \leftarrow \theta_2$, $\ddot{\psi} \leftarrow \psi$ and replay buffer $\mathcal{B}$
while $n=0 \le n_{\max}$ do
    Reset the environment and initialize the input states $\mathbf{s}_i$ for each agent
    while not done do
        Select action with exploration noise $a_i \sim \text{clip}\left(\pi_\psi(\mathbf{s}_i)+\mathcal{N}(0,\sigma_1), a_{\text{low}}=-1, a_{\text{high}}=1\right)$
        Compute $u^L_{i+1/2}$ and $u^R_{i-1/2}$ from Eq. (2.1)
        Observe rewards $r_i$, new states $\mathbf{s}_i^{'}$ and termination signal done for each agent
        if $\mathbf{s}_i$ is not marked as constant stencil then
            Store transition tuples $(\mathbf{s}_i,a_i,r_i,\mathbf{s}_i^{'})$ in $\mathcal{B}$
        end if
        Sample mini-batch of $N_B$ transitions $(\mathbf{s}_i,a_i,r_i,\mathbf{s}_i^{'})$ from $\mathcal{B}$
        $\tilde{a} \leftarrow \text{clip}(\pi_{\ddot{\psi}}(\mathbf{s}')+\text{clip}(\mathcal{N}(0,\sigma_2),-c,c), a_{\text{low}}=-1, a_{\text{high}}=1)$
        $y \leftarrow r+\gamma \min_{k=1,2} Q_{\theta_k'}(\mathbf{s}',\tilde{a})$
        Update critics $\theta_k \leftarrow \text{argmin}_{\theta_k} {N_B}^{-1}\sum(y-Q_{\theta_k}(\mathbf{s},a))^2$
        if $n$ mod $d$ then
            Update $\psi$ by the deterministic policy gradient:
            $\nabla_\psi J(\psi)={N_B}^{-1}\sum \nabla_a Q_{\theta_1}(\mathbf{s},a)|_{a=\pi_\psi(\mathbf{s})}\nabla_\psi \pi_\psi(\mathbf{s})$
            Update target networks:
            $\ddot{\theta}_k \leftarrow \tau\theta_k+(1-\tau)\ddot{\theta}_k$
            $\ddot{\psi} \leftarrow \tau\psi+(1-\tau)\ddot{\psi}$
        end if
    end while
end while
```

2.3 Reward design

In reinforcement learning, the reward function plays a critical role in shaping the agent's ability to learn an effective control policy. In the context of designing numerical schemes, previous studies have often incorporated high-resolution data into the reward function.

Table 1: Hyperparameters of the reinforcement learning algorithm.

Hyperparameters	Implementation
Learning rate	2×10^{-4}
Optimizer	Adam [43]
Target update rate (τ)	5×10^{-3}
Batch size (N_B)	128
Discount factor (γ)	0.999
Normalized observation	True
Exploration policy noise (σ_1)	$\mathcal{N}(0,0.1)$
Policy update frequency (d)	2
Policy noise (σ_2)	$\mathcal{N}(0,0.001)$
Policy noise clip (c)	1×10^{-3}
Max training step ($n_{\max}$)	5×10^{5}

However, this approach introduces a more supervised learning process due to the reliance on labeled data during training. Such supervision can hinder the development of optimal schemes, especially when the best solution for scenarios like shocks on coarse grids is unknown. To address this limitation, we developed a reward function that does not require labeled data, maintaining the nature of reinforcement learning:

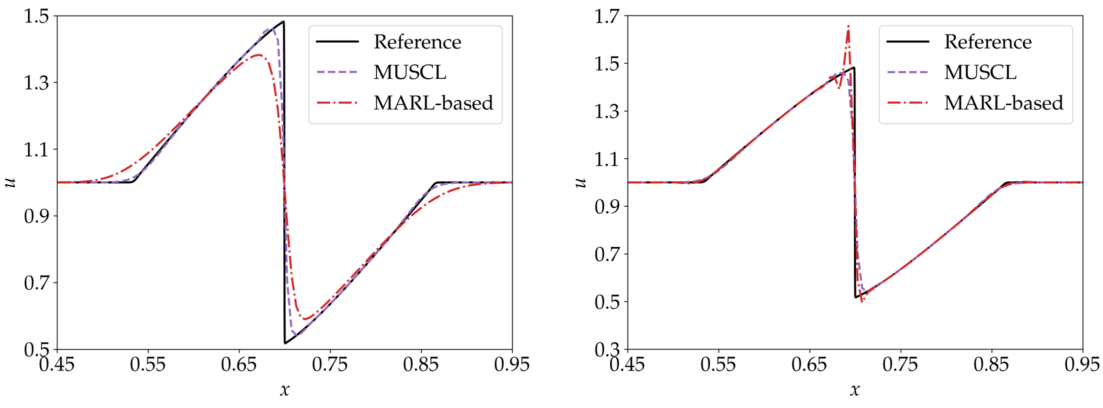
$$r_i^t(\mathbf{s}_i^{t+1},\mathbf{s}_i^t,a_i^t)=\alpha r_{\mathrm{D}}+r_{\mathrm{A}}, \tag{2.3a}$$

with

$$r_{\mathrm{D}}=\min(TV^t-TV^{t+1},0), \tag{2.3b}$$

$$r_{\mathrm{A}}=-\frac{1}{N_x}\sum_{i=1}^{N_x}||a_i-a_{\mathrm{ref}}||_{L_1}. \tag{2.3c}$$

Herein, the superscripts t and $t+1$ stand for the discrete time steps, and 'TV' stands for the total variation defined as $TV=\sum_{i=1}^{N_x}|f(x_{i+1})-f(x_i)|$ for a discrete function u_i where N_x is the total number of computational (physical) cells. A numerical method is said to be total variation diminishing (TVD) if the total variation of the solution u at time step $t+1$ is less than or equal to its total variation at time step t, that is $TV(u^{t+1})\leq TV(u^t)$. The concept of TVD, introduced by Harten [44], is a desired property for numerical schemes used for shock capturing. Thus, we design r_{D} based on the concept of TVD, which we regard as one of the 'first principles' of numerical scheme design for hyperbolic conservation laws. Indeed, r_{D} is a stability-promoting reward term, which suppresses the growth of total variation without actively promoting diffusion through a positive reward, i.e., r_{D} is always non-positive. If the total variation TV^{t+1} of the next state $\mathbf{s}^{t+1}$ is larger than the TV^t of the current state $\mathbf{s}^t$, a negative reward is distributed to agents who contribute to the increase of total variation. In Fig. 2a, we demonstrate the results of the trained scheme



(a) Results from stability-promoting scheme (b) Results from accuracy-promoting scheme

Figure 2: Numerical results of the 1D inviscid Burgers' equation with sinusoidal wave initial condition generated with (a) only the first part of the reward r_D being active, and (b) only the second part of the reward r_A being active.

using the 1D Burgers' equation with only the first part of the reward r_D being activated. The result is very dissipative compared to the reference solution because the algorithm chooses parameters which make the total variation diminish.

