technische universität dortmund

fakultät für mathematik

A thesis submitted for the degree of
Master of Science

Discrete Model Reduction for the Shallow Water Equations Using Neural Networks and Flux Limiting

Alexey Schwarzmann

July 27, 2023

Lehrstuhl für Angewandte Mathematik und Numerik
Fakultät für Mathematik
Technische Universität Dortmund

Acknowledgements

I want to thank Prof. Dr. Dmitri Kuzmin and Prof. Dr. Ilya Timofeyev for supervising this thesis. I thank Dmitri for his invaluable advice, financial support, and proofreading parts of this thesis. His guidance and assistance were always helpful, and he introduced me to Ilya when I expressed interest in conducting research abroad.

I am genuinely grateful to Ilya for agreeing to supervise me at the University of Houston (UH) and allowing me to visit UH and conduct research there. I appreciate the fruitful discussions with Ilya and his advice, which played a significant role in creating this work. Moreover, I would like to thank Ilya for making my time in Houston a wonderful experience and for his help and tips, which made it easy for me to arrive in the USA.

I sincerely thank the Studienstiftung des Deutschen Volkes for their financial support, without which it would not have been possible for me to conduct research in Houston. I would also like to extend my thanks to Dr. Maximilian Stegemeyer for proofreading my thesis and providing valuable suggestions. Additionally, I am also grateful to the Applied Mathematics (LS III) members at TU Dortmund for their support and contributions.

Lastly, my heartfelt thanks go to my family for their persistent support, motivation, and presence when I needed them the most. I am particularly grateful to my wife, Manon, whose support has enabled me to complete this work. Thank you so much for everything!

Contents

1 Introduction

How can we use machine learning and neural networks (NNs) in synergy with ocean, climate, and atmospheric science models? Simulating these models requires solving partial differential equations spanning a wide range of time and length scales in computational fluid dynamics. However, limited computational power may prevent climate models from fully resolving these processes before the effects of climate change occur [Sch17]. To overcome these limitations, *parameterizations*, known as closure models, represent subgrid-scale processes. Nevertheless, parameterizations introduce significant uncertainties and biases [Wil07, Bec08, Far11, Ste13, Gri15].

One promising approach is to use machine learning algorithms (ML) to develop new parameterization schemes by fitting them to the results of relatively expensive physical models that more accurately represent the subgrid dynamics. ML models can learn complex mappings by minimizing errors between their predictions and known outputs over many training examples.

In climate modeling, NNs are one of the most popular areas of machine learning. In particular, they have been applied to solve forced Burger's equation [Alc21a, Sub21], and used for ocean modelling [Bol19, Zan20, Gui21], to represent clouds [Ras18], residual heating and moistening [Bre19], convection [Kra13, Yuv21]. NNs have some computational advantages over other ML algorithms, such as the possibility of greater accuracy due to their universal approximation properties [Cyb89, Hor89] and needing substantially less memory when implemented. Furthermore, very efficient parameterizations can be obtained by implementing NNs at reduced precision in graphics processing units (GPUs), tensor processing units (TPUs), and even in central processing units (CPUs) [Van11].

However, neural networks can suffer from certain limitations, including instabilities, lack of interpretability, and poor generalization to unseen conditions during training [O'G18, Ras18, Bol19, Bre19]. One possible reason for these issues is that NN-based models do not inherently conserve energy and mass, which can impact their performance in specific applications. To address this concern, Beucler et al. [Beu19] suggested that by either constraining the loss function or modifying the architecture of the network, NNs can be enforced to satisfy physical properties. Another approach for preserving conservation laws was introduced by Alcala and Timofeyev in [Alc21a]. Using a finite difference method for spatial discretization, they could

parametrize subgrid effects by local *subgrid fluxes*, which they added to the resolved fluxes. This approach ensures mass conservation and improves the model's ability to represent subgrid-scale processes.

This thesis focuses on the application of the approach proposed in [Alc21a] to the shallow water equations (SWE), which are extensively used in environmental modeling and the prediction of natural disasters, including tsunamis, storm surges, dam breaks, and more. Our primary objectives are to ensure the positivity of the height state and maintain numerical admissibility while solving the SWE. To accomplish this, we implement the monolithic convex limiting strategy introduced by Kuzmin [Kuz20].

We begin, in Chapter 2, by introducing the SWE. Unlike [Alc21a], we use the finite volume method for spatial discretization and feedforward neural networks. In Chapter 3, after briefly introducing the finite volume method, we present the local Lax-Friedrichs (LLF) scheme, which will serve as our base scheme. We also discuss the time discretization used and the properties of the LLF in this context. In Chapter 4, we present the subgrid parameterization proposed by [Alc21a] and feedforward neural networks. We will refer to the resulting scheme as a *reduced model*. The monolithic convex limiting strategy for SWE is discussed in Chapter 5. In Chapters 6 and 7, we perform numerical studies and draw preliminary conclusions based on the findings of this research.

2 Shallow water equations

Let us consider a fluid flowing through a one-dimensional pipe of the length L with some velocity $v(x,t)$, which depends on the position in the pipe $x \in [0,L]$ and time $t > 0$. We assume that the fluid is incompressible, i.e., its density is constant. Instead, we allow the fluid to vary its depth or *height* $h(x,t)$. The flow of this fluid can be described with the *shallow water equations* (SWE), which were first introduced by Saint-Venant in [SV71]. That is why they are sometimes called *Saint-Venant equations*. As the name of these equations suggests, the pipe length L must be significantly large relative to the depth h. Furthermore, let g be the gravitational acceleration. Using subscripts to denote partial derivatives, we are ready to present the shallow water equations

$$\begin{bmatrix} h \\ hv \end{bmatrix}_t + \begin{bmatrix} hv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_x = 0.$$

See [LeV02, Ch. 13] for a derivation of these equations. We also refer to [Vre94, Ch. 2] for a more general system of equations with non-constant bathymetry and its derivation.

To solve this problem, we also require boundary conditions and initial data. For simplicity, we assume periodic boundary conditions at the boundaries of the pipe. We can interpret this as the pipe having a shape of a torus. The initial data at $x \in [0,L]$ is given by

$$h(x,0) = h_0(x), \quad v(x,0) = v_0(x).$$

For further discussion, it is also useful to note that the problem can also be defined as a *hyperbolic conservation law*

$$u_t + (f(u))_x = 0, \tag{2.1}$$

where u denotes the unknown and f is referred to as the *flux function*. For that, we assume the height h is strictly positive, i.e., no dry states are present. Then, we obtain by introducing $q = hv$:

$$\begin{bmatrix} h \\ q \end{bmatrix}_t + \begin{bmatrix} q \\ q^2/h + \frac{1}{2}gh^2 \end{bmatrix}_x = 0. \tag{2.2}$$

The quantity q is often called the *discharge* in the shallow water theory since it measures the flow rate of water. The equation (2.2) has the form of (2.1) with

$$u = \begin{bmatrix} h \\ q \end{bmatrix}, \qquad f(u) = \begin{bmatrix} q \\ q^2/h + \frac{1}{2}gh^2 \end{bmatrix}$$

and the initial data is given by

$$u(\cdot, 0) = \begin{bmatrix} h_0 \\ q_0 \end{bmatrix} \qquad \text{in } [0, L],$$

where $q_0 = h_0 v_0$.

3 Local Lax-Friedrichs method

We now turn to the design of a reliable method for solving the SWE we introduced in the previous chapter. Section 3.1 begins by considering general conservation laws and presenting the finite volume discretization. Then we formulate a concrete numerical scheme for solving our problem. Section 3.2 uses the local Lax-Friedrichs method to formulate a spatial semi-discrete scheme. Afterward, we transform it in Section 3.3 into a fully discrete scheme using explicit Runge-Kutta methods. This scheme will form the base for the reduced model in Chapter 4 and the monolithic convex limiting strategy in Chapter 5. Finally, in Section 3.4, we show that our fully discrete scheme preserves the positivity of the height.

3.1 Finite volume discretization

Let us start with a conservation law of the form

$$u_t(x,t) + (f(u(x,t)))_x = 0, \qquad (x,t) \in [0,L] \times (0,T), \tag{3.1}$$

where $T > 0$ is the target time we aim to reach. We assume periodic boundary conditions in 0 and L for simplicity. The initial condition is given by

$$u(x,0) = u_0, \qquad x \in [0,L].$$

Integrating (3.1) between two points $x_1, x_2 \in [0,L]$ leads to

$$\int_{x_1}^{x_2} u_t \mathrm{d}x = -\int_{x_1}^{x_2} f(u)_x \mathrm{d}x = f(u(x_1,t)) - f(u(x_2,t)).$$

If it is allowed to take the derivative outside the integral, we obtain

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{x_1}^{x_2} u(x,t)\mathrm{d}x = f(u(x_1,t)) - f(u(x_2,t)). \tag{3.2}$$

The rate of change of u between x_1 and x_2 is determined by the difference in $f(u)$ evaluated at these points. Thus, the quantity u is conserved and can only flow out at the edges. This behavior is why we refer to $f(u)$ as the flux function.

Next, we discretize the spatial domain $[0, L]$ into uniform computational cells

$$C_i = \left[x_{i-1/2}, x_{i+1/2}\right],$$

with the cell width $\Delta x = x_{i+1/2} - x_{i-1/2}$ and $i = 1, \ldots, N$. Furthermore, the *cell average*

$$u_i(t) = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} u(x,t) \mathrm{d}x \tag{3.3}$$

approximates the average value over the cell C_i at time t. Using (3.2) and approximating the flux function $f(u(x_{i+1/2}))$ by a suitably chosen *numerical flux* $F_{i+1/2}$, we obtain a system of ordinary differential equations

$$\frac{\mathrm{d}}{\mathrm{d}t} u_i = \frac{F_{i-1/2} - F_{i+1/2}}{\Delta x}, \qquad i = 1, \ldots, N. \tag{3.4}$$

We refer to this system of equations as *spatial semi-discretization*.

Since hyperbolic information propagates with finite speed, it is reasonable to assume that $F_{i+1/2}$ depends on the cell averages u_i and u_{i+1}. In other words, there exists a function denoted as $\mathcal{F}$, such that

$$F_{i+1/2} = \mathcal{F}(u_i, u_{i+1}),$$

where $\mathcal{F}$ is referred to as a *numerical flux function*.

Many standard finite-volume methods employ a two-point stencil for the numerical flux $F_{i+1/2}$, such as the local Lax-Friedrichs one in the following section. Using larger stencils may also be reasonable, which we will discuss in Section 4.1.

3.2 Local Lax-Friedrichs Flux

There are many ways to choose the numerical flux $F_{i+1/2}$ (see for instance [LeV02]). A popular choice for the numerical flux we will use in this thesis is the one associated with the *local Lax-Friedrichs* (LLF) method. This method is dissipative but stable and preserves the height positivity, as seen in Section 3.4. Therefore, it is ideally suited as a base space discretization.

To obtain the LLF (Local Lax-Friedrichs) flux, we treat each inter-cell boundary as a Riemann problem. Assuming that the solution has the form depicted in Fig. 3.1, which consists of two traveling discontinuities, we can use the Rankine-Hugoniot condition. Applying this condition, we obtain the LLF given by

$$F_{i+1/2} = \frac{f(u_i) + f(u_{i+1})}{2} - \frac{\lambda_{i+1/2}}{2}(u_{i+1} - u_i),$$

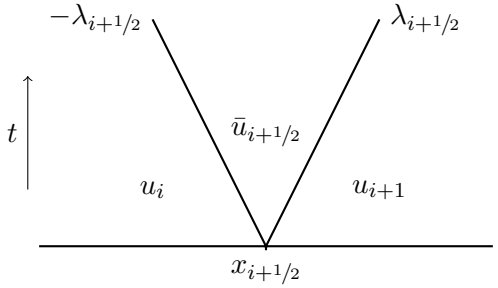


Figure 3.1: Structure of the local Lax-Friedrichs approximate Riemann solution at the edge $x_{i+1/2}$. The solution consists of two waves propagating at the speeds $-\lambda_{i+1/2}$ and $\lambda_{i+1/2}$. The left and the right states are the cell averages u_i and u_{i+1}. In between, we have a single new state $\bar{u}_{i+1/2}$.

where $\lambda_{i+1/2}$ represents the wave speed of the traveling discontinuities. In addition, the intermediate state is

$$\bar{u}_{i+1/2} := \frac{u_{i+1} + u_i}{2} - \frac{1}{2\lambda_{i+1/2}}(f(u_{i+1}) - f(u_i)). \tag{3.5}$$

In the following, we refer to the intermediate state $\bar{u}_{i+1/2}$ as a *bar state*.

Let us discuss the choice of the wave speed $\lambda_{i+1/2}$. It estimates the largest possible wave speed of the associated Riemann problem. For systems like the SWE, we use the eigenvalues of the Jacobian of the flux function

$$f'(h, v) = \begin{bmatrix} v & h \\ v^2 + gh & 2hv \end{bmatrix},$$

which are given by

$$\lambda_- = v - \sqrt{gh}, \qquad \lambda_+ = v + \sqrt{gh}.$$

We employ the maximum eigenvalue of the Jacobian matrices $f'(u_i)$ and $f'(u_{i+1})$ as an upper bound estimate [LeV02] of the wave speed, i.e.,

$$\lambda_{i+1/2} := \max\{\, |v_i| + \sqrt{gh_i},\, |v_{i+1}| + \sqrt{gh_{i+1}} \,\}. \tag{3.6}$$

Finally, using the LLF flux 3.2 and the definition of the bar states (3.5), we write spatial semi-discretization (3.4) of the LLF scheme as

$$\frac{\mathrm{d}}{\mathrm{d}t} u_i = \frac{1}{\Delta x}\left[\lambda_{i-1/2}(\bar{u}_{i-1/2} - u_i) + \lambda_{i+1/2}(\bar{u}_{i+1/2} - u_i)\right], \tag{3.7}$$

for $i = 1, \ldots, N$. This equation is also known as the *fluctuation form* of the finite volume scheme [LeV02].

3.3 Time discretization

Next, we transform the spatial semi-discretization (3.7) into a full discretization. For this purpose, we need a numerical scheme to solve the initial value problem

$$\begin{aligned} u'(t) &= z(u(t)) \qquad t \in (0,T), \\ u(0) &= u_0, \end{aligned}$$

where $u(t) = (u_1(t), \dots, u_N(t))$ is the vector of cell averages, z the right-hand side and u_0 the initial condition. We wish to replace the function $u(t)$ by a sequence of approximations $u^0 = u_0$, $u^1 \approx u(t_1)$, $\dots$, $u^n \approx u(t_n)$ at discrete time instances $0 = t_0 < t_1 < \cdots < t_n = T$.

A popular choice is to use the explicit *forward Euler* method

$$u^{k+1} = u^k + \Delta t z(u^k), \tag{3.9}$$

where $\Delta t = \Delta t(k)$ denotes the time step between t_{k+1} and t_k. Using the forward Euler method, the spatial semi-discretization (3.7) transforms into the full discretization

$$u_i^{k+1} = u_i^k + \frac{\Delta t}{\Delta x}\big[\lambda_{i-1/2}(\bar{u}_{i-1/2}^k - u_i^k) + \lambda_{i+1/2}(\bar{u}_{i+1/2}^k - u_i^k)\big], \tag{3.10}$$

where $i = 1, \dots, N$ and $k = 0, \dots, n$.

In our numerical examples (Chapter 6), we employ a different method known as *Heun's method*, which can be expressed as follows:

$$u^{(1)} = u^k + \Delta t z(u^k), \tag{3.11a}$$

$$u^{k+1} = \frac{1}{2}u^k + \frac{1}{2}\big(u^{(1)} + \Delta t z(u^{(1)})\big), \tag{3.11b}$$

Heun's method is also known as the *modified Euler's method* or the *explicit trapezoid rule*. This scheme can be understood as a two-step Runge-Kutta method, which consists of two forward Euler steps. It is often preferred over the standard forward Euler method as it provides better accuracy and stability for specific problems.

Other procedures are also conceivable. An exciting class of methods for our purposes are the *strong stability preserving* (SSP) methods [Shu88, Got01, Got11], to which both the forward Euler and Heun's method belong. They can be represented as convex combinations of Euler steps, among other things. We will see how the LLF method and monolithic convex limiting strategy exploit this fact in the following section and Chapter 5, respectively. Nevertheless, choosing a suitable time step is crucial, which we will also discuss in the following section.

3.4 Height positivity preservation

One reason we select the LLF method is not only the method's stability but also that it preserves height positivity. However, time discretization must fulfill certain conditions that we investigate in this section. This will be an essential basis for the monolithic convex limiting in Chapter 5.

Definition 3.1 (Height positivity preserving)**.** Consider a state $(h_1, q_1, \dots, h_N, q_N)$ where the height cell averages are strictly positive, i.e., $h_i > 0$ for all $i = 1, \dots, N$. We call a fully discrete scheme *height positivity preserving* (HPP) if the height cell averages of the time updated state $(\hat{h}_1, \hat{q}_1, \dots, \hat{h}_N, \hat{q}_N)$ are also strictly positive, i.e., $\hat{h}_i > 0$ for all $i = 1, \dots, N$.

First, we demonstrate the HPP property of the LLF scheme (3.7) discretized in time with the forward Euler method (3.9). We have already captured the resulting algorithm in (3.10). The following theorem is adapted from [Gue16].

Theorem 3.2. *Assume that the condition*

$$\Delta t \leq \frac{1}{\lambda_{i-1/2} + \lambda_{i+1/2}} \Delta x \tag{3.12}$$

is satisfied. Then the full discretization (3.10) *is HPP.*

The central concept revolves around reformulating the scheme (3.10) into the following form [Gue16, Kuz20]:

$$\begin{aligned} \hat{u}_i &= u_i + \frac{\Delta t}{\Delta x}\big[\lambda_{i-1/2}(\bar{u}_{i-1/2} - u_i) + \lambda_{i+1/2}(\bar{u}_{i+1/2} - u_i)\big], \\ &= \Big(1 - \frac{\Delta t}{\Delta x}(\lambda_{i-1/2} + \lambda_{i+1/2})\Big)u_i + \frac{\Delta t}{\Delta x}(\lambda_{i-1/2}\bar{u}_{i-1/2} + \lambda_{i+1/2}\bar{u}_{i+1/2}). \end{aligned}$$

Here, $\hat{u}_i$ represents the updated solution, and Δt denotes the current time step. By observing that the Courant-Friedrichs-Lewy (CFL)-like condition (3.12) is met, then the update $\hat{u}_i$ corresponds to a convex combination of the initial state u_i and the bar states $\bar{u}_{i-1/2}$ and $\bar{u}_{i+1/2}$, where the bar states are defined in (3.5). Therefore, verifying the positivity of the height components of the bar states $\bar{u}_{i-1/2}$ and $\bar{u}_{i+1/2}$ is sufficient to prove the HPP property of the scheme.

Lemma 3.3. *Let $h_i > 0$ for all $i = 1, \dots, N$, then $\bar{h}_{i+1/2} > 0$ for all $i = 1, \dots, N$.*

Proof. Let $i = 1, \dots, N$. According to (3.5), the height bar state is defined via

$$\bar{h}_{i+1/2} := \frac{h_{i+1} + h_i}{2} - \frac{1}{2\lambda_{i+1/2}}(v_{i+1}h_{i+1} - v_i h_i). \tag{3.14}$$

We reformulate (3.14) into

$$\bar{h}_{i+1/2} = \frac{1}{2}\Big(1 - \frac{v_{i+1}}{\lambda_{i+1/2}}\Big)h_{i+1} + \frac{1}{2}\Big(1 + \frac{v_i}{\lambda_{i+1/2}}\Big)h_i. \tag{3.15}$$

Using the definition of the wave speed (3.6)

$$\lambda_{i+1/2} := \max\{\, |v_i| + \sqrt{gh_i},\ |v_{i+1}| + \sqrt{gh_{i+1}}\,\}$$

and $h_i, h_{i+1} > 0$, we obtain

$$\frac{|v_i|}{\lambda_{i+1/2}} \le \frac{|v_i|}{|v_i| + \sqrt{gh_i}} < 1, \qquad \frac{|v_{i+1}|}{\lambda_{i+1/2}} \le \frac{|v_{i+1}|}{|v_{i+1}| + \sqrt{gh_{i+1}}} < 1.$$

Combining this with (3.15) completes the proof. □

The HPP property of the LLF method, discretized in time with Heun's method, can be deduced from Theorem 3.2. To accomplish this, we recognize that each stage is a convex combination of a forward Euler update and the initial stage. As a result, the following corollary can readily be extended to SSP methods.

Corollary 3.4. *Assume that the condition*

$$\Delta t \le \frac{1}{\lambda_{i-1/2} + \lambda_{i+1/2}}\Delta x$$

is satisfied. Then the LLF method (3.7) *discretized in time with Heun's method* (3.11) *is HPP.*

Proof. Let $h_j > 0$ for every $j = 1, \dots, N$ and fix $i = 1, \dots, N$. Using the HPP property of the LLF method, which is discretized in time with the forward Euler method (3.10), we can deduce $h_i^{(1)} > 0$, where $h^{(1)}$ denotes the first stage of Heun's method (3.11a). Another use of the HPP property of (3.10) yields

$$h_i^{(1)} + \Delta t z(h^{(1)})_i > 0.$$

Since we assumed $h_i > 0$, we conclude that the time update $\hat{h}_i$ defined by (3.11b) is also positive. □

4 Model reduction using Neural Networks

Achieving accurate resolution of SWE using the LLF method requires a fine mesh, which can be computationally expensive. To address this issue, we present a subgrid parameterization proposed by Alcala and Timofeyev [Alc21a] in Section 4.1. This approach modifies an LLF method on a coarse mesh to reproduce the behavior of the LLF method on the fine mesh by introducing a subgrid flux. However, computing the subgrid flux still requires the solution on the fine mesh.

To overcome the need for a fine mesh solution in computing the subgrid flux, we employ a feedforward neural network (FNN). This FNN approximates the subgrid flux using the available coarse mesh data. Our approach diverges from that of [Alc21a], which employs on *generative adversarial networks* (GAN) [Goo14] or Wasserstein GAN (WGAN) [Arj17, Gul17]. We present the construction and training of FNNs in Section 4.2. Our presentation is largely based on [Nie15, Goo16, Qua22].

4.1 Subgrid flux parameterization

We begin by considering a fine uniform mesh of some interval $I = [a, b]$, consisting of cells

$$C_i = [x_{i-1/2}, x_{i+1/2}], \qquad i = 1, \dots, N,$$

where each cell C_i has a cell length of Δx. Additionally, $\mathcal{F}$ represents a numerical flux function used in a finite volume scheme, and u_i corresponds to the cell averages of cell C_i. Referring to (3.4), the *subgrid semi-discrete scheme* can be defined by

$$\frac{\mathrm{d}}{\mathrm{d}t} u_i = \frac{F_{i-1/2} - F_{i+1/2}}{\Delta x}, \qquad i = 1, \dots, N. \tag{4.1}$$

Here, $F_{i+1/2} = \mathcal{F}(u_i, u_{i+1})$ represents the numerical flux between the cell averages u_i and u_{i+1}. In the following, we will refer to the cell average u_i as a *subgrid mode*.

This thesis primarily focuses on the LLF method as the chosen finite volume scheme. However, it is worth noting that alternative finite volume schemes can be employed. Additionally, the approach can also be extended to incorporate finite difference schemes. In such cases, the variable u_i no longer represents the cell average (3.3) of

cell C_i. Instead, it denotes the numerical approximation of the unknown variable $u(x,t)$ at the cell center.

To obtain a scheme on larger length scales, we reduce the number of cells by merging them. Let $k > 0$ represent the *coarsening degree*, and assume that N is a multiple of k. Then, the *coarse mesh* consists of

$$\bar{C}_i := \bigcup_{j=k(i-1)+1}^{ki} C_j = [x_{k(i-1)+1/2},\, x_{ki+1/2}], \qquad i = 1, \dots, N/k,$$

and the cell length of the coarse mesh is given by $\Delta X := k\Delta x$. Fig. 4.1 illustrates an example of such a coarsening process with $k = 3$. In the subsequent discussion, the index i always refers to the range $\{1, \dots, N/k\}$ without explicit mention.

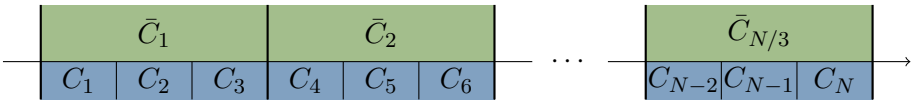


Figure 4.1: A fine mesh, which includes cells $C_1, \dots, C_N$, undergoes a coarsening process with a coarsening degree $k = 3$. This results in a coarse mesh consisting of $\bar{C}_1, \dots, \bar{C}_N$.

By employing the chosen finite volume method, we obtain the following semi-discretization for the coarse mesh:

$$\frac{\mathrm{d}}{\mathrm{d}t} U_i = \frac{\bar{F}_{i-1/2} - \bar{F}_{i+1/2}}{\Delta X}. \tag{4.2}$$

Here, U_i represents the cell average of cell $\bar{C}_i$ and $\bar{F}_{i+1/2} = \mathcal{F}(U_i, U_{i+1})$. Henceforth, we refer to the cell average U_i of cell $\bar{C}_i$ as a *resolved mode*, and the flux $\bar{F}_{i+1/2}$ is called a *resolved flux*. However, it is important to note that this resulting scheme may exhibit reduced accuracy compared to the subgrid scheme described in (4.1).

To resolve the underlying subgrid processes that arise from the discretization on the underlying fine mesh (4.1), we aim to represent the resolved modes using the subgrid modes. To accomplish this, we consider the definition of cell averages 3.3, which give us:

$$U_i(t) = \frac{1}{k} \sum_{j=k(i-1)+1}^{ki} u_j(t). \tag{4.3}$$

Essentially, we obtain the resolved modes by averaging the subgrid ones. This idea is similar to box filtering used in large eddy simulations [Pop00]. By differentiat-

ing (4.3), we obtain

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}U_i &= \frac{1}{k}\sum_{j=k(i-1)+1}^{ki}\frac{\mathrm{d}}{\mathrm{d}t}u_j \\ &= \frac{1}{k}\sum_{j=k(i-1)+1}^{ki}\frac{F_{j-1/2}-F_{j+1/2}}{\Delta x} \\ &= \frac{F_{k(i-1)+1/2}-F_{ki+1/2}}{\Delta X},\end{aligned} \tag{4.4}$$

where we employed the subgrid semi-discretization (4.1) and the property of telescoping cancellation. To establish the connection between the resolved semi-discrete-scheme (4.2) and (4.4), we rewrite (4.4) as follows:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}U_i &= \frac{\bar{F}_{i-1/2}-\bar{F}_{i+1/2}}{\Delta X}+\left(\frac{F_{k(i-1)+1/2}-F_{ki+1/2}}{\Delta X}-\frac{\bar{F}_{i-1/2}-\bar{F}_{i+1/2}}{\Delta X}\right) \\ &= \frac{\bar{F}_{i-1/2}-\bar{F}_{i+1/2}}{\Delta X}+\frac{1}{\Delta X}[(F_{k(i-1)+1/2}-\bar{F}_{i-1/2})-(F_{ki+1/2}-\bar{F}_{i+1/2})].\end{aligned}$$

By introducing the *subgrid flux*

$$G_{i+1/2} := F_{ki+1/2}-\bar{F}_{i+1/2}, \tag{4.5}$$

we obtain

$$\frac{\mathrm{d}}{\mathrm{d}t}U_i = \frac{(\bar{F}_{i-1/2}+G_{i-1/2})-(\bar{F}_{i+1/2}+G_{i+1/2})}{\Delta X}.$$

This formulation allows us to fully resolve the subgrid processes resulting from the discretization on the underlying fine mesh. By including the subgrid flux $G_{i+1/2}$ in each resolved flux term, we account for the effects of these subgrid processes and achieve a more accurate representation.

We proceed by defining the *subgrid flux function*

$$\mathcal{G}(u_{ki}, u_{ki+1}, U_i, U_{i+1}) := G_{i+1/2} = \mathcal{F}(u_{ki}, u_{ki+1}) - \mathcal{F}(U_i, U_{i+1}). \tag{4.6}$$

The main challenge is that calculating the subgrid flux $G_{i+1/2}$ typically involves solving the problem on a fine mesh, which we aim to avoid due to its computational cost. To address this limitation, Alcala and Timofeyev proposed a solution in their work [Alc21a]. They suggested approximating the subgrid flux $\mathcal{G}$ with a function $\widetilde{\mathcal{G}}$ that solely depends on the resolved modes. To achieve this, they employed a neural network to model the function $\widetilde{\mathcal{G}}$. This innovative approach enables us to efficiently compute the subgrid flux by only relying on resolved modes, thereby eliminating the need for fine mesh calculations. Consequently, we will refer to an LLF method with a subgrid flux calculated using a neural network as a *reduced model*.

4.2 Feedforward neural networks

Machine learning is the discipline that allows computers to learn without being explicitly programmed to do so. Nevertheless, what do we mean by learning? Mitchell [Mit97] provides a concise definition:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

In the context of machine learning, our *task* is to approximate the subgrid flux function $\mathcal{G}$ defined in equation (4.6) based on a stencil of resolved modes. In Section 4.2.1, we explore the use of feedforward neural networks as a possible class of *models* capable of transforming resolved variables into subgrid fluxes. A model can be associated with a parameter-valued function f_θ that maps the input (resolved modes) to the output (subgrid flux approximation), where θ represents the eventual multidimensional parameter.

Moving on to Section 4.2.2, we begin with a *training set* that comprises *input data* (resolved modes) and *output data* (subgrid fluxes). This data set serves as the *experience* we provide for our machine learning algorithm. Next, we discuss how we train the chosen model f_θ to find a suitable parameter θ that minimizes the error between the model's predictions and the desired subgrid fluxes in the training set. This error is quantified using the *mean square loss* function, and the minimization is performed using stochastic gradient descent. The mean square loss function acts as our *performance measure*, allowing us to evaluate how well the model approximates the subgrid fluxes.

4.2.1 Model

To approximate the subgrid flux function $\mathcal{G}$ mentioned above, we employ a *feedforward neural network* (FNN). The term *network* describes this model because networks are commonly represented as a sequential composition of vector-valued functions. Specifically, we can consider a set of functions $\widetilde{G}_1, \dots, \widetilde{G}_n$ with $n \in \mathbb{N}$, where the overall function $\widetilde{G}$ can be expressed as the composition $\widetilde{G} = \widetilde{G}_n \circ \cdots \circ \widetilde{G}_1$. Each function $\widetilde{G}_i$ is referred to as the *i-th layer* of the network. In this context, the function $\widetilde{G}_1$ is known as the *input layer*, $\widetilde{G}_n$ as the *output layer*, and each function in between is referred to as a *hidden layer*. Fig. 4.2 depicts a visual representation of an FNN, showcasing the interconnected layers.