The accuracy-promoting reward term r_A measures in L_1-norm the average deviation of the trained coefficient in a selected high-order accurate numerical scheme from its reference value. In this study, the third-order MUSCL scheme is selected as the high-order method template to design the accuracy-promoting reward term r_A. As a result, r_A measures the deviation of the trained ϕ from its k-exact value in the MUSCL scheme, i.e., the reference value $a_{\text{ref}} \equiv \phi_{\text{ref}} = 1$ as presented in Eq. (2.1). In Fig. 2b, we demonstrate the results of the trained scheme using the 1D Burgers' equation with only the second part of the reward r_A being activated. There exist expected oscillations near the shock because the trained scheme can be considered as a third-order MUSCL scheme with low artificial numerical dissipation. This artificial numerical dissipation comes from the fact that the action cannot be exactly one since there exist error limits for the reinforcement learning algorithm. Note that for smooth problems without shock waves, the optimal value of ϕ trained with the MARL approach should be one. Therefore, we used the linear advection equation with a smooth wave profile to train the MARL-based scheme. As demonstrated in Appendix A, the convergence behavior of the smooth MARL-based scheme is the same as that of the third-order MUSCL scheme. This also verifies the code implementation. However, as will be discussed in Section 2.4, the order of accuracy that can be achieved by MARL-based schemes trained with the 1D Burgers' equation admitting shock waves can only be up to two when they are used to simulate smooth problems, such as the linear advection equation, as presented in Appendix A as well. This is also due to the artificial numerical dissipation 'memorized' by the reinforcement learning algorithm when it is used to train nonlinear schemes which can stabilize flow simulation admitting shock waves. Similar error limits were observed in a validation case, where the algorithm was trained in a smooth problem setting. Further details can be found in Appendix A.

In Eq. (2.3a), α is a function of the Courant–Friedrichs–Lewy (CFL) number, which is used to blend the two parts of stability-promoting and accuracy-promoting rewards. In this study, α is defined as $\alpha=\alpha_0/\mathrm{CFL}$, where α_0 is a constant. A key consideration to build α is to ensure that the reward can strike a good balance between adding numerical dissipation to ensure stability and maintaining high-order numerical scheme features to promote accuracy. Note that the maximum magnitude of r_{A} in our study is 2 when ϕ is -1. To match the order of magnitude of r_{A}, the term $\alpha r_{\mathrm{D}}=\alpha_0 r_{\mathrm{D}}/\mathrm{CFL}$ should have an order of magnitude of $\mathcal{O}(1)$. Through numerical experiments, we find that the machine-learned numerical schemes are not sensitive to the variation of α_0 when it varies between 1 and 100. Therefore, α_0 is set as 50 in this study. More details regarding the choice of the parameter α_0 are discussed in Appendix B.

By activating both components of the reward function, the training algorithm seeks to balance numerical dissipation with accuracy. Therefore, there is no need to reference high-resolution solution data or empirical flux limiter features. Instead, the reward function is developed using a first-principle-like approach grounded in fundamental CFD concepts. This design focuses on controlling the total variation of numerical solutions to ensure better numerical stability, while employing k-exact reconstruction parameters for smooth problems as a guide to enhance accuracy.

Although these fundamental concepts are not classified as first principles of physics, they constitute the most basic and irreducible laws governing the behavior of numerical schemes within a shock capturing finite volume framework. By applying these concepts, we ensure that the generated interaction samples accurately reflect the core physics in computational fluid dynamics. The term "first-principle-like" highlights the method's reliance on these essential CFD principles.

2.4 Training details

Due to the statistical nature of reinforcement learning, different choices of equation environment and initial condition will lead to different trained schemes. We selected the 1D inviscid Burgers' equation as our training environment not only because it is a fundamental scalar hyperbolic conservation law that can develop discontinuities in finite time, but also it aligns with the TVD constraint, which is applicable exclusively to scalar conservation laws. The Burgers' equation reads

$$\frac{\partial u}{\partial t}+\frac{\partial(u^2/2)}{\partial x}=0, \tag{2.4}$$

where u is the working variable. The sinusoidal wave initial condition with added noises is given as follows,

$$u=\begin{cases} 1+(\frac{1}{2}+\sigma_4)\sin\left(\frac{2\pi(x-(\frac{1}{3}-\sigma_3))}{\frac{1}{3}+2\sigma_3}\right), & \text{if } x\in[\frac{1}{3}-\sigma_3,\frac{2}{3}+\sigma_3], \\ 1, & \text{otherwise,} \end{cases} \tag{2.5}$$

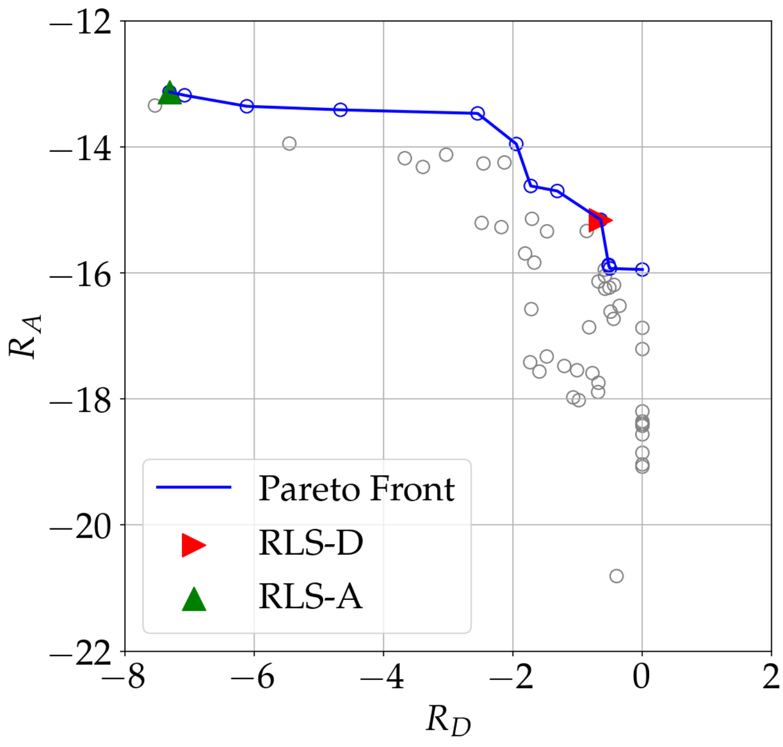


Figure 3: Scatter plot showing the distribution of the highest sixty rewards from three different training processes. The axes represent the cumulative rewards, with R_A being the y-axis and R_D being the x-axis. Each point corresponds to a specific training result, with RLS-D indicated by a red triangle and RLS-A by a green triangle. Blue points on the approximate Pareto front are examples of Pareto optimal choices.

where $\sigma_3 \sim \mathcal{U}(-0.1,0.1)$ and $\sigma_4 \sim \mathcal{U}(-0.2,0.2)$. σ_3 and σ_4 are parameters designed to control the shape of the initial sinusoidal wave. The grid size N_x is chosen from a pool consisting of 5 different grid sizes 100,200,400,600,800 for each simulation. The simulation end time is $T_{\text{end}} = 0.2$, and the CFL number is fixed at 0.2. During the simulation period, a shock will occur, resulting in a final solution that includes different components: discontinuity regions and smooth regions.

We use the proposed MARL-based design method Algorithm 1 to train nonlinear numerical schemes with 1D inviscid Burgers' equation. We identified two typical schemes that demonstrate different numerical properties. This distinction arises because a weighted sum was employed when building the reward function. This approach can result in varying patterns of reward distribution, even though the total rewards are approximately close. We plot the scatter distribution of highest sixty rewards from three different training processes and the corresponding Pareto front in Fig. 3. One notable pattern is that some points are concentrated near the line $R_D = 0$, while another pattern is observed with points aligning closely $R_A = -14$. Here, R_D and R_A represent the cumulative total rewards at the final time step, computed by summing the per-agent rewards r_D and r_A across all agents at each time step, and then accumulating these sums over time. Through testing in the same Burgers' equation environment and calculating the corresponding R_D and R_A, we identified the former pattern, which has an R_D value near zero, indicating dissipative properties. We selected a scheme from this pattern and named it 'RLS-D'. The latter pattern, which has a higher R_A, allows less dissipation near the shock. We also selected a scheme from this pattern and called it 'RLS-A'. Herein, 'RLS' stands for the acronym of reinforcement learning scheme. The convergence rate (i.e. order of accuracy) test results when they are used to simulate smooth problems, such as the linear advection equation with a smooth wave profile, are presented in Appendix A. There we observe that the less dissipative scheme 'RLS-A' can achieve a second-order convergence rate, while the dissipative scheme 'RLS-D' presents a first-order convergence rate.