The term *feedforward* describes this type of network because information flows through the FNN in a unidirectional manner without any feedback loops that would

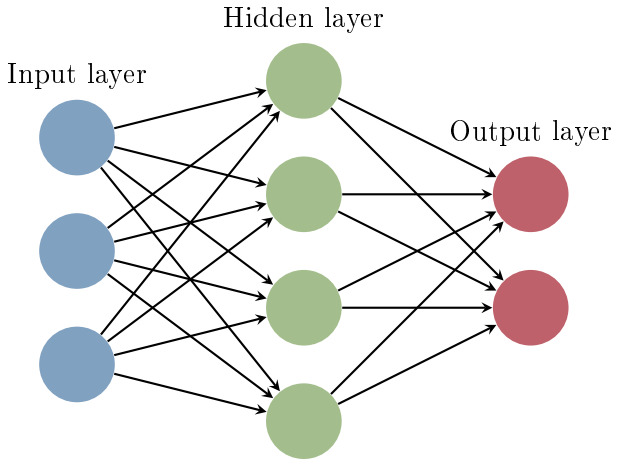


Figure 4.2: Visual representation of a FNN. The flow of information starts from the input layer, passes through the hidden layers, and ultimately generates an output in the output layer. Significantly, information in a feedforward network moves strictly in one direction and does not flow backward.

feed the output back into the network. However, it is worth noting that if such feedback loops exist, the networks are referred to as *recurrent networks*. We refer to Goodfellow et al. [Goo16, Chap. 10] for further details and insights.

Finally, FNNs are named *neural* due to their loose connection to neuroscience. In these networks, a layer is typically vector-valued, and each element of this vector can be considered as a *neuron* or *unit*. They are referred to as neurons because their design is inspired by neuroscientific observations about the functions performed by biological neurons.

Three main components characterize each neuron:

1. A *bias* parameter $b \in \mathbb{R}$,
2. a *weight* parameter $\omega \in \mathbb{R}^n$, where $n \in \mathbb{N}$ represents the dimension of the input, and
3. an *activation function* f.

When given an input $x \in \mathbb{R}^n$, the output of a neuron is computed as $f(\omega \cdot x + b)$, where $\omega \cdot x$ represents the dot product between the weight vector ω and the input vector x.

Fig. 4.3 illustrates the structure of such a neuron, highlighting its three components. However, it is essential to note that mathematical and engineering disciplines pri-

marily guide modern neural network research. The objective of neural networks is not to model the human brain but to develop robust mathematical models that excel in various practical applications.

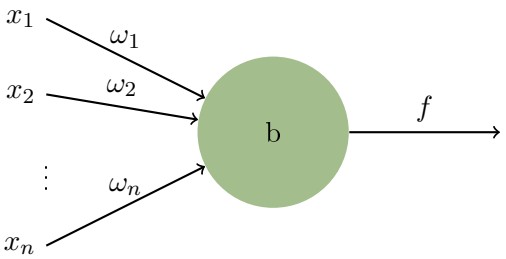


Figure 4.3: A neuron of an FNN. For a given input $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the neuron's output is computed as $f(\omega \cdot x + b)$, where $\omega = (\omega_1, \ldots, \omega_n) \in \mathbb{R}^n$ represents the *weights*, $b \in \mathbb{R}$ denotes the *bias*, and f represents the chosen *activation function*.

Let us discuss the selection of activation functions. One straightforward choice is the identity $f(z) = z$, which leads to neurons referred to as *linear units*. However, an FNN consisting solely of linear units becomes a linear affine mapping, effectively reducing the training process to a linear regression. Therefore, introducing non-linearity becomes crucial to enable the FNN to learn complex connections.

For hidden layers, the *ramp function* $f(z) = \max\{0, z\}$ is the widely used activation function. This function is similar to linear units, making them computationally efficient to optimize while introducing the necessary non-linearity to handle complex functions. However, a limitation of rectified linear units (ReLU) is that they cannot learn if their activation values vanish.

In our thesis, we adopt the *leaky ReLU* [Maa13] to address this limitation. This neuron introduces a fixed constant slope parameter α to prevent vanishing activations for negative inputs. The leaky ReLU's activation function is defined as $f(z) = \max\{\alpha z, z\}$, and typically, a small value for the slope parameter is chosen, with a typical value being 0.01.

The upcoming section discusses the automatic training of the FNN's weights and biases. However, certain decisions regarding the FNN's architecture require careful consideration. These decisions encompass selecting suitable activation functions, determining the number of layers, and specifying the number of neurons per layer.

The complexity of the specific problem plays a pivotal role in determining the appropriate number of layers and neurons per layer. Networks with multiple hidden layers can yield effective results with significantly fewer neurons per layer, reducing

the number of parameters. Experimentation becomes essential to identify the optimal network architecture for our specific task. By iteratively experimenting with different configurations, we aim to discover the most suitable FNN architecture that best meets the requirements of our problem.

4.2.2 Training

Once we have determined an architecture for the FNN, our objective is to fine-tune the weights and biases of the network to better approximate the subgrid flux function $\mathcal{G}$.

In machine learning, the approximation quality can be described by how the neural network processes an *example*. In our case, an example can consist of two neighboring resolved states U_i and U_{i+1}, which the network uses to generate an approximation $\widetilde{G}_{i+1/2}$ of the subgrid flux $G_{i+1/2}$ defined by (4.5). We refer to $G_{i+1/2}$ as the *target* of the example (U_i, U_{i+1}). It is important to note that, due to our setup, the target is not necessarily unique, as it also depends on the subgrid states u_i and u_{i+1}, which are unknown to us. Additionally, an example may contain more resolved states than just two. The networks we use require an example of four resolved states, which will be discussed later.

To train the network, we expose it to an entire *training set* of examples, each associated with a corresponding target, with such a pair often referred to as a *data point*. Our goal is to enable the network to replicate the targets provided in this set. To evaluate the network's performance during training, we employ the *mean squared loss* or *quadratic cost function*.

Consider a training set consisting of data points $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $N \in \mathbb{N}$ represents the number of data points. The mean squared loss function $\mathcal{C}$ is then defined by

$$\mathcal{C}(\omega, b) := \frac{1}{N} \sum_{j=1}^{N} \|\widetilde{\mathcal{G}}(x_j, \omega, b) - y_j\|_2^2. \tag{4.7}$$

Here, ω denotes the collection of all weights in the FNN, and b represents all the biases. The function $\widetilde{\mathcal{G}}(x_j, \omega, b)$ corresponds to the output of the neural network when processing the input example x_j using the current weights and biases. As a measure of discrepancy between the output $\widetilde{\mathcal{G}}(x_j, \omega, b)$ and the target y_j, we employ the Euclidean norm $\|\cdot\|_2$.

The training process aims to minimize the cost function $\mathcal{C}$ concerning the network's weights and biases. We employ the *gradient descent* method to achieve this. Gradient descent is built upon the observation that a multi-variable function $\mathcal{C}$, like the

cost, decreases fastest at a specific point p in the direction of the negative gradient $-\nabla\mathcal{C}(p)$. The negative gradient points toward the steepest descent, where the cost function is expected to decrease most rapidly. By choosing a small enough step size or *learning rate* $\gamma > 0$, we can update the current point p using the following formula:

$$\hat{p} = p - \gamma\nabla\mathcal{C}(p). \tag{4.8}$$

The new point $\hat{p}$ obtained through this update will satisfy $\mathcal{C}(\hat{p}) \leq \mathcal{C}(p)$, resulting in a decrease in the function $\mathcal{C}$.

To initiate the optimization process, we start with an initial guess (ω_0, b_0) and employ the update formula (4.8) to obtain new parameters (ω_1, b_1). We repeat this procedure iteratively until a stopping criterion is met, typically based on a predefined number of iterations or when the improvement in the cost function becomes negligible. This iterative process generates a sequence of parameter sets $(\omega_k, b_k)_{k=0,\dots,K}$, ideally converging towards the desired minimum, representing the optimal weights and biases for the network.

Remark 4.1. Indeed, the mean squared error $\mathcal{C}$ defined in (4.7) may not be differentiable at every point when we use rectified linear units (ReLU). However, it is essential to note that the mean squared error remains differentiable almost everywhere, despite having a finite number of non-differentiable points. In practice, this is sufficient because gradient descent typically does not reach these non-differentiable points.

A substantial amount of data points is typically required to ensure our network can generalize well, extending its capability beyond the training set. However, employing gradient descent with the entire dataset results in a computationally expensive process due to the cost function's dependence on all data points. To address this issue, we can adopt a technique known as *stochastic gradient descent* (SGD), where the dataset is randomly divided into batches of a specific size, known as the *batch size*. The gradient of the cost function is approximated by averaging the gradients of local cost functions calculated on each batch. In neural networks, each optimization step is referred to as an *epoch*. It is worth noting that SGD drives most neural network learning.

Aside from SGD, other optimization algorithms have gained popularity and deserve mentioning. One such algorithm is *Adagrad*[Duc11], which employs adaptive learning rates for individual parameters during training. Other notable algorithms are *RMSProp* [Hin12] and *Adam*[Kin14], which also incorporate adaptive learning rates for individual parameters.

An essential algorithm to mention in the context of neural networks is *backpropagation*. Backpropagation is used to calculate the gradient of the cost function at a specific point during the training process. This algorithm employs a straightforward and cost-effective procedure, significantly enhancing the efficiency of training neural networks. Backpropagation gained widespread popularity following an experimental analysis published by Rumelhart et al. [Rum86]. More details regarding this technique can be found in Goodfellow et al. [Goo16].

Generalization is a critical aspect we must address to prevent the issue of overfitting. Overfitting occurs when a neural network becomes too specialized in reproducing the training data accurately, resulting in poor performance on unseen data. To tackle this problem, during the training process, when we update the model's weights and biases using the training set, we also evaluate its performance on a separate dataset called the *validation set*, which is explicitly not used for training.

The validation set serves as a good indicator of the network's ability to generalize to new data. If the training error decreases, but the validation error remains high or even increases, this strongly suggests overfitting is occurring. It is common to allocate 80% of the available data for training and 20% for validation. To achieve optimal results, we experiment with adjusting *hyperparameters*, such as those related to the network architecture (e.g., number of layers or neurons per layer) and the learning rate of the optimization algorithm, to obtain a minimal validation cost. This process helps us fine-tune the model and ensures that it performs well on the training data and on new, unseen data.

In order to address the issue of overfitting, numerous regularization techniques have been developed. One such straightforward method is *early stopping*. This technique suggests stopping the training process if the validation error fails to improve after a certain number of epochs.

Another approach is data normalization for both input and output. During the training phase, the network is trained using the normalized data. However, during prediction, we need to transform the inputs using the mean and standard deviation of the examples in the training data. This transformation ensures consistency with the data the network was trained on. Since the network learned with normalized labels during training, we use the mean and standard deviation of the labels from the training data to transform the output during prediction. Ensuring this normalization consistency allows the model's predictions to be aligned with the original scale of the output data, despite training being performed with normalized labels.

This thesis employs early stopping and data normalization as part of our regularization strategy. It is worth noting that various other regularization techniques have been introduced. One notable example is *dropout*, a method that randomly

deactivates certain units during training to prevent over-reliance on specific connections. Additionally, we incorporate *L2 regularization*, which adds a penalty term to the cost function to discourage the model from emphasizing specific features with substantial weights. For further exploration of different regularization methods, we recommend referring to [Goo16].

5 Monolithic Convex Limiting

Using a neural network to approximate subgrid fluxes introduces the risk of encountering issues such as under- and overshoots, which could result in negative heights and disrupt the algorithm's functionality. To address these challenges, we integrate the *monolithic convex limiting* (MCL) strategy proposed by Kuzmin [Kuz20] into our approach. The MCL approach enables us to effectively control and restrict the subgrid fluxes generated by the neural network, ensuring numerical admissibility and maintaining height positivity within the scheme.

Section 5.1 discusses the MCL approach's underlying principles and design philosophy. In Section 5.2, we conclude this chapter by presenting the specific limiting strategy we have implemented to stabilize the reduced model.

5.1 Design philosophy

The MCL strategy is built upon the flux-corrected transport (FCT) method, initially introduced by Boris and Book [Boo75] in the context of finite difference approximations. Later, Zalesak extended FCT to finite volume approximations in [Zal79]. Although initially proposed in a finite element setting, the MCL strategy can also be adapted for the finite volume framework, as demonstrated in [Kuz22]. For a more comprehensive MCL approach that applies to multidimensional SWE with non-flat bathymetry, we refer to [Haj22a] and [Haj22b].

Typically, the MCL approach adopts a high-order finite volume method as its target scheme. However, such methods often need to be revised for hyperbolic problems like the SWE due to spurious oscillations. To address this issue, we adopt a two-step strategy. Firstly, we select a low-order method that preserves key properties. This base discretization is the LLF method introduced in Chapter 3. The LLF method serves as the foundation for our approach. Next, we construct the desired target scheme by augmenting the LLF method with antidiffusive fluxes. These antidiffusive fluxes are determined as the difference between the numerical flux of the target scheme and that of the low-order LLF method. This combination allows us to restore some high-order accuracy. However, in order to effectively eliminate

the above-mentioned oscillations and enforce fundamental properties, MCL limits the raw antidiffusive fluxes.

Typically, MCL considers a high-order finite volume method as a target scheme. However, such methods are generally unsuitable for hyperbolic problems such as the SWE due to spurious oscillations, for example, caused by Gibbs phenomena. To overcome this issues, we first choose a low-order, property-preserving, but diffusive method as a base discretization. This is the LLF method we introduced in Chapter 3. We may obtain the original target scheme by adding antidiffusive fluxes defined by the difference between the numerical flux of the target and the low-order scheme. However, to eliminate or reduce the above oscillations, MCL limits the raw antidiffusive fluxes. Instead of a high-order finite volume method, we consider our reduced model.

We now aim to provide a more detailed specification of the MCL procedure, particularly concerning the raw antidiffusive flux. In this context, choosing a higher-order method as the target scheme is unnecessary. Instead, we can select our previously introduced reduced model from Chapter 4.

Let us fix $i = 1, \dots, N$. We denote the LLF numerical flux as $F^L_{i+1/2}$. Furthermore, the numerical flux of the reduced model, introduced in Section 4.1, is defined as

$$F^R_{i+1/2} := F^L_{i+1/2} + \widetilde{G}_{i+1/2}.$$

Here, $\widetilde{G}_{i+1/2}$ represents the subgrid flux calculated by the neural network, which we will refer to as the *network subgrid flux* from now on. As previously mentioned, the raw antidiffusive flux $F^A_{i+1/2}$ is obtained by subtracting the LLF flux from the reduced model's numerical flux:

$$F^A_{i+1/2} := F^R_{i+1/2} - F^L_{i+1/2} = \widetilde{G}_{i+1/2}.$$

Thus, it is evident that the network subgrid flux $\widetilde{G}_{i+1/2}$ serves as the raw antidiffusive flux in the MCL procedure.

To gain a better understanding of the objectives for the limited counterpart $\widetilde{G}^*_{i+1/2}$ of the network subgrid flux $\widetilde{G}_{i+1/2}$, we introduce limited bar states proposed by [Kuz20]:

$$\bar{u}^{*,\pm}_{i+1/2} := \bar{u}_{i+1/2} \pm \frac{\widetilde{G}^*_{i+1/2}}{\lambda_{i+1/2}}. \tag{5.1}$$

Here, $\bar{u}_{i+1/2}$ is given by (3.5). By using these limited bar states, we can rewrite the semi-discretization

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = \frac{1}{\Delta x}\left[(F_{i-1/2} + \widetilde{G}^*_{i-1/2}) - (F_{i+1/2} + \widetilde{G}^*_{i+1/2})\right], \qquad i = 1, \dots, N,$$

in a manner similar to (3.7), as follows:

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = \frac{1}{\Delta x}\big[\lambda_{i-1/2}(\bar{u}^{*,+}_{i-1/2} - u_i) + \lambda_{i+1/2}(\bar{u}^{*,-}_{i+1/2} - u_i)\big], \qquad i = 1, \dots, N. \tag{5.2}$$

When we discretize (5.2) in time using the forward Euler method, we obtain the following updated formula, proposed in [Gue16, Kuz20]:

$$\hat{u}_i = \Big(1 - \frac{\Delta t}{\Delta x}(\lambda_{i-1/2} + \lambda_{i+1/2})\Big)u_i + \frac{\Delta t}{\Delta x}\big(\lambda_{i-1/2}\bar{u}^{*,+}_{i-1/2} + \lambda_{i+1/2}\bar{u}^{*,-}_{i+1/2}\big), \tag{5.3}$$

where $i = 1, \dots, N$. This equation resembles (3.13), which we introduced in Section 3.4 to establish the HPP property of the fully discretized LLF scheme. If we assume that the time step Δt is sufficiently small, satisfying the CFL-like condition

$$\Delta t \leq \frac{1}{\lambda_{i-1/2} + \lambda_{i+1/2}}\Delta x,$$

and the limited bar states $\bar{u}^{*,+}_{i+1/2}$ and $\bar{u}^{*,-}_{i-1/2}$ lie in a specific convex *admissible set* $\mathcal{A}_i \subset \{(h, q) : h > 0\}$ that also contains u_i, then the update $\hat{u}_i$, as given by (5.3), is a convex combination of elements of the set $\mathcal{A}_i$. Consequently, we have $\hat{u}_i \in \mathcal{A}_i$, and the scheme satisfies the HPP property.

We choose the admissible set $\mathcal{A}_i$ in such a way that we enforce *numerical admissibility conditions*, which reduce or eliminate over and undershoots as mentioned above. By appropriately constraining $\bar{u}^{*,\pm}_{i+1/2}$ to ensure the HPP property of the fully discrete scheme (5.3), this also guarantees the HPP property of full discretizations of (5.2) using other SSP Runge-Kutta methods for time discretization. This is because each stage in an SSP Runge-Kutta method is a convex combination of a forward Euler update (5.3) and u_i.

In the following section, we will detail the bounds for the individual components of the limited bar states, which will provide a representation of the limited fluxes. The boundaries established here will implicitly define the admissible set $\mathcal{A}_i$, making a separate specification unnecessary.

5.2 Sequential limiting

Before we present the approach pursued by this chapter to enforce the numerical admissibility conditions, we highlight a few suggestions from the literature. Initially, limiting each component could be considered. However, this approach leads

to spurious oscillations [Löh87]. To address this issue, Löhner proposed a *synchronized limiter* in [Löh87], where the network subgrid flux $\widetilde{G}_{i+1/2}$ is adjusted using a correction factor $\alpha_{i+1/2} \in [0,1]$, i.e.,

$$\widetilde{G}^*_{i+1/2} = \alpha_{i+1/2}\widetilde{G}_{i+1/2}.$$

Examples of modern synchronized flux correction schemes can be found in [Loh16, Gue18, Paz21].

In this thesis, we employ the *sequential limiting* technique, first introduced by [Dob18] and further developed in [Haj19, Kuz20]. Instead of enforcing the numerical admissibility constraints on conserved quantities, height, and discharge, this approach enforces them on the derived ones, i.e., height, and velocity. Our presentation of the sequential limiting technique is based on [Kuz20, Sec. 5.1], where the author discusses the technique in the context of the Euler equations. We adapt this approach to the context of SWE, as described below.

Let us consider a fixed index $i = 1, \dots, N$. As a general rule, the admissible set $\mathcal{A}_i$ must contain the bar and limited bar states, which are associated with the edges of the cell C_i, i.e.,

$$\bar{u}_{i-1/2}, \bar{u}_{i+1/2} \in \mathcal{A}_i, \qquad \bar{u}^{*,-}_{i-1/2}, \bar{u}^{*,+}_{i+1/2} \in \mathcal{A}_i, \tag{5.4}$$

where the bar and limited bar states are defined in (3.5) and (5.1). The property in (5.4) for the i-th and $i+1$-th admissible set implies

$$\bar{u}^{*,+}_{i+1/2} \in \mathcal{A}_i, \qquad \bar{u}^{*,-}_{i+1/2} \in \mathcal{A}_{i+1}. \tag{5.5}$$

Let us start by limiting the height component. Motivated by (5.5), we design the limited height bar states $\bar{h}^{*,\pm}_{i+1/2}$ using the constraints

$$h_i^{\min} \le \bar{h}^{*,-}_{i+1/2} \le h_i^{\max}, \qquad h_{i+1}^{\min} \le \bar{h}^{*,+}_{i+1/2} \le h_{i+1}^{\max}, \tag{5.6}$$

where the local bounds $h_i^{\min}$ and $h_i^{\max}$ are defined by

$$h_i^{\min} := \min\{\bar{h}_{i-1/2}, \bar{h}_{i+1/2}\}, \qquad h_i^{\max} := \max\{\bar{h}_{i-1/2}, \bar{h}_{i+1/2}\}. \tag{5.7}$$

To derive constraints for the height component of the limited network subgrid flux $\widetilde{G}^{h,*}_{i+1/2}$, we rewrite (5.6) to

$$b^{h,L}_{i+1/2} \le -\widetilde{G}^{h,*}_{i+1/2} \le B^{h,L}_{i+1/2}, \qquad b^{h,R}_{i+1/2} \le \widetilde{G}^{h,*}_{i+1/2} \le B^{h,R}_{i+1/2}, \tag{5.8}$$

where

$$b^{h,L}_{i+1/2} = \lambda_{i+1/2}(h^{\min}_i - \bar{h}_{i+1/2}), \qquad B^{h,L}_{i+1/2} = \lambda_{i+1/2}(h^{\max}_i - \bar{h}_{i+1/2}),$$
$$b^{h,R}_{i+1/2} = \lambda_{i+1/2}(h^{\min}_{i+1} - \bar{h}_{i+1/2}), \qquad B^{h,R}_{i+1/2} = \lambda_{i+1/2}(h^{\max}_{i+1} - \bar{h}_{i+1/2}).$$

The first inequality in (5.8) constrains the outflow caused by the limited network subgrid flux $\widetilde{G}^{*,h}_{i+1/2}$ on the left-hand side of the edge $x_{i+1/2}$ and the second inequality constrains the inflow on the right-hand side.

A particular choice for the limited network subgrid flux $\widetilde{G}^{*,h}_{i+1/2}$ proposed by [Kuz20] is

$$\widetilde{G}^{h,*}_{i+1/2} = \begin{cases} \min\{\, \widetilde{G}^{h}_{i+1/2},\, -b^{h,L}_{i+1/2},\, B^{h,R}_{i+1/2} \,\}, & \text{if } \widetilde{G}^{h}_{i+1/2} \geq 0, \\ \max\{\, \widetilde{G}^{h}_{i+1/2},\, -B^{h,L}_{i+1/2},\, b^{h,R}_{i+1/2} \,\}, & \text{else.} \end{cases} \tag{5.9}$$

For alternative variations of these choices, we refer to [Kuz22].

Remark 5.1. The presented limiting technique for the height component is identical to the procedure employed for a scalar problem, as discussed in [Kuz20, Sec. 4]. The author uses the bounds

$$u^{\min}_i := \min\{u_{i-1}, u_i, u_{i+1}\}, \qquad u^{\max}_i := \max\{u_{i-1}, u_i, u_{i+1}\}.$$

These bounds are sufficient for the scalar case since the bar states $\bar{u}_{i-1/2}$ and $\bar{u}_{i+1/2}$, defined in (3.5), are guaranteed to be within the interval $[u^{\min}_i, u^{\max}_i]$. However, it is essential to note that this property is valid only for the bar states of scalar conservation laws. For systems like the SWE, the bar states must be included in the definition of $u^{\max}_i$ and $u^{\min}_i$.

Next, we aim to limit $\widetilde{G}^{*,q}_{i+1/2}$ to satisfy the numerical admissibility conditions for individual velocity components. To do this, we need to specify the local bounds. Let us first define the velocity bar state

$$\bar{v}_{i+1/2} := \frac{\bar{q}_{i+1/2}}{\bar{h}_{i+1/2}}. \tag{5.10}$$

Then, similar to the definition of the local height bounds in (5.7), we define the local velocity bounds

$$v^{\min}_i := \min\{\bar{v}_{i-1/2}, \bar{v}_{i+1/2}\}, \qquad v^{\max}_i := \max\{\bar{v}_{i-1/2}, \bar{v}_{i+1/2}\}.$$

With these bounds, we can now specify the numerical admissibility constraints that will be used to limit $\widetilde{G}^q_{i+1/2}$. The constraints are

$$\bar{h}^{*,-}_{i+1/2} v_i^{\min} \leq \bar{q}^{*,-}_{i+1/2} \leq \bar{h}^{*,-}_{i+1/2} v_i^{\max}, \qquad \bar{h}^{*,+}_{i+1/2} v_{i+1}^{\min} \leq \bar{q}^{*,+}_{i+1/2} \leq \bar{h}^{*,+}_{i+1/2} v_{i+1}^{\max}, \tag{5.11}$$

where the limited discharge bar state is defined according to (5.1):

$$\bar{q}^{*,\pm}_{i+1/2} = \bar{q}_{i+1/2} \pm \frac{\widetilde{G}^{q,*}_{i+1/2}}{\lambda_{i+1/2}}. \tag{5.12}$$

As proposed by Kuzmin [Kuz20], we rewrite (5.12) to

$$\begin{aligned}
\bar{q}^{*,\pm}_{i+1/2} &= \bar{q}_{i+1/2} \pm \frac{\widetilde{G}^{q,*}_{i+1/2}}{\lambda_{i+1/2}} \\
&= \bar{h}^{*,\pm}_{i+1/2} \bar{v}_{i+1/2} + (\bar{q}_{i+1/2} - \bar{h}^{*,\pm}_{i+1/2} \bar{v}_{i+1/2}) \pm \frac{\widetilde{G}^{q,*}_{i+1/2}}{\lambda_{i+1/2}} \\
&= \bar{h}^{*,\pm}_{i+1/2} \bar{v}_{i+1/2} + (\bar{q}_{i+1/2} - \bar{h}_{i+1/2} \bar{v}_{i+1/2}) \pm \frac{1}{\lambda_{i+1/2}} (\widetilde{G}^{q,*}_{i+1/2} - \widetilde{G}^{h,*}_{i+1/2} \bar{v}_{i+1/2}) \\
&= \bar{h}^{*,\pm}_{i+1/2} \bar{v}_{i+1/2} \pm \frac{1}{\lambda_{i+1/2}} (\widetilde{G}^{q,*}_{i+1/2} - \widetilde{G}^{h,*}_{i+1/2} \bar{v}_{i+1/2}),
\end{aligned}$$

where we used the definition of the limited height bar state (5.1) and the velocity bar state (5.10). Hence, we can also represent the limited discharge bar state (5.12) as

$$\bar{q}^{*,\pm}_{i+1/2} = \bar{h}^{*,\pm}_{i+1/2} \bar{v}_{i+1/2} \pm \frac{g^{q,*}_{i+1/2}}{\lambda_{i+1/2}},$$

where $g^{q,*}_{i+1/2}$ is a limited counterpart of

$$g^q_{i+1/2} = \widetilde{G}^q_{i+1/2} - \widetilde{G}^{h,*}_{i+1/2} \bar{v}_{i+1/2}. \tag{5.13}$$

We limit $g^q_{i+1/2}$ as described above. First, we rewrite the constraints (5.11) into the form

$$b^{q,L}_{i+1/2} \leq -g^{q,*}_{i+1/2} \leq B^{q,L}_{i+1/2}, \qquad b^{q,R}_{i+1/2} \leq g^{q,*}_{i+1/2} \leq B^{q,R}_{i+1/2},$$

where

$$\begin{aligned}
b^{q,L}_{i+1/2} &= \lambda_{i+1/2} \bar{h}^{*,-}_{i+1/2} (v_i^{\min} - \bar{v}_{i+1/2}), \qquad & B^{q,L}_{i+1/2} &= \lambda_{i+1/2} \bar{h}^{*,-}_{i+1/2} (v_i^{\max} - \bar{v}_{i+1/2}), \\
b^{q,R}_{i+1/2} &= \lambda_{i+1/2} \bar{h}^{*,+}_{i+1/2} (v_{i+1}^{\min} - \bar{v}_{i+1/2}), \qquad & B^{q,R}_{i+1/2} &= \lambda_{i+1/2} \bar{h}^{*,+}_{i+1/2} (v_{i+1}^{\max} - \bar{v}_{i+1/2}).
\end{aligned}$$

The limited counterpart of $g^q_{i+1/2}$ is then given by

$$g^{q,*}_{i+1/2} = \begin{cases} \min\{\, g^q_{i+1/2},\, -b^{q,L}_{i+1/2},\, B^{q,R}_{i+1/2} \,\}, & \text{if } g^q_{i+1/2} \geq 0, \\ \max\{\, g^q_{i+1/2},\, -B^{q,L}_{i+1/2},\, b^{q,R}_{i+1/2} \,\}, & \text{else.} \end{cases} \tag{5.14}$$

Inspired by the definition of $g^q_{i+1/2}$ in (5.13), we finally obtain the expression for the discharge component of the limited network subgrid flux

$$\widetilde{G}^{q,*}_{i+1/2} = g^{q,*}_{i+1/2} + \widetilde{G}^{h,*}_{i+1/2}\bar{v}_{i+1/2}. \tag{5.15}$$

Thus, the overall limited subgrid network flux $\widetilde{G}^*_{i+1/2}$ is given by its height component (5.9) and its discharge component (5.15).

6 Numerical Examples

This chapter focuses on implementing and analyzing two subgrid parameterizations introduced in Chapter 4. Specifically, we denote the first reduced model as NN-1, while the second is NN-2. These parameterizations differ in the degree of coarsening they apply. NN-1 simulates an LLF method with a mesh eight times finer, while NN-2 employs the coarsening degree of 40.

Additionally, we will enhance these reduced models by incorporating the MCL technique. MCL-NN-1 denotes the first limited reduced model equipped with MCL, and the second is MCL-NN-2. We may append the number of cells in the underlying mesh to distinguish the different mesh sizes. For example, LLF-400 indicates that the underlying mesh size for the LLF method is 400.

We present the first reduced model in Section 6.1. We provide insights into the architecture and training procedure of the underlying network, followed by testing the model by computing solutions for four initial conditions. The underlying network of the second model is similar to the first, but in Section 6.2, we highlight the differences in architecture and training procedure. We also perform the same test on the second model using the same four initial conditions. However, we observe spurious oscillations in the second model, which leads us to apply the MCL strategy to both reduced models in Section 6.3.

To further evaluate the models, we apply tests with a more significant number of initial conditions in Section 6.4. We examine the relative errors over time and even employ initial conditions outside the training set range to observe the performance of our models beyond their training range. However, it is worth noting that our models demonstrate slower computation times compared to the reference methods, and we address this issue in Section 6.5 by making necessary adjustments to the time step.

Throughout this chapter, we solve the SWE on the domain $[0, 100]$ up to a final time $T = 40$ while using the gravitational acceleration $g = 9.812$. Our simulations assume periodic boundary conditions, and to interpolate initial conditions, we calculate cell averages using a quadrature formula. We employ the Heun method for temporal discretization, as introduced in Section 3.3. We combine this method with a constant time step of the form $\Delta t = \nu \Delta x$, where $\nu > 0$ represents the *CFL*

parameter. Specifically, for LLF methods, we set $\nu = 0.1$, which satisfies the CFL condition (3.12) discussed in Section 3.4, in conjunction with the initial conditions used in this chapter. The CFL parameter for the reduced models will be further discussed later on.

All the results presented in this chapter can be replicated using the author's Python framework [Sch23].

6.1 First model

6.1.1 Network Architecture

We consider a coarse mesh consisting of 50 cells, and the objective of the reduced model NN-1 is to simulate the LLF method on a fine mesh comprising 400 cells. To achieve this, NN-1 employs a network $\widetilde{G}_1$ to approximate the subgrid flux function, as proposed in Section 4.1. In the context of SWE, the subgrid flux function has two components. Hence, we need two output neurons to represent these components.