3 Results and discussions

In this section, the numerical performance of MARL-based nonlinear numerical schemes is examined with several sets of shock-capturing problems governed by the 1D inviscid Burgers' equation, 1D Euler equations, and 2D Euler equations. They are all representative cases of hyperbolic conservation laws. The third-order MUSCL scheme with van Albada limiter is chosen as the baseline scheme. The MARL-based schemes presented here are trained solely using the training environment described in Section 2.4. Note that, once the training is complete, no exploration noise is added to the action a_i during execution. The generalizability of the trained 1D nonlinear numerical schemes is evaluated across diverse test cases involving varying physics, grid resolutions, and spatial dimensions.

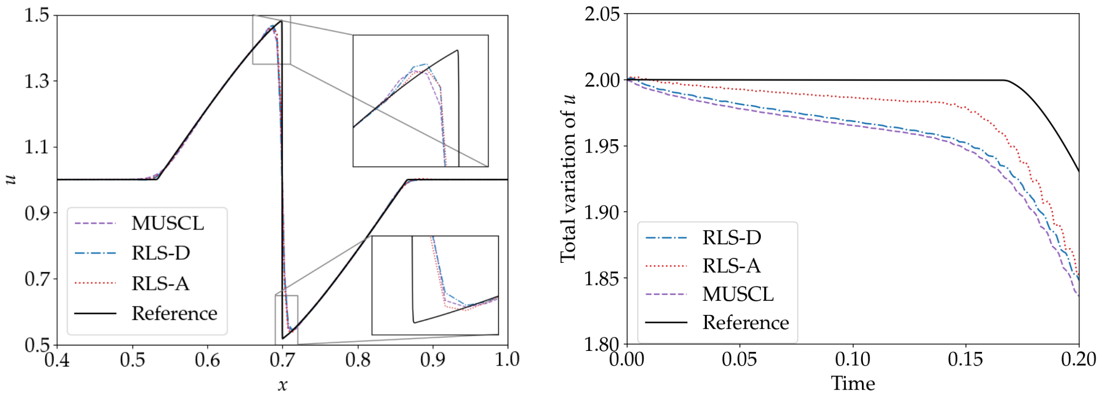
3.1 1D inviscid Burgers' equation

The first testing equation environment is built based on the training equation environment. The difference is that the initial sinusoidal wave condition is deterministic, which is given as follows,

$$u=\begin{cases} 1+\sin(6\pi(x-1/3))/2, & \text{if } 1/3\le x\le 2/3, \\ 1, & \text{if } x<1/3 \text{ or } 2/3<x. \end{cases} \tag{3.1}$$

The simulation parameter setting is given as follows: the number of cells within the computational domain is $N_x=200$, the simulation end time is $T_{\text{end}}=0.2$, and the CFL number is fixed at 0.2.

The results in Fig. 4a show that numerical results generated from the scheme have good agreement with reference solution data as the shock is well captured even though in reward design we did not include any high-resolution data. We also include the numerical result calculated from the third-order MUSCL scheme using van Albada limiter



(a) Numerical results (b) Total variation change

Figure 4: (a) Comparison of numerical simulation results of the 1D inviscid Burgers' equation with sinusoidal wave initial condition generated from RLS-D, RLS-A, the third-order MUSCL scheme with van Albada limiter and reference solution. (b) Total variation changes of each solutions.

with the same grid size as the baseline scheme for comparison. Note that the reference solution is calculated from the same MUSCL scheme but on a very fine mesh in which the number of cells is $N_x = 6400$. From Fig. 4a, we observe that RLS-A outperforms both RLS-D and the third-order MUSCL scheme with the limiter by more sharply capturing the shock structure. In Fig. 4b, it is clear that total variation is under good control throughout the entire simulation. Both RL-based schemes are able to maintain the TVD numerical property successfully. For RLS-A, before the shock formation, its total variation decreases much slower than RLS-D, which indicates that the scheme is able to choose actions that are less dissipative during the smooth wave propagation. After the shock is formed, the scheme switches its actions to a dissipative mode.

3.2 Generalization across different types of governing equations

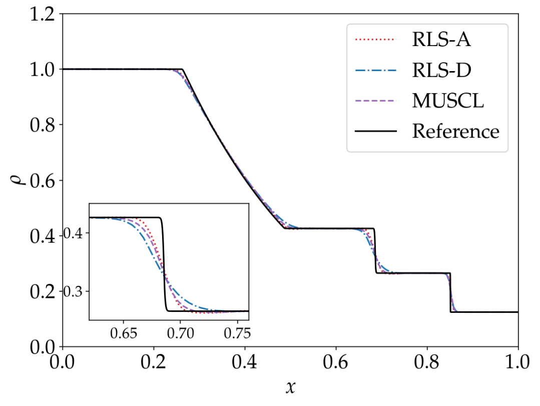
Euler equations are a set of equations that describe the flow of inviscid, compressible fluids. In 1D form, Euler equations for an ideal gas consist of three equations that govern the conservation of mass, momentum and energy. They are written in the vector format as:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0, \tag{3.2}$$

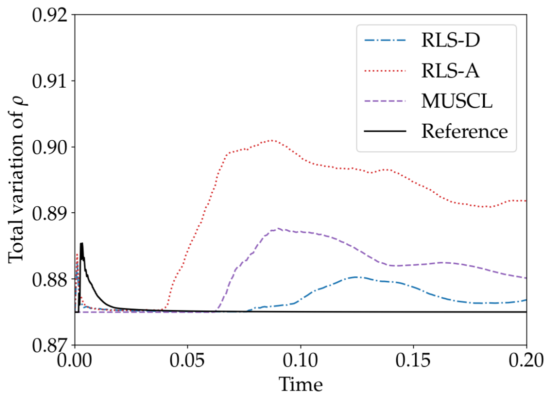
with

$$U = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}, \quad \text{and} \quad F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ u(E+p) \end{pmatrix}.$$

Herein, ρ is the density, u is the velocity, p is the pressure, and E is the total energy. The ideal gas law $E = p/(\gamma_g - 1) + \rho u^2/2$, where $\gamma_g = 1.4$ is the constant specific heat ratio, is



(a) Numerical results



(b) Total variation change

Figure 5: (a) Comparison of numerical simulation results of density ρ of the 1D Euler equations with Sod shock tube initial condition generated from RLS-D, RLS-A, the third-order MUSCL scheme with van Albada limiter and reference solution. (b) Total variation changes of each solutions.

used to close the system. The Sod shock tube problem is used to conduct tests in this section, and its initial condition is given as follows,

$$(\rho,u,p)=\begin{cases}(1,0,1), & x<0.5,\\(0.125,0,0.1), & x\geq 0.5.\end{cases} \tag{3.3}$$

The simulation parameter setting for this problem is the same as that of the Burgers' equation: $N_x=200$, $T_{\text{end}}=0.2$, and the CFL number is 0.2. The reference solution is calculated using the same MUSCL scheme on a finer grid with $N_x=6400$ cells.