In terms of the input layer, there are several choices available. Initially, a natural option would be to use the neighboring resolved modes U_i and U_{i+1}, as defined in (4.3). However, employing only these values resulted in relatively poor network performance. Therefore, we suggest considering a larger stencil of reduced modes by incorporating the values U_{i-1}, U_i, U_{i+1}, and U_{i+2} as inputs. This extended stencil has proven to yield improved results in approximating the subgrid flux function.

Indeed, similar observations were made by Alcala in [Alc21b], where a two-point stencil was found to be insufficient. This limitation might be attributed to the fact that using only two points does not provide enough information to estimate local smoothness accurately. We suspect that by increasing the stencil to incorporate more neighboring resolved modes, the network gains the ability to estimate whether a subgrid flux should be computed within a steep gradient or in a smoother region.

In summary, our network $\widetilde{\mathcal{G}}_1$ consists of 8 input neurons (two for each resolved mode) and two output neurons. Fig. 6.1 illustrates this network architecture, with the hidden neurons not shown for simplicity.

Regarding the rest of the architecture, the underlying network of NN-1, $\widetilde{\mathcal{G}}_1$, consists of *two* hidden layers, each with 32 neurons. We employ leaky ReLUs, as introduced in Section 4.2.1, in the hidden layers, with the slope parameter set to $\alpha = 0.01$. Furthermore, we use linear units in the output layer. Finally, each network provides data normalization in both the input and output layers. For further details on data normalization, please refer to Section 4.2.2.

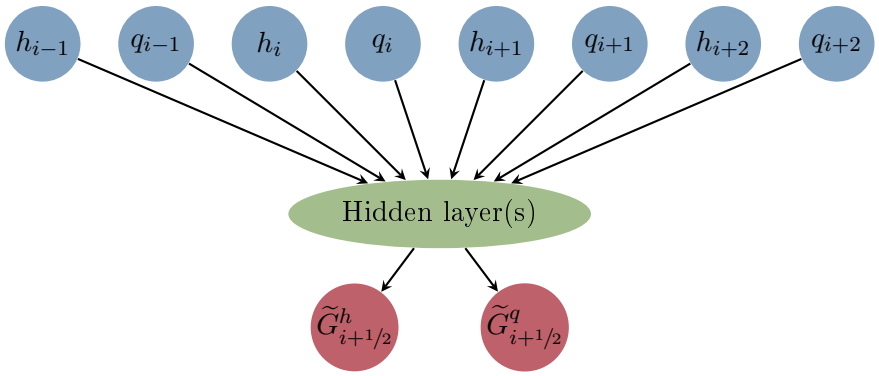


Figure 6.1: Visual representation of the neural networks to approximate the subgrid flux. To get the approximate value $\widetilde{G}_{i+1/2}$ for the subgrid flux $G_{i+1/2}$, the input layer processes four resolved modes. In the SWE context, each quantity consists of two values, one for the height and one for the discharge component.

6.1.2 Training

This section covers the training procedure of the NN-1 network $\widetilde{\mathcal{G}}_1$. To begin, we require a data set for training and validation, as mentioned in Section 4.2.2. The data set comprises examples and corresponding labels. As discussed in the previous section, the examples are stencils of resolved modes, while the labels correspond to the real subgrid fluxes $G_{i+1/2}$ defined by (4.5). Given that we assume periodic boundary conditions, we can consider subgrid fluxes $G_{i+1/2}$ of an edge $x_{i+1/2}$ for a fixed $i = 1, \dots, 50$.

For our particular scenario, we fix $i = 2$ and utilize the time series of U_0, U_1, U_2, and U_3 as examples, with $G_{2+1/2}$ as their corresponding labels. It is essential to reiterate that each variable has height and discharge components.

To construct time series of resolved modes and corresponding subgrid fluxes, we begin by generating 100 randomly oscillating initial data of the type

$$h_0(x) = H_0 + A_h \sin(2\pi k_h x/L + \varphi_h), \quad v_0(x) = V_0 + A_v \sin(2\pi k_v x/L + \varphi_v), \tag{6.1}$$

where $x \in [0, L]$ and $L = 100$. The height average is constant for all initial conditions, i.e., $H_0 = 2.0$. The remaining parameters are randomly generated following the distributions

$$\begin{aligned} V_0 &\sim \mathcal{U}(1,2), & A_h, A_v &\sim \mathcal{U}(0.2, 0.6), \\ k_h, k_v &\sim \mathcal{U}\{1,6\}, & \varphi_h, \varphi_v &\sim \mathcal{U}(0, 2\pi), \end{aligned} \tag{6.2}$$

where $\mathcal{U}(a,b)$ denotes the *continuous uniform distribution* between $a, b \in \mathbb{R}$ and $\mathcal{U}\{n,m\}$ represents the *discrete uniform distribution* between $n, m \in \mathbb{N}$. These

initial data configurations model fluids moving from left to right with an average height of 2.0.

Next, we calculate spatial and temporal discretization for a given initial condition. For the spatial discretization, we employ the LLF method, introduced in Section 3.2, on the fine mesh, which consists of 400 cells. Regarding temporal discretization, we use Heun's method, presented in Section 3.3, to perform time steps until reaching $T = 40$. During each Heun update, we compute the coarse cell averages U_1, U_2, U_3, U_4, as well as the subgrid flux $G_{2+1/2}$ using (4.3) and (4.5). We then add these values to our data set. We must note that we exclude the intermediate stages of Heun's method from the data set. This procedure for all 100 initial conditions results in a final data set containing approximately 160,000 data points.

Subsequently, we randomly partition the resulting data set into training and validation sets. The training set contains 80% of the data, while the validation set comprises the remaining 20%. Stochastic gradient descent is employed as the optimizer, with a batch size of 128. As a regularization technique, we implement early stopping. The network $\widetilde{\mathcal{G}}_1$ is trained for approximately 2000 epochs, using a learning rate of $\gamma = 0.01$. Please refer to Section 4.2.2 for further in-depth information.

The training process is performed using the Python library *skorch*[Tie17], which serves as a *PyTorch*[Pas19] wrapper, built upon *scikit-learn* [Ped11].

6.1.3 Tests

We begin with a preliminary test, and in Section 6.4, we subject the reduced model NN-1 to more comprehensive testing. For this initial test, we select four initial conditions of the form (6.1). The specific parameters used for each initial condition are listed in Tab. 6.1, and the initial conditions are depicted in Fig. 6.2.

(a) Height parameters

Initial Datum	H_0	A_h	k_h	φ_h
1	2.0	0.2	1	0.0
2	2.0	0.0	0	0.0
3	2.0	0.2	3	0.0
4	2.0	0.45	4	2.78

(b) Velocity parameters

Initial Datum	V_0	A_v	k_v	φ_v
1	1.0	0.0	0	0.0
2	1.0	0.5	1	0.0
3	1.0	0.0	0	0.0
4	1.1	0.5	3	4.5

Table 6.1: Parameters of the initial conditions of type (6.1) used for testing.

The first three initial conditions are designed to exhibit either constant velocity or constant height. However, the last initial condition is oscillatory in both compo-

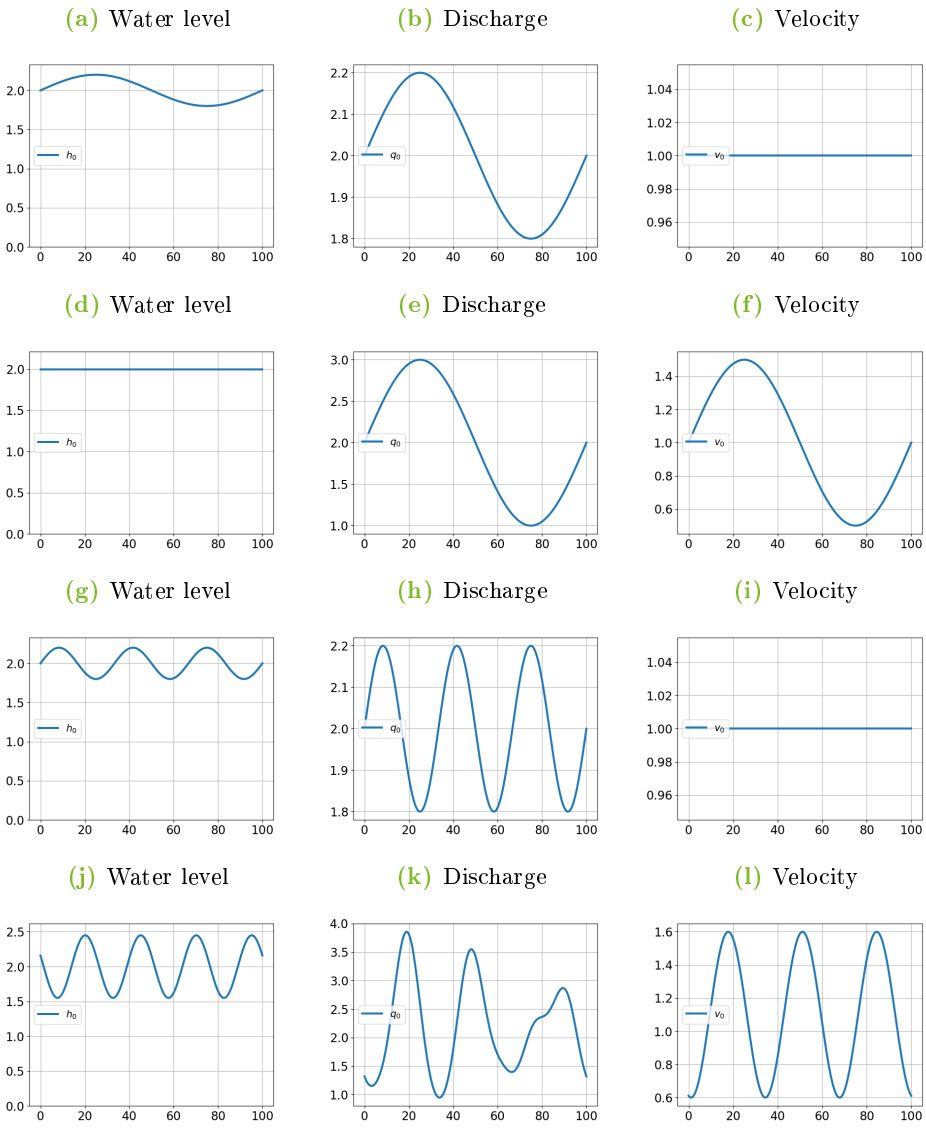


Figure 6.2: Initial conditions were used for testing the reduced models. Each row belongs to an initial condition of type (6.1), and the corresponding parameters are defined in Tab. 6.1.

nents. It serves as a challenging test case intended to assess the limitations of the second reduced model (refer to Section 6.2).

In Fig. 6.3, we illustrate discretizations at $T = 40$ obtained with Heun's method for the above-mentioned initial conditions. Alongside NN-1 and the target scheme (LLF method on 400 cells), we also calculate the unmodified LLF method on 50 cells. Additionally, we employ an LLF method on 10000 cells to compute a reference solution. For all LLF methods, we use a CFL parameter of $\nu = 0.1$. To ensure that the reduced model's time steps match the target model's, we adjust the CFL parameter and set it to $\nu = 0.0125$.

In Fig. 6.3, we observe that the reduced model NN-1 reproduces the target scheme for all initial conditions. Furthermore, it is evident from the results that the subgrid flux shows an antidiffusive behavior. This is evident when comparing the LLF scheme on 50 cells, which shows considerable diffusion, to the one used on 400 cells. However, the LLF method on 400 cells is still diffusive compared to the reference discretization. The subsequent reduced model aims to reproduce an LLF method on 2000 cells using the coarse mesh of only 50 cells to address this.

6.2 Second model

For the reduced model NN-2, we continue employing a coarse mesh of 50 cells. However, unlike NN-1, NN-2 aims to simulate the LLF method on a mesh comprising 2000 cells instead of 400 cells. The underlying network of NN-2 is denoted as $\widetilde{\mathcal{G}}_2$ and, similar to NN-1, it also focuses on approximating the subgrid flux function, as introduced in Section 4.1.

Similar to the architecture of the underlying network in the first reduced model, as discussed in Section 6.1.1, $\widetilde{\mathcal{G}}_2$ employs eight neurons in the input layer and two neurons in the output layer. It also uses leaky ReLUs as hidden neurons and linear units as output neurons. Furthermore, $\widetilde{\mathcal{G}}_2$ includes normalization layers for both the input and output. The primary distinction between $\widetilde{\mathcal{G}}_1$ and $\widetilde{\mathcal{G}}_2$ lies in the design of the hidden layers. While $\widetilde{\mathcal{G}}_1$ has two hidden layers, each with 32 neurons, $\widetilde{\mathcal{G}}_2$ consists of three hidden layers, each with 128 neurons.

The data set used for training and validation is generated similarly to that of the first model. However, since we use a finer mesh than the first example, the time steps become smaller. As a result, the data set contains approximately 800,000 data points, surpassing the size of the first data set.

The training of the network $\widetilde{\mathcal{G}}_2$ is similar to the one of the network $\widetilde{\mathcal{G}}_2$, as described in Section 6.1.2. We initiate by randomly partitioning the data set into training

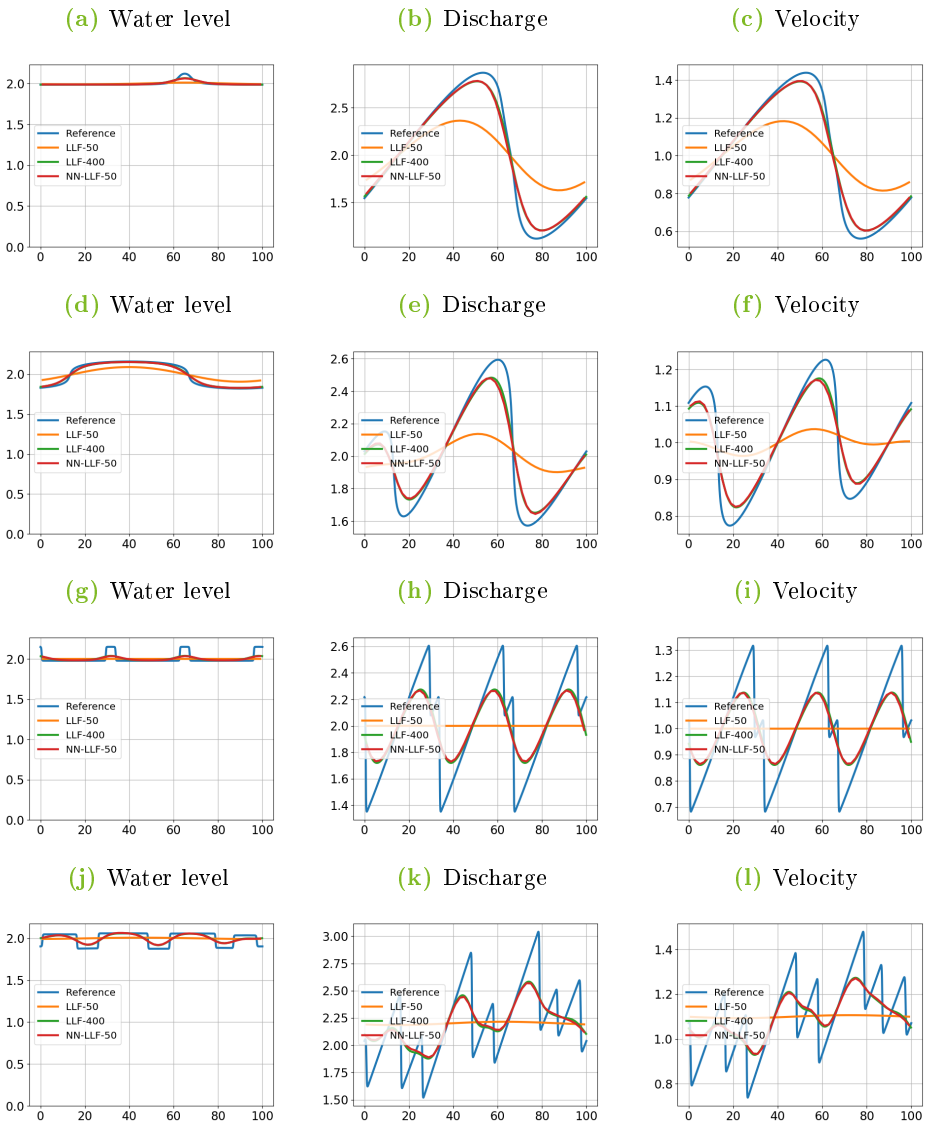


Figure 6.3: SWE with initial conditions illustrated in Fig. 6.2. Discretizations at $T = 40$ were obtained with Heun's method on periodic meshes. The reference discretization is generated with an LLF scheme on a uniform mesh consisting of 10000 cells.

and validation sets. The training set contains 80% of the data, while the validation set comprises the remaining 20%. Stochastic gradient descent is employed as the optimizer, with a batch size of 128. As a regularization technique, we implement early stopping. However, there is a slight difference in the training approach compared to $\widetilde{\mathcal{G}}_1$. We train the network $\widetilde{\mathcal{G}}_2$ twice using different learning rates and early stopping. Initially, we train for almost 500 epochs with a learning rate of $\gamma = 0.1$. Subsequently, we continue training for another 2000 epochs with a lower learning rate of $\gamma = 0.001$.

After training, we can perform the preliminary test proposed in Section 6.1.3. We select the same four initial conditions of the form (6.1) and specific parameters listed in Tab. 6.1. Fig. 6.2 depicts these initial conditions.

In Fig. 6.4, we present discretizations at $T = 40$ obtained using Heun's method for the mentioned initial conditions. We compare the second reduced model NN-2 with the target scheme (LLF method on 2000 cells) and the first reduced model NN-1. Additionally, we employ an LLF method on 10000 cells to compute a reference solution. To ensure that the reduced model's time steps match the target scheme, we adjust the CFL parameter and set it to $\nu = 0.0025$. As discussed earlier, the CFL parameter for the LLF schemes is chosen as $\nu = 0.1$, and for NN-1, it is set to $\nu = 0.0125$.

As expected, the discretization provided by NN-2 demonstrates lower diffusivity than NN-1. However, NN-2 encounters challenges in accurately reproducing the target scheme (LLF method on 2000 cells). In particular, NN-2 deviates the most from the target discretization at local extrema or points with steep gradients. This discrepancy may be attributed to the rarity of associated events in the training data, as the data only provides the time evolution of discretizations from one specific point.

In specific cases, such as the last initial condition, the scheme becomes unstable (refer to Fig. 6.4j, Fig. 6.4k, and Fig. 6.4l). Interestingly, this unstable behavior is observed in only a small subset of initial conditions. Initial conditions with high oscillations may likely lead to such behavior.

6.3 Limited models

As we have noticed spurious oscillations in a few cases with the second reduced model NN-2, we apply the MCL strategy (see Section 5) to both models and perform the same preliminary test as we did for the unlimited counterparts in Sections 6.1.3

Figure 6.4: SWE with initial conditions illustrated in Fig. 6.2. Discretizations at $T = 40$ were obtained with Heun's method on periodic meshes. The reference discretization is generated with an LLF scheme on a uniform mesh consisting of 10000 cells.

and 6.2. We denote the limited versions of NN-1 and NN-2 as MCL-NN-1 and MCL-NN-2, respectively.

We select the same four initial conditions of the form (6.1) and specific parameters listed in Tab. 6.1, where the initial conditions are depicted in Fig. 6.2.

We begin investigating MCL-NN-1. In Fig. 6.5, we illustrate discretizations at $T = 40$ obtained using Heun's method for the mentioned initial conditions. We compare the MCL-NN-1 with its unlimited counterpart NN-1 and the target scheme (LLF method on 400 cells). As discussed earlier, the CFL parameter for the LLF scheme is $\nu = 0.1$, and for MCL-NN-1 and NN-1, it is set to $\nu = 0.0125$. The differences between the MCL version of the model and the original model are insignificant, but the MCL version tends to be slightly more diffusive at local extrema.

Let us move on to MCL-NN-2. In Fig. 6.6, we showcase discretizations at $T = 40$ obtained using Heun's method for the mentioned initial conditions. We compare MCL-NN-2 with its unlimited counterpart NN-2 and the target scheme (LLF method on 2000 cells). As previously discussed, the CFL parameter for the LLF scheme is $\nu = 0.1$, and for MCL-NN-2 and NN-2, it is set to $\nu = 0.0025$.

For the first three initial conditions, similar to MCL-NN-1, the differences between MCL-NN-2 and NN-2 are not substantial. However, MCL-NN-2 tends to be slightly more diffusive at local extrema while also exhibiting a smoother behavior. Additionally, in Fig. 6.6j, Fig. 6.6k, and Fig. 6.6l, we can observe that MCL successfully stabilizes NN-1 for the fourth initial condition. Let us examine this case more closely.

Fig. 6.7 depicts discretizations of MCL-NN-2 and the target LLF scheme for the fourth initial condition without employing NN-2.

The discretization of MCL-NN-2 is less accurate than for the other initial conditions. However, there are noticeable differences between the height and momentum discretizations. Compared to the discharge discretization, MCL-NN-2 simulates the height discretization of LLF-2000 relatively accurately. On the other hand, MCL-NN-2 faces challenges in resolving all local extrema of the LLF-2000 discharge discretization, likely due to its oscillatory nature. Nevertheless, MCL-NN-2 reproduces the overall shape of the LLF-2000 discharge discretization, and approximately every second extremum is resolved relatively accurately. The intermediate regions are adequately approximated on average.

Figure 6.5: SWE with initial conditions illustrated in Fig. 6.2. Discretizations $T = 40$ were obtained with Heun's method on periodic meshes.

Figure 6.6: SWE with initial conditions illustrated in Fig. 6.2. Discretizations at $T = 40$ were obtained with Heun's method on periodic meshes.

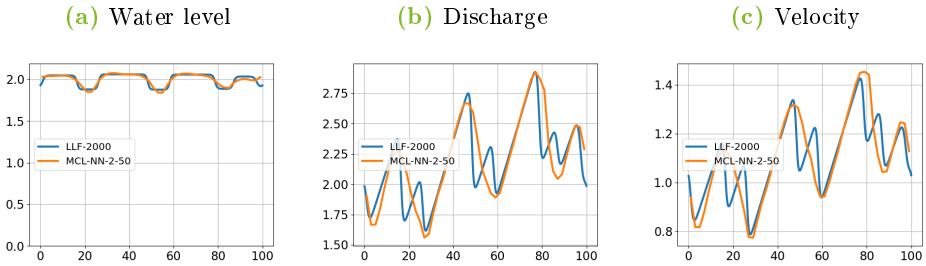


Figure 6.7: SWE with initial condition illustrated in Fig. 6.2j, Fig. 6.2k and Fig. 6.2l. Discretizations at $T = 40$ obtained with Heun's method on periodic meshes.

6.4 Performance outside the training range

This section presents more comprehensive tests of our reduced models, specifically focusing on their performance outside the training range. Since the second reduced model NN-2 is potentially unstable, as observed in Section 6.2, we focus on the first reduced model NN-1 and the limited counterparts of both models, MCL-NN-1 and MCL-NN-2.

Our main objective is to analyze the accuracy evolution of these models across various initial conditions. We assess accuracy by calculating the relative error concerning the corresponding LLF target scheme. Here is a more detailed explanation of our approach:

1. For a given set of initial conditions, we compute discretizations using the following methods: NN-1-50, MCL-NN-1-50, LLF-400, MCL-NN-2-50, and LLF-2000. We employ Heun's method and evaluate these discretizations up to $T = 40$, considering the number of cells in the underlying mesh as indicated in their respective names. We use the CFL parameters as previously defined: $\nu = 0.1$ for LLF methods, $\nu = 0.0125$ for NN-1 and MCL-NN-1, and $\nu = 0.0025$ for MCL-NN-2.

2. Since our reduced models aim to replicate the discretizations of the corresponding LLF schemes on a coarse mesh, we average the LLF method discretizations using the average operation (4.3).

3. Next, we determine the relative L^2 errors between each reduced model's discretization and the corresponding averaged LLF discretization for each time step. This process yields a time series of relative errors for each initial condition.

4. Finally, we consider either the mean or the maximum of the relative errors for each time point in the time series as our measure of accuracy.

By following this methodology, we can comprehensively assess and compare the accuracy performance of the reduced models under various initial conditions. We begin by generating time series of relative errors for initial conditions within the training, serving as a reference. Subsequently, we systematically explore initial conditions outside the training range and compare their relative error evolutions with the reference errors.

Reference errors

For the reference errors, we generate 20 initial conditions of type (6.1), where all parameters are randomly generated from the distribution specified in (6.2). It is important to emphasize that these initial conditions are distinct from the ones we used in the training set. Fig. 6.8 illustrates the time series of the mean and maximum relative L^2-error of the height and discharge discretizations. The maximum relative L^2 error is computed for each time step and, therefore, is unrelated to a specific initial condition.

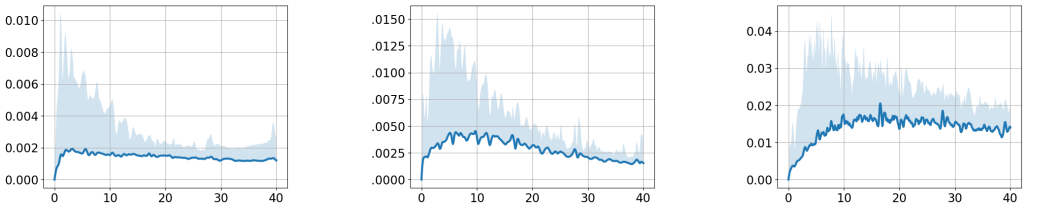
Firstly, it is evident that the relative error evolutions of the height and discharge discretizations exhibit similar patterns in the profile, with the errors of the discharge discretizations being approximately 2.5 times larger than those of the height discretizations. As a result, we will focus on the time series of the relative error concerning the height discretizations in the subsequent analysis.

Upon examining the relative error plots, we observe in all cases that the maximum relative errors initially rise steeply and then gradually decrease. In contrast, the mean relative errors reach a stable level relatively quickly before experiencing a slow decline. This behavior is consistent with what we will observe in the subsequent tests. We believe that this phenomenon is linked to the diffusive nature of the LLF method, which may lead to the observed error patterns.

In Fig. 6.8a, we observe the most negligible errors for NN-1, with the relative mean error being approximately 0.2% for the height discretization. Moving to the limited version MCL-NN-1, the performance is slightly worse, but the mean error remains below 1%, as depicted in Fig. 6.8b. This outcome is expected since the MCL strategy limits the subgrid flux, reducing its antidiffusive nature and increasing the error.

However, in Fig. 6.8c, we see a significant increase in errors for MCL-NN-2. The mean error of the height discretization exceeds 2%, and the maximum error is below 5%. These results are consistent with our observations from the previous section,

(a) NN-1, rel. h-error (b) MCL-NN-1, rel. h-error (c) MCL-NN-2, rel. h-error



(d) NN-1, rel. q-error (e) MCL-NN-1, rel. q-error (f) MCL-NN-2, rel. q-error

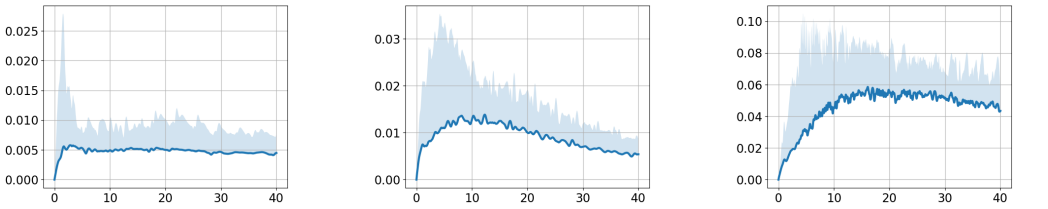


Figure 6.8: Time evolution of the mean (represented by the thick blue line) and maximum relative height and discharge L^2-errors among 20 reduced model's discretizations and the corresponding averaged methods they aim to replicate. The initial data used are of type (6.1), where all parameters are randomly generated from the distributions specified in (6.2). Reduced model discretizations were obtained with Heun's method on periodic meshes consisting of 50 cells.

indicating that the discretizations are not as accurate as those of NN-1 and MCL-NN-1.

Test

After generating the reference time series of relative errors, our next step is to test the reduced models beyond the training range. For this purpose, we create initial conditions of type (6.1), where one parameter is not generated from the distributions specified in (6.2).

First, we select initial conditions with lower and higher average height parameters than those encountered in the training set. Subsequently, we consider initial conditions with larger height amplitudes than what was seen during training. Lastly, we investigate scenarios where the wave number of the height component is increased beyond the range covered in the training data.

Height average variation

We generate two sets of 20 initial conditions of type (6.1), where all parameters except the height average H_0 are randomly drawn from the distributions specified in (6.2). For the first set, we set the height average H_0 to be 1.6; for the second set, we use $H_0 = 2.4$. Subsequently, we calculate the mean and maximum relative error evolutions described earlier in this section.

Fig. 6.9 illustrates the mean and maximum relative error evolution of the height discretizations up to $T = 40$. In the first row, we display the results as a reference, which are identical to Fig. 6.8a, Fig. 6.8b, and Fig. 6.8c.

As the initial conditions lie outside the training set range, the relative errors are expected to increase. Indeed, we observe more significant relative errors for the NN-1 and MCL-NN-1 methods when $H_0 = 1.6$ compared to the reference results. Specifically, for NN-1, the mean relative error in Fig. 6.9d is approximately twice as significant as the mean error in Fig. 6.9a, starting from $T = 10$ or even earlier. Nevertheless, the mean errors, still below 0.5%, remain smaller than the mean relative errors of MCL-NN-1 and MCL-NN-2. Similarly, for MCL-NN-1, it appears that the errors are roughly twice as large, although a direct comparison is difficult due to the different scaling of the y-axis in Fig. 6.9e and Fig. 6.9b.

Furthermore, we notice an increased number of outliers up to $T = 20$ in Fig. 6.9d and Fig. 6.9e, compared to the corresponding references. However, these outliers quickly level off and have no significant impact on the mean errors.