To demonstrate the generalizability of MARL-based nonlinear numerical schemes to different physics, we use the same schemes trained from the 1D inviscid Burgers' equation to directly solve the 1D Euler equations. The trained policy is applied separately to three conservative variables to generate values on cell interfaces. In this study, we use the Local Lax-Friedrichs (i.e., Rusanov) method to solve the Riemann problem in the 1D Euler equations. We observe from Fig. 5a that without any prior knowledge of Euler equations or the assistance of high-resolution solution data, numerical results of density ρ from machine-learned schemes have good agreement with those from the MUSCL scheme on the same grid and reference solution. RLS-D underperforms the third-order MUSCL scheme in the contact discontinuity region, i.e., showing more dissipative results there, due to that the scheme prefers adding more numerical dissipation to stabilize the simulation. In contrast, RLS-A is capable of capturing the contact discontinuity more sharply than RLS-D and the third-order MUSCL scheme. In Fig. 5b, we observe that total variations of the density from RLS-A and RLS-D are under good control. Note that the total variation of the Sod shock tube problem should remain constant during the development of the rarefaction wave, contact discontinuity, and shock wave. Any total variation increase indicates the creation of local extremes. From an enlarged view near the end of the rarefaction wave (i.e., $x \approx 0.5$), and the contact discontinuity (i.e., $x \approx 0.7$) in Fig. 5a, we observe apparent, although very small, troughs from the density predicted by RLS-A. This is due to the low numerical dissipation nature of RLS-A, and explains why the total variation value of RLS-A is larger than that of the reference value.

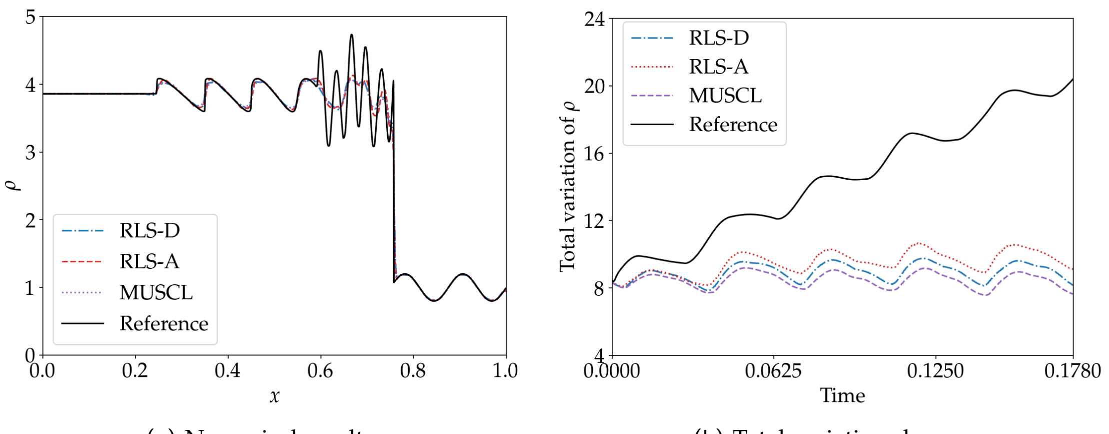
In summary, the good generalizability across Burgers' equation and Euler equations demonstrated in this section is consistent with standard CFD schemes, and has not been demonstrated in previous reinforcement learning-based works.

3.3 Generalization across different initial conditions

In this section, the same scheme trained from the 1D Burgers' equation is tested with the 1D Shu-Osher problem where a shock wave interacts with an entropy wave. Its initial conditions are given as:

$$(\rho,u,p)=\begin{cases} (3.857143,2.629369,10.3333), & x<1/8, \\ (1+0.2\sin(16\pi x),0,1), & x\geq 1/8. \end{cases} \tag{3.4}$$

The computational domain is $[0,1]$, and a grid with $N_x=400$ cells is used to conduct simulations. The CFL number is set as 0.1 and the end time is set to $T_{\text{end}}=0.178$. The reference solution is calculated using the baseline MUSCL scheme on a finer grid with 6400 cells. Similar to the Sod shock tube test presented in Section 3.2, we observe in Fig. 6a that RLS-A outperforms RLS-D and MUSCL. For all schemes, the amplitudes of the high-frequency waves are under-predicted compared to the reference solution. Total variations of the density from all tests are presented in Fig. 6b. Note that in the shock-entropy



(a) Numerical results (b) Total variation change

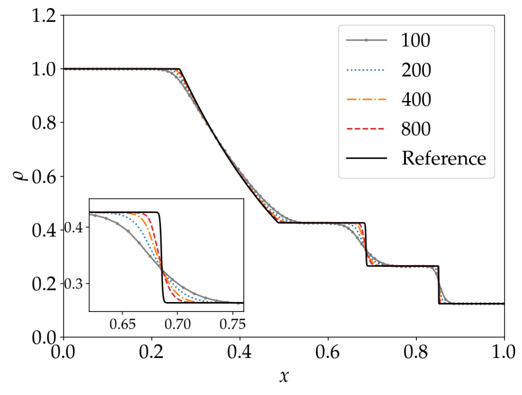
Figure 6: (a) Comparison of numerical simulation results of density ρ of the 1D Euler equations with Shu-Osher shock tube initial condition generated from RLS-D, RLS-A, the third-order MUSCL scheme with van Albada limiter and reference solution. (b) Total variation changes of each solutions.

wave interaction, new extremes can be created. As a result, the total variation of the conserved variables can increase (see the reference solution). Compared to those of RLS-D and MUSCL, the time history of total variation of RLS-A demonstrates a closer alignment with the reference solution, effectively capturing the underlying trend from the reference one.

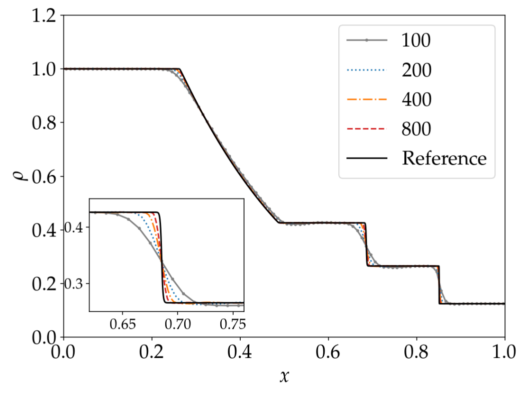
3.4 Generalization across varying grid resolutions

In this section, we present the numerical results of grid refinement studies for the Sod problem and Shu-Osher problem to test the generalizability of the MARL-based schemes across varying grid resolutions. Specifically, the grid is refined from a coarse one with 100 cells to a fine one with 800 cells.

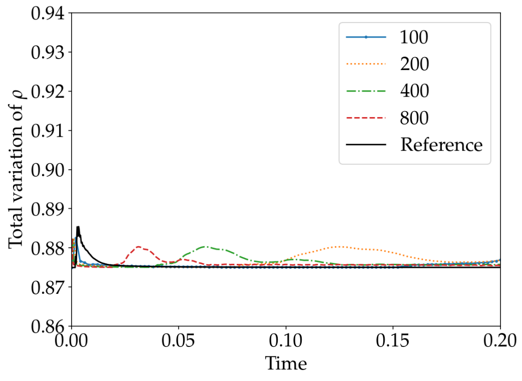
We observe in Fig. 7 that when the grid is refined, density fields of the Sod shock



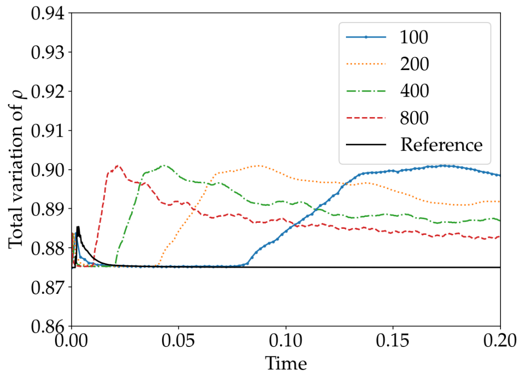
(a) Density RLS-D



(b) Density RLS-A



(c) TV RLS-D



(d) TV RLS-A

Figure 7: Grid refinement study results of the 1D Euler equations with the Sod shock tube initial condition. (a) Density fields for RLS-D, (b) density fields for RLS-A, (c) total variation evolution histories for RLS-D, and (d) total variation evolution histories for RLS-A are presented here for comparison.