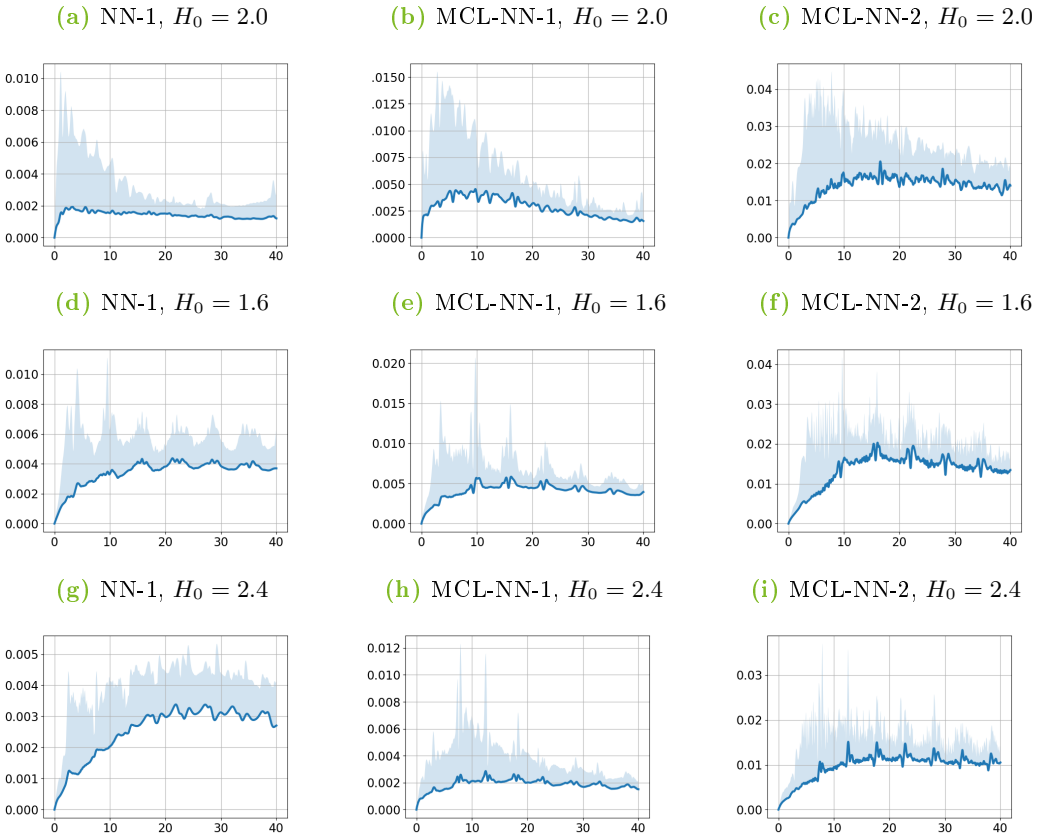


Figure 6.9: Time evolution of the mean (represented by the thick blue line) and maximum relative height L^2-errors among 20 reduced model's discretizations and the corresponding averaged methods they aim to replicate. The initial data used are of type (6.1), where all parameters except the height average H_0 are randomly generated from the distributions specified in (6.2). The parameter H_0 is selected as indicated in the headings. Reduced model discretizations were obtained with Heun's method on periodic meshes consisting of 50 cells.

For MCL-NN-2, the situation is quite different. It is challenging to determine whether the error in Fig. 6.9f is larger or smaller than the reference in Fig. 6.9c. Moreover, the maximum errors tend to be smaller overall.

For $H_0 = 2.4$, the mean relative error evolution of NN-1 illustrated in Fig. 6.9g is, as expected, more significant than the reference. However, the maximum error remains consistently low. Surprisingly, the errors of the limited versions in Fig. 6.9h and Fig. 6.9i become even smaller compared to the reference results. In fact, the mean errors of MCL-NN-1 are mostly minor than those of NN-1, but the same cannot be said for the maximum errors.

These results indicate that the behavior of the reduced models concerning initial conditions outside the training range can be nuanced. While NN-1 generally exhibits more significant errors as expected, the limited versions MCL-NN-1 and MCL-NN-2 surprisingly show improvements in specific scenarios, especially considering the mean errors. However, the discrepancy between mean and maximum errors suggests that there might be outliers or specific cases where MCL-NN-1 and MCL-NN-2 do not perform as well as NN-1.

Nevertheless, our models accurately reproduce their respective target schemes for other height averages in the initial conditions, similar to those encountered during training.

Height amplitude variation

Next, we generate three sets of 20 initial conditions of type (6.1), where all parameters except the height amplitude A_h are randomly drawn from the distributions specified in (6.2). For each set, we set the height amplitude A_h to different values: 0.8 for the first set, 1.0 for the second set, and 1.5 for the last set. Subsequently, we calculate the mean and maximum relative error evolutions described earlier in this section.

Fig. 6.10 illustrates the mean and maximum relative error evolution of the height discretizations up to $T = 40$ for the discretizations of the reduced models. Again, in the first row, we display the results as a reference, which are identical to Fig. 6.8a, Fig. 6.8b, and Fig. 6.8c. These reference plots serve as a basis for comparison with the subsequent results obtained from the three sets of initial conditions with varying height amplitudes.

In all cases, we observe that the relative errors are initially quite large but gradually diminish over time, eventually approaching the errors observed in the reference

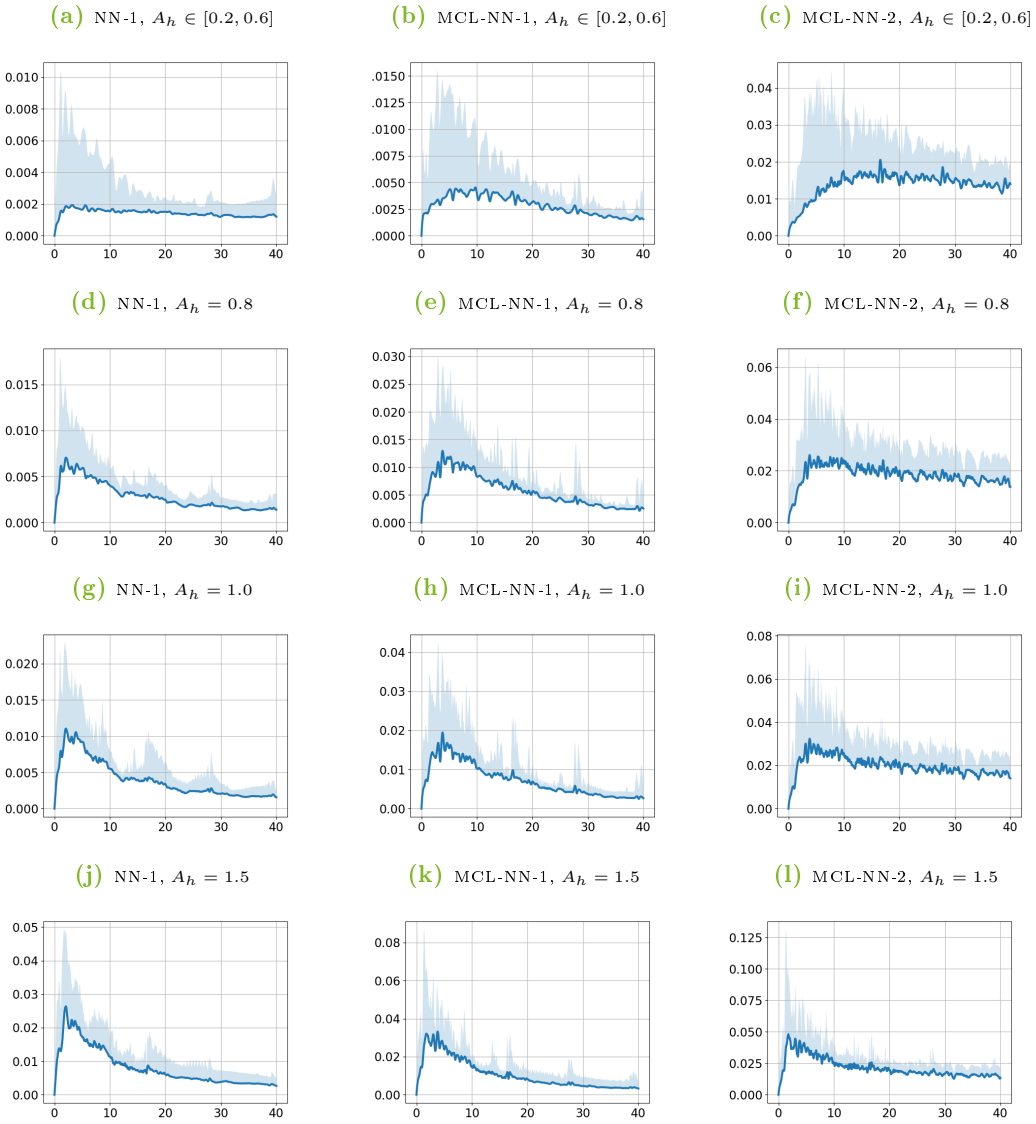


Figure 6.10: Time evolution of the mean (represented by the thick blue line) and maximum relative height L^2-errors among 20 reduced model's discretizations and the corresponding averaged methods they aim to replicate. The initial data used are of type (6.1), where all parameters except the height amplitude A_h are randomly generated from the distributions specified in (6.2). The parameter A_h is selected as indicated in the headings. Reduced model discretizations were obtained with Heun's method on periodic meshes consisting of 50 cells.

plots. As expected, the errors become more significant as we increase the value of A_h. However, with time, these errors tend to diminish.

As mentioned earlier, this phenomenon may be linked to the diffusive nature of the LLF method. The added diffusion in the discretizations causes the height amplitude to decrease over time as the network encounters modes it is already familiar with. This leads to a gradual reduction in errors as the model gains better knowledge of the problem dynamics.

Furthermore, we find that MCL-NN-2 exhibits a less distinct increase in errors compared to NN-1 or MCL-NN-1 compared to the reference. This suggests that MCL-NN-2 demonstrates improved error behavior in this context.

Overall, our models demonstrate a relatively accurate reproduction of their target schemes for advanced times and various height amplitudes in the initial conditions, similar to those encountered during training.

Height wave number variation

Finally, we create two sets of 20 initial conditions of type (6.1), where all parameters except the wave number k_h are randomly generated from the distributions (6.2). For the first set, we set the height wave number k_h to be 10; for the second set, we use $k_h = 15$. Subsequently, we calculate the mean and maximum relative error evolutions described earlier in this section.

Fig. 6.11 illustrates the mean and maximum relative error evolution of the height discretizations up to $T = 40$ for the discretizations of the reduced models. Again, in the first row, we display the results as a reference, which are identical to Fig. 6.8a, Fig. 6.8b, and Fig. 6.8c.

Once again, we observe a similar pattern where the relative errors become very large initially and then diminish as time progresses. This behavior may be linked to the diffusive nature of the LLF method. As time elapses, the added diffusion in the discretizations causes the errors to reduce, leading to more accurate predictions over time.

6.5 Relaxation of the CFL condition

Until now, we have employed small CFL parameters for our reduced models to obtain time steps that closely match those of the process they aim to reproduce. As

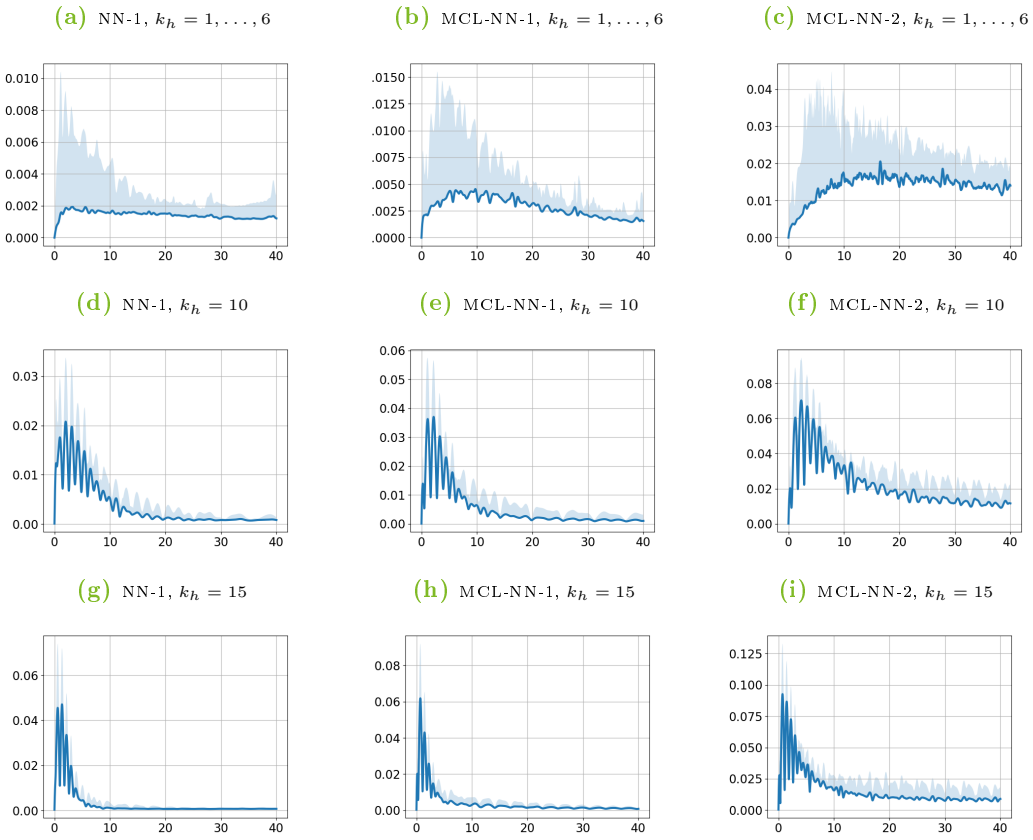


Figure 6.11: Time evolution of the mean (represented by the thick blue line) and maximum relative height L^2-errors among 20 reduced model's discretizations and the corresponding averaged methods they aim to replicate. The initial data used are of type (6.1), where all parameters except the height wave number k_h are randomly generated from the distributions specified in (6.2). The parameter k_h is selected as indicated in the headings. Reduced model discretizations were obtained with Heun's method on periodic meshes consisting of 50 cells.

a reminder, for NN-1 and MCL-NN-1, we have been using $\nu = 0.0125$, and for MCL-NN-2, we have been using $\nu = 0.0025$. This approach has resulted in an increased number of time steps proportionate to reducing the number of degrees of freedom. Consequently, the current procedures are not more efficient than the original ones. To improve efficiency, an increase in the time step is desirable. Therefore, we test more extensive CFL parameters in this Section to aim for a more efficient model.

To test different CFL parameters, we proceed as in the section before. We create two sets of 20 initial conditions of type (6.1), where all parameters are randomly generated from the distributions (6.2). Subsequently, we calculate the mean and maximum relative error evolutions described earlier in the previous section. For the first set of initial conditions, we use a CFL parameter of $\nu = 0.05$, while for the second set, we use a CFL parameter of $\nu = 0.1$. By varying the CFL parameters in this way, we aim to observe how the choice of time step size influences the accuracy of the reduced models. This analysis will help us determine the optimal CFL parameter.

Fig. 6.12 illustrates the mean and maximum relative error evolution of the height discretizations up to $T = 40$ for the discretizations of the reduced models. In the first row, we display the results as a reference, which are identical to Fig. 6.8a, Fig. 6.8b, and Fig. 6.8c.

We observe that when using $\nu = 0.05$ as the CFL parameter, the relative error evolution of all reduced methods differs very little from the reference results. This indicates that the reduced models can maintain a high level of accuracy with a relatively larger time step compared to the original LLF methods.

However, when we increase the CFL parameter to $\nu = 0.1$, the errors in the reduced discretizations start to increase. These relative errors are still relatively small, but it becomes apparent that more accurate results can be obtained if we do not use the same CFL parameter as the LLF methods.

In summary, while the reduced models can maintain accuracy with a more significant time step of $\nu = 0.05$, a larger CFL parameter of $\nu = 0.1$ may lead to slightly higher errors.