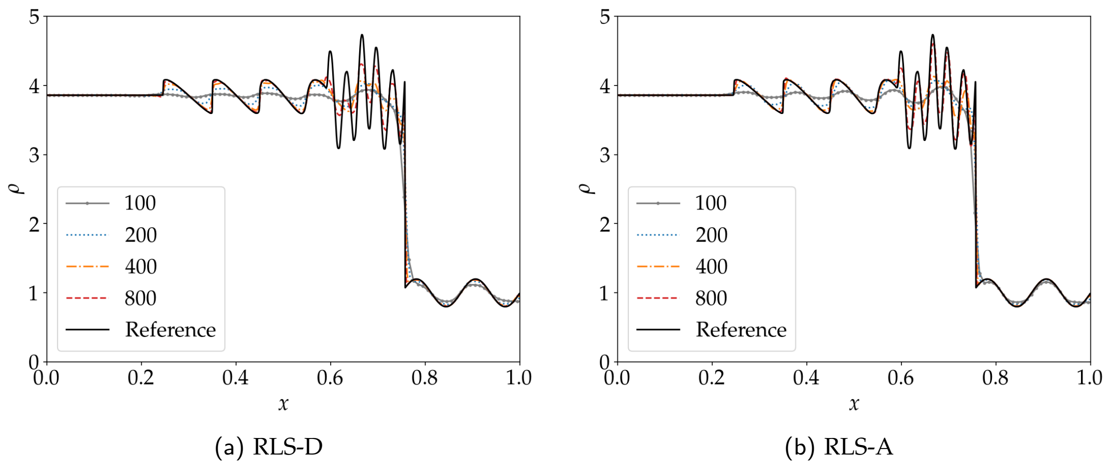


Figure 8: Numerical results of density ρ of the 1D Euler equations with Shu-Osher shock tube initial condition under different mesh sizes generated from (a) RLS-D and (b) RLS-A.

tube problem from both RLS schemes become more resolved and accurate. As expected, RLS-A performs better than RLS-D as it can capture the rarefaction wave, contact discontinuity, and shock wave with sharper resolutions. We observe from Fig. 7a that RLS-D can sharply capture the shock wave when the grid is refined to 400 cells. However, even with 800 cells, RLS-D cannot sharply capture the contact discontinuity. Instead, RLS-A with 800 cells can almost achieve the resolution of the reference solution with 6400 cells. We find from Fig. 7c that the total variation evolution histories of RLS-D with different grid resolutions do not show large differences when the wave propagation reaches a relatively stable stage, i.e., rarefaction, contact discontinuity, and shock have been sufficiently separated from each other. From Fig. 7d, we observe that as the grid is refined, the total variation for $t > 0.1$ increasingly aligns with the reference solution, although noticeable discrepancies persist during the initial stages of the computation. This trend corresponds well to the decreasing local extreme values near the end of the rarefaction wave and the contact discontinuity as observed in Fig. 7b.

In the Shu-Osher problem test, as shown in Fig. 8a, RLS-D is not able to predict the high-frequency waves caused by shock-entropy wave interaction even with 800 cells. Instead, RLS-A captures the high-frequency waves more sharply when 800 cells are used (see Fig. 8b).

3.5 Generalization across different problem dimensions

In this section, the same schemes trained from the 1D Burgers' equation are used to solve the 2D Euler equations with tensor-product-based solution and flux constructions. In the two-dimensional form, Euler equations consist of four equations that govern the conser-

Table 2: Initial conditions in different quadrants for Case 1 and Case 2.

IC1	$p_2=1, \rho_2=1, u_2=0.7276, v_2=0$	$p_1=0.4, \rho_1=0.5313, u_1=0, v_1=0$
	$p_3=1, \rho_3=0.8, u_3=0, v_3=0$	$p_4=1, \rho_4=1, u_4=0, v_4=0.7276$
IC2	$p_2=1, \rho_2=2, u_2=0.75, v_2=0.5$	$p_1=1, \rho_1=1, u_1=0.75, v_1=-0.5$
	$p_3=1, \rho_3=1, u_3=-0.75, v_3=0.5$	$p_4=1, \rho_4=3, u_4=-0.75, v_4=-0.5$

vation of mass, momentum and energy. This reads

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}=0, \tag{3.5}$$

with

$$U=\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ E\end{pmatrix},\quad F=\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ u(E+p)\end{pmatrix},\quad \text{and}\quad G=\begin{pmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ v(E+p)\end{pmatrix}.$$

Herein, ρ is the density, u and v are the velocity components in the x and y directions, respectively, p is the pressure, and E is the total energy. The system is closed with the perfect gas law.

Two 2D Riemann problems are simulated. The computational domain $[0,1]\times[0,1]$ is divided into four quadrants centered at $(0.5,0.5)$. The initial conditions of p, ρ, u, and v in different quadrants for the two problems are given in Table 2.

The grid consists of 200×200 cells in the x and y directions, respectively. The CFL number is 0.2, and the simulation end time is $T_{\text{end}}=0.25$ for Case 1 and $T_{\text{end}}=0.3$ for Case 2. In the tensor-product-based approach, directional spatial splitting is applied in each spatial dimension and the method of lines is used to march the resulting system of ordinary differential equations in time. Thus, the 1D numerical schemes trained from Burgers' equation can be directly used in the 2D implementation. As in previous tests, the second-order Runge–Kutta method is used to march the system in time.

We observe from both Figs. 9 and 10 that simulation results from the trained numerical schemes RLS-A and RLS-D agree reasonably well with those calculated from the baseline third-order MUSCL scheme. The results generated from RLS-A show sharper resolutions of flow discontinuities and vortex structures. Instead, RLS-D shows numerical dissipation features like the baseline scheme.

3.6 Discussion of the statistical nature of the MARL-based framework

Despite promising performance of the MARL-based framework, a certain degree of randomness is inherent in the actions chosen for cells with shocks. In these cells, r_{A} and r_{D}

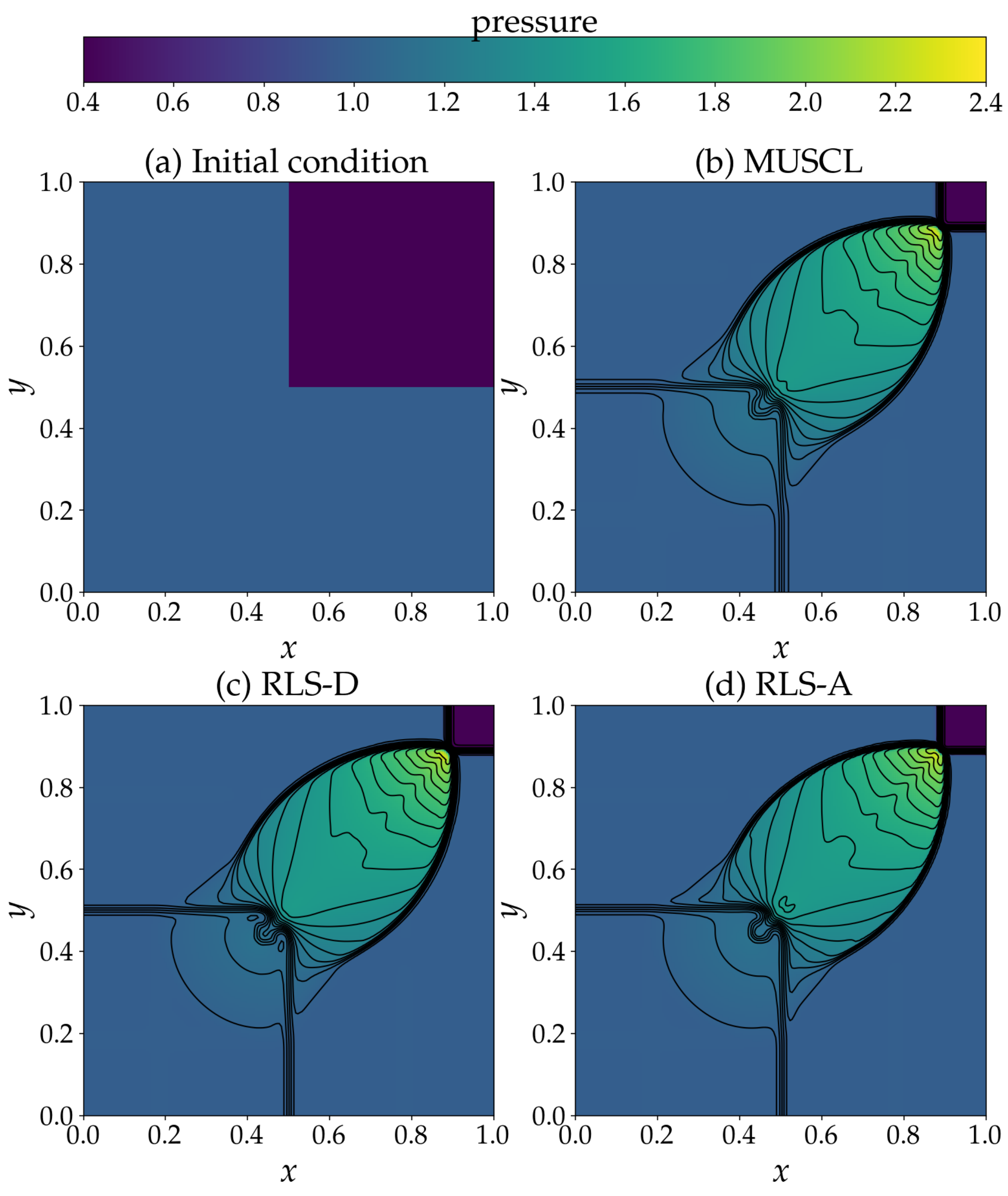


Figure 9: Numerical results of 2D Euler equations with Case 1 (a) initial condition generated from (b) the third-order MUSCL scheme with van Albada limiter and (c) RLS-D, (d) RLS-A. Pressure is displayed by color and density by 30 contours (0.54 to 1.7 step 0.04) [45].