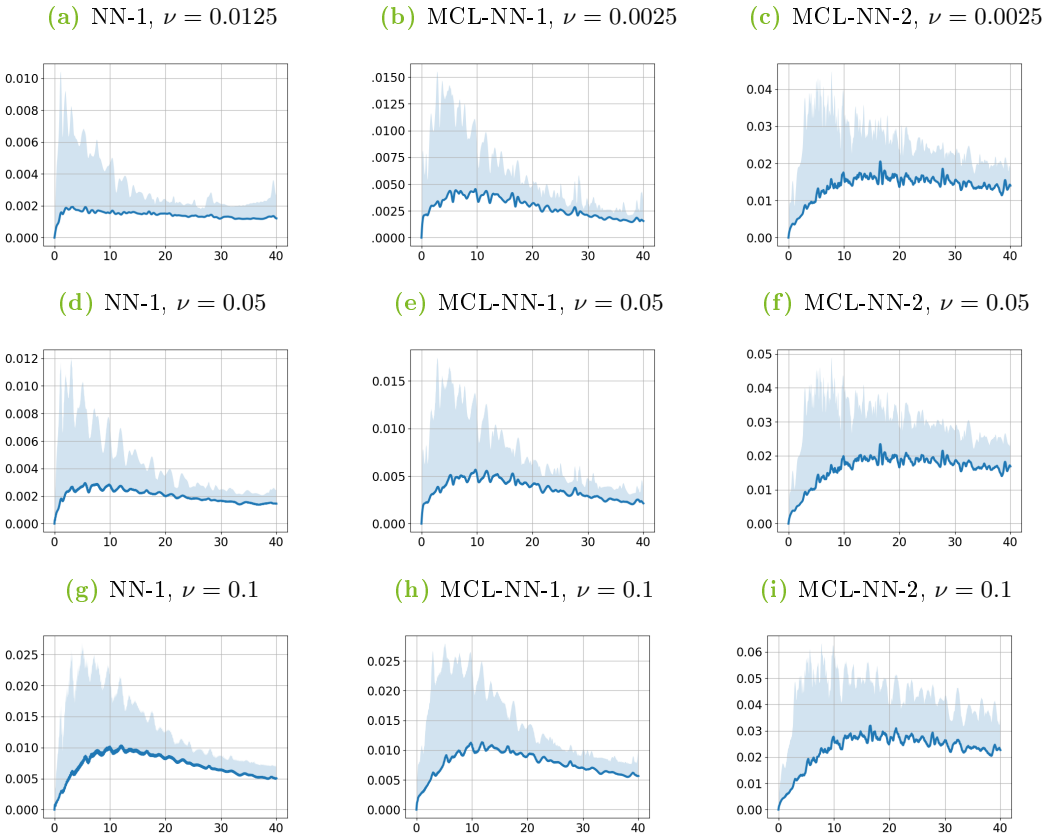


Figure 6.12: Time evolution of the mean (represented by the thick blue line) and maximum relative height L^2-errors among 20 reduced model's discretizations and the corresponding averaged methods they aim to replicate. The initial data used are of type (6.1), where all parameters except the are randomly generated from the distributions specified in (6.2). Discretizations obtained with Heun's method on periodic meshes consisting of 50 cells, where the CFL parameter ν is selected as indicated in the headings.

7 Conclusions

In this thesis, we developed a reduced model using the LLF method for SWE by approximating subgrid fluxes with feedforward neural networks. We trained two networks to ensure the reduced model could accurately reproduce LLF methods on meshes eight or 40 times finer than the available mesh for specific problems.

However, we observed that the first reduced model resolves the target LLF scheme more accurately than the second reduced model. This outcome is not surprising, given that the task of the second model is more challenging. The second reduced model encounters difficulties capturing the underlying dynamics, particularly around local extrema or points with steep gradients. This discrepancy may be attributed to the scarcity of associated events in the training data, as the data only provides the time evolution of discretizations from one specific point. We even encountered instances of initial conditions with high oscillations within the training range of the second network, resulting in instabilities.

To address this issue, we implemented the monolithic convex limiting strategy proposed by Kuzmin [Kuz20], which successfully resolved the difficulty. This finding underscores the importance of developing procedures to ensure a data-driven model's specific numerical and physical properties. Instabilities can manifest without a complete understanding of their underlying reasons.

We proceeded by investigating the first reduced model and the limited versions of the first and second reduced models. Our findings revealed that our models can be applied to initial data that falls outside the training range of the networks. Although the models perform less accurately than for problems within their familiar domain, they perform reasonably well. Interestingly, the limited second reduced model appears less susceptible to issues with unfamiliar data. In specific scenarios, we even found instances where the limited versions of the reduced models showed surprising improvements.

We also explored the impact of relaxing the CFL parameters to create more efficient models. By increasing the CFL parameter to 0.05 for all models of interest, we found that we could maintain accuracy without sacrificing performance, resulting in a significant speed-up of the procedure. However, it remains to be seen how the method behaves outside the training range when using a larger CFL parameter.

Further investigation is needed to determine if the model's performance remains consistent under these conditions.

Furthermore, our current findings raise several questions that require further investigation. One key aspect is exploring the possibility of obtaining other physical properties, such as entropy inequalities, through our approach. Moreover, our method suits conservation laws, so we can consider parametrizing other conservation laws, like the Euler equation, using the same approach. Extending this approach to solve SWE with additional complexities, including bathymetry, friction, Coriolis forces, and accommodating wetting and drying states is conceivable. Additionally, it would be valuable to explore the applicability of our method in two dimensions, as it could provide insights and enable performance evaluation in more intricate scenarios.

References

[Alc21a] J. Alcala, I. Timofeyev (2021) *Subgrid-scale parametrization of unresolved scales in forced Burgers equation using generative adversarial networks (GAN)* Theoretical and Computational Fluid Dynamics **35**: 875–894

[Alc21b] J. S. Alcala (2021) *Subgrid-scale parametrization of unresolved processes* Ph.D. thesis

[Arj17] M. Arjovsky, S. Chintala, L. Bottou (2017) *Wasserstein Generative Adversarial Networks* in D. Precup, Y. W. Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017* vol. 70 of *Proceedings of Machine Learning Research* 214–223 PMLR URL `http://proceedings.mlr.press/v70/arjovsky17a.html`

[Bec08] P. Bechtold, M. Köhler, T. Jung, F. Doblas-Reyes, M. Leutbecher, M. J. Rodwell, F. Vitart, G. Balsamo (2008) *Advances in simulating atmospheric variability with the ECMWF model: From synoptic to decadal time-scales* Quarterly Journal of the Royal Meteorological Society: A journal of the atmospheric sciences, applied meteorology and physical oceanography **134**: 1337–1351

[Beu19] T. Beucler, S. Rasp, M. Pritchard, P. Gentine (2019) *Achieving conservation of energy in neural network emulators for climate modeling* ArXiv preprint **abs/1906.06622** URL `https://arxiv.org/abs/1906.06622`

[Bol19] T. Bolton, L. Zanna (2019) *Applications of deep learning to ocean data inference and subgrid parameterization* Journal of Advances in Modeling Earth Systems **11**: 376–399

[Boo75] D. L. Book, J. P. Boris, K. Hain (1975) *Flux-corrected transport II: Generalizations of the method* Journal of Computational Physics **18**: 248–283

[Bre19] N. D. Brenowitz, C. S. Bretherton (2019) *Spatially extended tests of a neural network parametrization trained by coarse-graining* Journal of Advances in Modeling Earth Systems **11**: 2728–2744

[Cyb89] G. Cybenko (1989) *Approximation by superpositions of a sigmoidal function* Mathematics of control, signals and systems **2**: 303–314

[Dob18] V. Dobrev, T. Kolev, D. Kuzmin, R. Rieben, V. Tomov (2018) *Sequential limiting in continuous and discontinuous Galerkin methods for the Euler equations* Journal of Computational Physics **356**: 372–390

[Duc11] J. Duchi, E. Hazan, Y. Singer (2011) *Adaptive subgradient methods for online learning and stochastic optimization.* Journal of machine learning research **12**

[Far11] R. Farneti, P. R. Gent (2011) *The effects of the eddy-induced advection coefficient in a coarse-resolution coupled climate model* Ocean Modelling **39**: 135–145

[Goo14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, Y. Bengio (2014) *Generative Adversarial Nets* in Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada* 2672–2680 URL `https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html`

[Goo16] I. Goodfellow, Y. Bengio, A. Courville (2016) *Deep learning* MIT press

[Got01] S. Gottlieb, C.-W. Shu, E. Tadmor (2001) *Strong stability-preserving high-order time discretization methods* SIAM review **43**: 89–112

[Got11] S. Gottlieb, D. Ketcheson, C.-W. Shu (2011) *Strong stability preserving Runge-Kutta and multistep time discretizations* World Scientific

[Gri15] S. M. Griffies, M. Winton, W. G. Anderson, R. Benson, T. L. Delworth, C. O. Dufour, J. P. Dunne, P. Goddard, A. K. Morrison, A. Rosati, et al. (2015) *Impacts on ocean heat from transient mesoscale eddies in a hierarchy of climate models* Journal of Climate **28**: 952–977

[Gue16] J.-L. Guermond, B. Popov (2016) *Invariant domains and first-order continuous finite element approximation for hyperbolic systems* SIAM Journal on Numerical Analysis **54**: 2466–2489

[Gue18] J.-L. Guermond, M. Nazarov, B. Popov, I. Tomas (2018) *Second-order invariant domain preserving approximation of the Euler equations using convex limiting* SIAM Journal on Scientific Computing **40**: A3211–A3239

[Gui21] A. P. Guillaumin, L. Zanna (2021) *Stochastic-deep learning parameterization of ocean momentum forcing* Journal of Advances in Modeling Earth Systems **13**: e2021MS002534

[Gul17] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. C. Courville (2017) *Improved Training of Wasserstein GANs* in I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, R. Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA* 5767–5777 URL `https://proceedings.neurips.cc/paper/2017/hash/892c3b1c6dccd52936e27cbd0ff683d6-Abstract.html`

[Haj19] H. Hajduk, D. Kuzmin, V. Aizinger (2019) *New directional vector limiters for discontinuous Galerkin methods* Journal of Computational Physics **384**: 308–325

[Haj22a] H. Hajduk (2022) *Algebraically constrained finite element methods for hyperbolic problems with applications in geophysics and gas dynamics* Ph.D. thesis Dortmund, Technische Universität

[Haj22b] H. Hajduk, D. Kuzmin (2022) *Bound-preserving and entropy-stable algebraic flux correction schemes for the shallow water equations with topography* ArXiv preprint **abs/2207.07261** URL `https://arxiv.org/abs/2207.07261`

[Hin12] G. Hinton, N. Srivastava, K. Swersky (2012) *Neural networks for machine learning lecture 6a overview of mini-batch gradient descent* Cited on **14**: 2

[Hor89] K. Hornik, M. Stinchcombe, H. White (1989) *Multilayer feedforward networks are universal approximators* Neural networks **2**: 359–366

[Kin14] D. P. Kingma, J. Ba (2014) *Adam: A method for stochastic optimization* arXiv preprint arXiv:1412.6980

[Kra13] V. M. Krasnopolsky, M. S. Fox-Rabinovitz, A. A. Belochitski (2013) *Using ensemble of neural networks to learn stochastic convection parameterizations for climate and numerical weather prediction models from*

data simulated by a cloud resolving model Advances in Artificial Neural Systems **2013**: 5–5

[Kuz20] D. Kuzmin (2020) *Monolithic convex limiting for continuous finite element discretizations of hyperbolic conservation laws* Computer Methods in Applied Mechanics and Engineering **361**: 112804

[Kuz22] D. Kuzmin, M. Quezada de Luna, D. I. Ketcheson, J. Grüll (2022) *Bound-preserving flux limiting for high-order explicit Runge–Kutta time discretizations of hyperbolic conservation laws* Journal of Scientific Computing **91**: 21

[LeV02] R. J. LeVeque (2002) *Finite volume methods for hyperbolic problems* vol. 31 Cambridge university press

[Löh87] R. Löhner, K. Morgan, J. Peraire, M. Vahdati (1987) *Finite element flux-corrected transport (FEM–FCT) for the euler and Navier–Stokes equations* International Journal for Numerical Methods in Fluids **7**: 1093–1109

[Loh16] C. Lohmann, D. Kuzmin (2016) *Synchronized flux limiting for gas dynamics variables* Journal of Computational Physics **326**: 973–990

[Maa13] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. (2013) *Rectifier nonlinearities improve neural network acoustic models* in *Proc. icml* vol. 30 3 Atlanta, GA

[Mit97] T. M. Mitchell (1997) *Machine learning* McGraw-Hill

[Nie15] M. A. Nielsen (2015) *Neural networks and deep learning* vol. 25 Determination press San Francisco, CA, USA

[O'G18] P. A. O'Gorman, J. G. Dwyer (2018) *Using machine learning to parameterize moist convection: Potential for modeling of climate, climate change, and extreme events* Journal of Advances in Modeling Earth Systems **10**: 2548–2563

[Pas19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala (2019) *PyTorch: An Imperative Style, High-Performance Deep Learning Library* in H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, R. Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver,*

BC, Canada 8024–8035 URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[Paz21] W. Pazner (2021) *Sparse invariant domain preserving discontinuous Galerkin methods with subcell convex limiting* Computer Methods in Applied Mechanics and Engineering **382**: 113876

[Ped11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. (2011) *Scikit-learn: Machine learning in Python* the Journal of machine Learning research **12**: 2825–2830

[Pop00] S. B. Pope (2000) *Turbulent flows* Cambridge university press

[Qua22] A. Quarteroni (2022) *Algorithms for a New World* Springer

[Ras18] S. Rasp, M. S. Pritchard, P. Gentine (2018) *Deep learning to represent subgrid processes in climate models* Proceedings of the National Academy of Sciences **115**: 9684–9689

[Rum86] D. E. Rumelhart, G. E. Hinton, R. J. Williams (1986) *Learning representations by back-propagating errors* nature **323**: 533–536

[Sch17] T. Schneider, J. Teixeira, C. S. Bretherton, F. Brient, K. G. Pressel, C. Schär, A. P. Siebesma (2017) *Climate goals and computing the future of clouds* Nature Climate Change **7**: 3–5

[Sch23] A. Schwarzmann (2023) *clf1: A collection of finite element and finite volume solver for one-dimensional conservation law problems* URL https://github.com/A-dot-S-dot/cls1

[Shu88] C.-W. Shu, S. Osher (1988) *Efficient implementation of essentially non-oscillatory shock-capturing schemes* Journal of computational physics **77**: 439–471

[Ste13] B. Stevens, S. Bony (2013) *What are climate models missing?* science **340**: 1053–1054

[Sub21] A. Subel, A. Chattopadhyay, Y. Guan, P. Hassanzadeh (2021) *Data-driven subgrid-scale modeling of forced Burgers turbulence using deep learning with generalization to higher Reynolds numbers via transfer learning* Physics of Fluids **33**

[SV71] A. d. Saint-Venant, et al. (1871) *Theorie du mouvement non permanent des eaux, avec application aux crues des rivieres et a l'introduction de marees dans leurs lits* Comptes rendus des seances de l'Academie des Sciences **36**: 174–154

[Tie17] M. Tietz, T. J. Fan, D. Nouri, B. Bossan, skorch Developers (2017) *skorch: A scikit-learn compatible neural network library that wraps PyTorch* URL `https://skorch.readthedocs.io/en/stable/`

[Van11] V. Vanhoucke, A. Senior, M. Z. Mao (2011) *Improving the speed of neural networks on CPUs*

[Vre94] C. B. Vreugdenhil (1994) *Numerical methods for shallow-water flow* vol. 13 Springer Science & Business Media

[Wil07] E. M. Wilcox, L. J. Donner (2007) *The frequency of extreme rain events in satellite rain-rate estimates and an atmospheric general circulation model* Journal of Climate **20**: 53–69

[Yuv21] J. Yuval, P. A. O'Gorman, C. N. Hill (2021) *Use of neural networks for stable, accurate and physically consistent parameterization of subgrid atmospheric processes with good performance at reduced precision* Geophysical Research Letters **48**: e2020GL091363

[Zal79] S. T. Zalesak (1979) *Fully multidimensional flux-corrected transport algorithms for fluids* Journal of computational physics **31**: 335–362

[Zan20] L. Zanna, T. Bolton (2020) *Data-driven equation discovery of ocean mesoscale closures* Geophysical Research Letters **47**: e2020GL088376