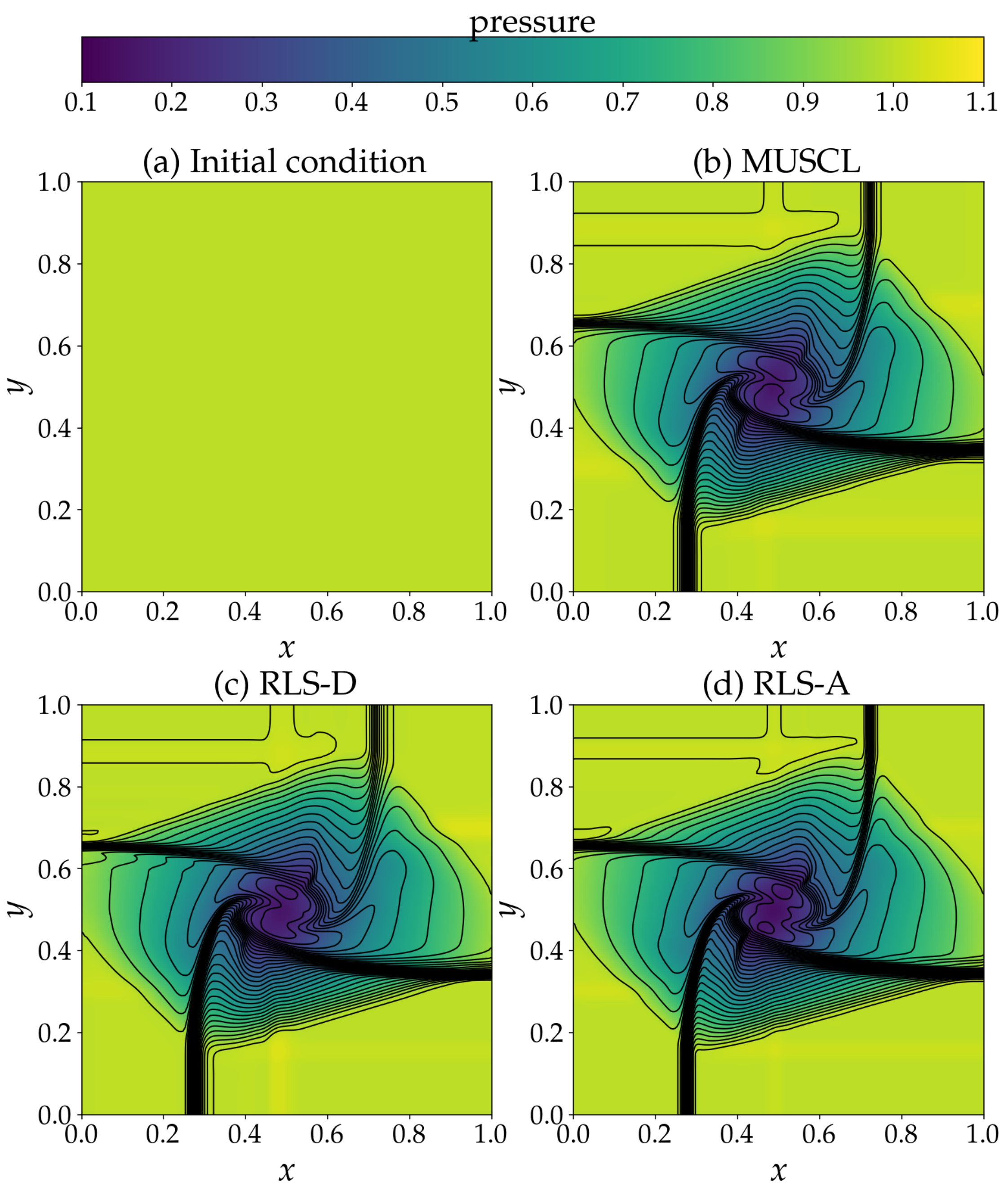


Figure 10: Numerical results of 2D Euler equations with Case 2 (a) initial condition generated from (b) the third-order MUSCL scheme with van Albada limiter and (c) RLS-D, (d) RLS-A. Pressure is displayed by color and density by 29 contours (0.25 to 3.05 step 0.1).

naturally become contradictory since choosing $\phi=1$ will lead to the increase of total variation. However, reward r_{A} does not serve as an exact accuracy measurement; instead it serves more as an indicator of non-smoothness. In the "troubled" cells, the agents will only learn not to pick action $\phi=1$ because the best action for those cells remains unknown. This fact indeed provides the trained schemes with a certain level of randomness (see discussions in Section 2.4). However, through numerical experiments, we found that as long as the return in Eq. (2.3a) of the schemes remains high, most of the schemes exhibit desirable numerical properties as designed: suppress spurious oscillations near discontinuities and maintain high order of accuracy in smooth regions.

4 Conclusions

This work presents a universal, first-principle-like MARL-based framework to design nonlinear numerical schemes for general hyperbolic conservation laws. Unlike existing reinforcement learning-based research, our reward design does not incorporate any reference data or empirical elements (i.e., flux limiters and artificial viscosity). Instead, the reward function is formulated based on a first-principle-like approach using fundamental CFD theory. Specifically, numerical stability is achieved by controlling the total variation of numerical solutions, while numerical accuracy is promoted by using k-exact polynomial reconstruction of solution variables. The agents are able to strike a balance between accuracy and numerical dissipation in the learned numerical schemes through iterative interactions with the governing equations (i.e., environment in reinforcement learning) and exploitation of many interaction samples. The one-dimensional inviscid Burgers' equation is employed to train the MARL-based nonlinear numerical schemes. The generalizability of the learned schemes is then tested using both 1D and 2D Euler equations simulations that admit shock waves under different flow conditions and grid resolutions. The MARL-based schemes are able to obtain comparable results to those using the third-order MUSCL scheme equipped with van Albada limiter, while the learned scheme with low numerical dissipation (i.e., RLS-A) outperforms the MUSCL scheme. These promising numerical simulation results demonstrate a new paradigm for designing nonlinear numerical schemes using reinforcement learning with first-principle-like rewards.

We also carried out 1D Leblanc shock tube tests under extreme flow conditions (results are not shown here). A key observation is that when the interaction samples used in reinforcement learning favor certain reward criteria, e.g., the TVD property for scheme stability due to harsh flow conditions, the learned numerical scheme can become biased. For example, the scheme is very dissipative when scheme stability is overly emphasized. There is no easy fix for such issues, as interaction samples favoring other reward criteria, e.g., k-exact reconstruction for scheme accuracy in the Leblanc problem, are rare. To address this issue, post-processing the learned numerical scheme using domain knowledge provides an alternative solution. Since it is known that negative density and internal energy can be easily generated when simulating the Leblanc problem, accuracy-preserving positivity limiting procedures, such as those developed by Zhang and Shu [46], can be employed to regulate the solution behavior. This is still under active research.

Acknowledgments

HW and HX are funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2075 — 390740016. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech).

A Convergence tests with smooth problems

To verify the code implementation, we conducted a series of error convergence tests using the linear advection equation with a smooth wave profile, where the optimal value of ϕ is known to be one. The trained scheme, referred to as the *validation scheme*, successfully demonstrated third-order accuracy, confirming the implementation's validity.

The experimental setup is as follows: the agents are trained on a smooth test case: the one-dimensional linear advection equation with a pure sinusoidal wave as the initial condition. The equation reads:

$$\frac{\partial u}{\partial t}+a\frac{\partial(u)}{\partial x}=0, \tag{A.1}$$

where u is the working variable of advection equation and $a=1$ is the advection speed. The initial condition is given as follows,

$$u(x)=\sin(2\pi x), \quad x\in[0,1]. \tag{A.2}$$

We evaluate the trained scheme alongside the third-order and second-order MUSCL schemes, as well as the RLS-A and RLS-D methods on five different grid resolutions and plot the resulting errors as a function of grid size in Fig. 11. As shown in the figure, the *validation scheme* successfully achieves third-order accuracy for the smooth test problem. This outcome is consistent with our expectation that, for a smooth solution, selecting $\phi=1$ should not increase the total variation of the numerical solution. Nonetheless, due to inherent limitations in machine learning-based methods, a slight degradation in the observed order of accuracy is evident on finer grids, reflecting the presence of an error limit. While RLS-A and RLS-D can also solve the linear advection equation, their accuracy is limited due to the inability to disable the built-in "limiter" effect in the machine learning models. RLS-A achieved a second-order convergence rate, whereas the convergence rate of RLS-D was more toward the first order due to stronger numerical dissipation built into the scheme.

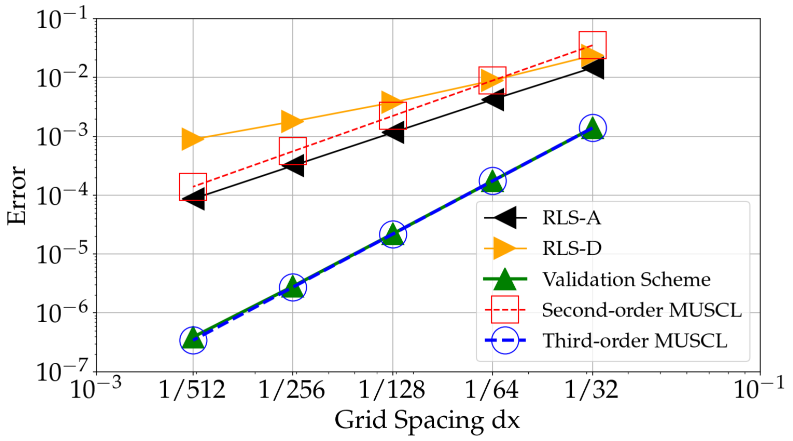


Figure 11: Comparison of order of accuracy from the pure second-order and third-order MUSCL scheme, the validation scheme trained in an environment with smooth problem setting, RLS-D and RLS-A.

B Parameter selection in reward design

The idea behind constructing the blending function α in Eq. (2.3a) is that the algorithm needs to reach a good balance between numerical dissipation and accuracy. Specifically, αr_{D} should have a similar order of magnitude to that of r_{A}. From numerical experiments, we indeed observed that αr_{D} and r_{A} exhibit similar orders of magnitude for each cell when a good balance is reached. Since under the same flow conditions the CFL number is positively correlated to the change of total variation, we approximate α as α_0/CFL to automatically control the weight of r_{D} for varying CFL numbers in this study.

Note that a constant α_0 is introduced when constructing the blending function α. At a first glance, it seems that α_0 can be a random number as the ratio between r_{A} and r_{D} depends on the problem to be solved. However, a preliminary analysis indicates that the order of magnitude of α_0 can be estimated when the flow problem is appropriately non-dimensionalized. The estimation procedure can go as follows. Based on the definition of r_{A}, its maximum magnitude is 2, around $\mathcal{O}(1)$. Therefore, the order of magnitude of $\alpha_0 r_{\mathrm{D}}/\mathrm{CFL}$ should be targeted around $\mathcal{O}(1)$ for all problems as the order of magnitude of r_{A} always holds. Since r_{A}, r_{D}, and CFL all have the dimension of one, the dimension of α_0 also needs to be one. At this point, numerical experiments can be carried out with non-dimensionalized problems to determine the order of magnitude of r_{D}. Before moving forward, we mention that the order of magnitude of r_{D} can actually be estimated from numerical simulations with the MUSCL scheme. Therein, we found that r_{D} is around $\mathcal{O}(10^{-2})$, and thus, α_0/CFL should be around $\mathcal{O}(10^{2})$ to make the order of magnitude of αr_{D} be around $\mathcal{O}(1)$, matching that of r_{A}. Since the CFL number varies between $\mathcal{O}(10^{-1})$ and $\mathcal{O}(1)$, the value of α_0 can be between $\mathcal{O}(10)$ and $\mathcal{O}(10^{2})$.

Now we present the observations from our numerical experiments. For all simulations, the CFL number is fixed at 0.2. We varied α_0 in a wide range from $\mathcal{O}(10^{-1})$ to

$\mathcal{O}(10^3)$, and found that the total variation change between two successive time steps for appropriately non-dimensionalized problems shows sloppy features. This sloppiness actually ensures the effectiveness of numerical experiments used to estimate the order of magnitude of r_D and the value of α_0. A key observation is that the reinforcement learning algorithm can always learn a certain policy to maximize the reward. However, the balance between r_D and r_A can be undesirable due to different choices of α_0. If α_0 is set as a large number (e.g., 10^3), the penalty for violating the TVD constraint can be very severe (i.e., αr_D is a large negative number). The algorithm then simply chose actions that satisfy the TVD constraints leading to highly dissipative numerical schemes. If α_0 is set as a small number (e.g., 0.1), we observed that the final policy excessively favored accuracy and led to spurious oscillations in simulation results because r_A dominated in the reward. From extensive tests, α_0 between 1 and 100 creates good balance between numerical dissipation and accuracy. Therefore, $\alpha_0 = 50$ is used in nonlinear numerical scheme training, and the machine-learned schemes can be generalized to different physics, grid resolutions, and spatial dimensions, as demonstrated in Section 3.

References

[1] Meister, Andreas, and Jens Struckmeier. Hyperbolic Partial Differential Equations: Theory, Numerics and Applications. Springer Science & Business Media, 2012.

[2] Godunov, Sergei K., and I. Bohachevsky. Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics. Mat. Sb. (N.S.), 47 (1959): 271-306.

[3] Van Leer, Bram. Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. J. Comput. Phys., 32 (1979): 101-136.

[4] Harten, Ami, Bjorn Engquist, Stanley Osher, and Sukumar R. Chakravarthy. Uniformly High Order Accurate Essentially Non-Oscillatory Schemes, III. Springer, 1997.

[5] Jiang, Guang-Shan, and Chi-Wang Shu. Efficient implementation of weighted ENO schemes. J. Comput. Phys., 126 (1996): 202-228.

[6] Cockburn, Bernardo, Chi-Wang Shu, Claes Johnson, Eitan Tadmor, and Chi-Wang Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. Springer, 1998.

[7] Cockburn, Bernardo, and Chi-Wang Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework. Math. Comput., 52 (1989): 411-435.

[8] Kim, Kyu Hong, and Chongam Kim. Accurate, efficient and monotonic numerical methods for multi-dimensional compressible flows: Part II: Multi-dimensional limiting process. J. Comput. Phys., 208 (2005): 570-615.

[9] You, Hojun, and Chongam Kim. High-order multi-dimensional limiting strategy with subcell resolution I. Two-dimensional mixed meshes. J. Comput. Phys., 375 (2018): 1005-1032.

[10] Corrigan, Andrew, Andrew D. Kercher, and David A. Kessler. A moving discontinuous Galerkin finite element method for flows with interfaces. Int. J. Numer. Methods Fluids, 89 (2019): 362-406.

[11] Luo, Hong, Gianni Absillis, and Robert Nourgaliev. A moving discontinuous Galerkin finite element method with interface condition enforcement for compressible flows. J. Comput.

Phys., 445 (2021).

[12] Persson, Per-Olof, and Jaime Peraire. Sub-cell shock capturing for discontinuous Galerkin methods. In: 44th AIAA Aerospace Sciences Meeting and Exhibit, 2006: 112.

[13] Yu, M.L., Francis X. Giraldo, Melinda Peng, and Zhi-Jian Wang. Localized artificial viscosity stabilization of discontinuous Galerkin methods for nonhydrostatic mesoscale atmospheric modeling. Mon. Weather Rev., 143 (2015): 4823-4845.

[14] Haga, Takanori, and Soshi Kawai. On a robust and accurate localized artificial diffusivity scheme for the high-order flux-reconstruction method. J. Comput. Phys., 376 (2019): 534-563.

[15] Ray, Deep, and Jan S. Hesthaven. An artificial neural network as a troubled-cell indicator. J. Comput. Phys., 367 (2018): 166-191.

[16] Ray, Deep, and Jan S. Hesthaven. Detecting troubled-cells on two-dimensional unstructured grids using a neural network. J. Comput. Phys., 397 (2019).

[17] Beck, Andrea D., Jonas Zeifang, Anna Schwarz, and David G. Flad. A neural network based shock detection and localization approach for discontinuous Galerkin methods. J. Comput. Phys., 423 (2020).

[18] Bezgin, Deniz A., Steffen J. Schmidt, and Nikolaus A. Adams. WENO3-NN: A maximum-order three-point data-driven weighted essentially non-oscillatory scheme. J. Comput. Phys., 452 (2022).

[19] Nguyen-Fotiadis, Nga, Michael McKerns, and Andrew Sornborger. Machine learning changes the rules for flux limiters. Phys. Fluids, 34 (2022).

[20] Bar-Sinai, Yohai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. Proc. Natl. Acad. Sci. U.S.A., 116 (2019): 15344-15349.

[21] Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. Nature, 550 (2017): 354-359.

[22] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[23] Zhao, Wenshuai, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: A survey. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2020: 737-744.

[24] Hu, Chang, Krishnakant V. Saboo, Ahmad H. Ali, Brian D. Juran, Konstantinos N. Lazaridis, and Ravishankar K. Iyer. REMEDI: Reinforcement learning-driven adaptive metabolism modeling of primary sclerosing cholangitis disease progression. arXiv preprint arXiv:2310.01426, 2023.

[25] Yu, Chao, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. ACM Comput. Surv., 55 (2021): 1-36.

[26] Chen, Wenjie, Qiulei Wang, Lei Yan, Gang Hu, and Bernd R. Noack. Deep reinforcement learning-based active flow control of vortex-induced vibration of a square cylinder. Phys. Fluids, 35 (2023).

[27] Novati, Guido, Hugues Lascombes de Laroussilhe, and Petros Koumoutsakos. Automating turbulence modelling by multi-agent reinforcement learning. Nat. Mach. Intell., 3 (2021): 87-96.

[28] Bae, H. Jane, and Petros Koumoutsakos. Scientific multi-agent reinforcement learning for wall-models of turbulent flows. Nat. Commun., 13 (2022).

[29] Sutton, Richard S., and Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press, 2018.

[30] Wang, Yufei, Ziju Shen, Zichao Long, and Bin Dong. Learning to discretize: Solving 1D scalar conservation laws via deep reinforcement learning. arXiv preprint arXiv:1905.11079, 2019.

[31] Way, Elliot, Dheeraj S.K. Kapilavai, Yiwei Fu, and Lei Yu. Backpropagation through time and space: Learning numerical methods with multi-agent reinforcement learning. arXiv preprint arXiv:2203.08937, 2022.

[32] Fu, Yiwei, Dheeraj S.K. Kapilavai, and Elliot Way. Multi-agent learning of numerical methods for hyperbolic PDEs with factored Dec-MDP. In: International Conference on Practical Applications of Agents and Multi-Agent Systems, Springer, 2022: 179-190.

[33] Fu, Lin, Xiangyu Y. Hu, and Nikolaus A. Adams. A family of high-order targeted ENO schemes for compressible-fluid simulations. J. Comput. Phys., 305 (2016): 333-359.

[34] Feng, Yiqi, Felix S. Schranner, Josef Winter, and Nikolaus A. Adams. A deep reinforcement learning framework for dynamic optimization of numerical schemes for compressible flow simulations. J. Comput. Phys., 493 (2023).

[35] Clain, Stéphane, Steven Diot, and Raphaël Loubère. A high-order finite volume method for systems of conservation laws–Multi-dimensional Optimal Order Detection (MOOD). J. Comput. Phys., 230 (2011): 4028-4050.

[36] Schwarz, Anna, Jens Keim, Simone Chiocchetti, and Andrea Beck. A reinforcement learning based slope limiter for second-order finite volume schemes. PAMM 23, no. 1 (2023).

[37] Keim, Jens, Anna Schwarz, Simone Chiocchetti, Christian Rohde, and Andrea Beck. A reinforcement learning based slope limiter for two-dimensional finite volume schemes. In: International Conference on Finite Volumes for Complex Applications, Springer, 2023: 209-217.

[38] Konda, Vijay, and John Tsitsiklis. Actor-critic algorithms. Adv. Neural Inf. Process. Syst., 12 (1999).

[39] Fujimoto, Scott, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning, PMLR, 2018: 1587-1596.

[40] Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. Adv. Neural Inf. Process. Syst., 32 (2019).

[41] Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. arXiv preprint arXiv:1606.01540, 2016.

[42] Agarap, Abien Fred. Deep learning using rectified linear units (ReLU). arXiv preprint arXiv:1803.08375, 2018.

[43] Kingma, Diederik P., and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[44] Harten, Ami. High resolution schemes for hyperbolic conservation laws. J. Comput. Phys., 135 (1997): 260-278.

[45] Liska, Richard, and Burton Wendroff. Comparison of several difference schemes on 1D and 2D test problems for the Euler equations. SIAM J. Sci. Comput., 25 (2003): 995-1017.

[46] Zhang, Xiangxiong, and Chi-Wang Shu. On positivity-preserving high order discontinuous Galerkin schemes for compressible Euler equations on rectangular meshes. J. Comput. Phys., 229 (2010): 8918-8934.