Lettuce: PyTorch-Based Lattice Boltzmann Framework

Mario Christopher Bedrunka[1,2], Dominik Wilde[1,2], Martin Kliemank[2], Dirk Reith[2,3], Holger Foysi[1], and Andreas Krämer[4(✉)]

[1] Department of Mechanical Engineering, University of Siegen, Paul-Bonatz-Straße 9-11, 57076 Siegen-Weidenau, Germany
[2] Institute of Technology, Resource and Energy-efficient Engineering (TREE), Bonn-Rhein-Sieg University of Applied Sciences, Grantham-Allee 20, 53757 Sankt Augustin, Germany
[3] Fraunhofer Institute for Algorithms and Scientific Computing (SCAI), Schloss Birlinghoven, 53754 Sankt Augustin, Germany
[4] Department of Mathematics and Computer Science, Freie Universität Berlin, Arnimallee 6, 14195 Berlin, Germany
andreas.kraemer@fu-berlin.de

Abstract. The lattice Boltzmann method (LBM) is an efficient simulation technique for computational fluid mechanics and beyond. It is based on a simple stream-and-collide algorithm on Cartesian grids, which is easily compatible with modern machine learning architectures. While it is becoming increasingly clear that deep learning can provide a decisive stimulus for classical simulation techniques, recent studies have not addressed possible connections between machine learning and LBM. Here, we introduce *Lettuce*, a *PyTorch*-based LBM code with a threefold aim. *Lettuce* enables GPU accelerated calculations with minimal source code, facilitates rapid prototyping of LBM models, and enables integrating LBM simulations with *PyTorch*'s deep learning and automatic differentiation facility. As a proof of concept for combining machine learning with the LBM, a neural collision model is developed, trained on a doubly periodic shear layer and then transferred to a different flow, a decaying turbulence. We also exemplify the added benefit of *PyTorch*'s automatic differentiation framework in flow control and optimization. To this end, the spectrum of a forced isotropic turbulence is maintained without further constraining the velocity field. The source code is freely available from https://github.com/lettucecfd/lettuce.

Keywords: Lattice Boltzmann method · Pytorch · Machine learning · Neural networks · Automatic differentiation · Computational fluid dynamics · Flow control

1 Introduction

Innovations are more important than ever in the face of global challenges such as climate change and pandemics. Bridging the gap from basic understanding to

© Springer Nature Switzerland AG 2021
H. Jagode et al. (Eds.): ISC High Performance 2021 Workshops, LNCS 12761, pp. 40–55, 2021.
https://doi.org/10.1007/978-3-030-90539-2_3

technological solutions often requires physical models in some stages of development. Such models can predict aspects of global warming or the spread of viruses based on fluid dynamics, for example [8,9,32]. However, the models' equations, usually in the form of partial differential equations (PDEs), require efficient solution approaches due to their complexity. By using numerical methods, computers solve these PDEs, which are discretized in space and time. Even though great care is taken, numerical errors inevitably occur during discretization, where the truncation error usually depends on grid and time step size. Hence, large resolutions are necessary if high accuracy is required, especially if modeling is reduced as much as possible, e.g., during direct numerical simulations (DNS). When DNS are computationally intractable, alternatives like the well-known Reynolds averaged Navier-Stokes equations (RANS) or large-eddy simulation (LES) are routinely used. These approaches require expressing unresolved or unclosed terms through known quantities [36,42]. Parameters involved in these modeling approaches are often not optimally determined or change for different flows. For this purpose, machine learning (ML) in fluid dynamics is a rapidly evolving research field that enables new impulses to tackle such problems and provide interesting new approaches to the modeling problem. The benefit of ML approaches in computational fluid dynamics (CFD) simulations has been demonstrated in various recent studies [10,19,39,40]. A successful use of machine learning in combination with well-known Navier-Stokes solvers was recently shown by Kochkov et al. [19]. They introduced learned forcing terms into PDE solvers, thereby reaching the same accuracy as a classical solver at 8-10x finer resolution.

When integrating machine learning into numerical algorithms, it can be beneficial to resort to simulation methods whose mathematical structure is easily compatible with neural networks. The present work shows that a highly suitable CFD approach for this purpose is the lattice Boltzmann method [21,27] (LBM), which is particularly competitive in the fields of transient, turbulent, or multiphase fluid dynamics. The LBM is a second order accurate simulation method that exhibits similar performance as classical finite difference schemes [41]. In contrast to classical solvers, it involves a linear streaming of particle distribution functions on a regular grid and a local collision step. Despite its successful application to many fluid problems, recent studies have only scarcely addressed possible combinations of ML and LBM. As a prototypical approach, Hennigh [15] has demonstrated the potential of ML-driven flow prediction based on LBM. He compressed flow fields onto coarser grids using convolutional autoencoders and learned the propagation of the latent representations, which can then be decoded back onto the fine grid. This approach, however, has limited transferability as it is primarily data-informed and does not encode the underlying physics. Furthermore, Rüttgers et al. [33] have applied deep learning methods to the lattice Boltzmann method to predict the sound pressure level caused by objects. They introduced an encoder-decoder convolutional neural network and discussed various learning parameters to accurately forecast the acoustic fields. To the best of our knowledge, no further ML-enhanced LBM methods were proposed.

Although the mathematics and physics behind the LBM are ambitious, the implementation is relatively simple. This allows beginners to quickly set up simple test cases in one or two dimensions with popular scripting languages like Matlab or Python. However, when switching to three dimensions, the algorithmic and computational complexity of simulations severely limits such prototypical implementations. More efficient simulations in compiled languages usually rely on optimized software packages and require initial training to simulate complex flows. This is particularly true for GPU-accelerated codes, which enhance performance [26,29] but defy fast implementation.

Both the lack of machine learning studies in the context of LBM and the code complexity of 3D implementations motivate an LBM framework that allows ease of use, despite extensive built-in functionality for machine learning algorithms. For this purpose, the software package *Lettuce* has been developed based on the open-source machine learning framework *PyTorch* [31]. *PyTorch* implements optimized numerical operations on CPUs and GPUs, which can easily be accessed via Python instructions. Internally, those operations are vectorized using efficient backends such as BLAS/LAPACK and highly optimized CUDA code. By resorting to those efficient *PyTorch* core routines, the LBM code remains maintainable and lean, which will be demonstrated throughout this article. Furthermore, the *Lettuce* framework can seamlessly integrate *PyTorch*'s machine learning modules into the fluid dynamics solver and thereby provide a substantial contribution to future work in this area.

Lettuce is meant to complement existing optimized codes like the well-known LBM frameworks OpenLB [17], Palabos [25], waLBerla [1,14], and others [28,30,38]. It intends to bridge the gap between scripting language codes for local machines and highly optimized codes based on compiled programming languages that run efficiently on computer clusters, too. With an off-the-shelf GPU, *Lettuce* can simulate fairly complex three-dimensional problems even on a local workstation with 24 GB of GPU memory. Independent of machine learning research, this also enables rapid prototyping of general methodological extensions for the lattice Boltzmann method.

This short paper is structured as follows. Section 2 presents an overview of the software functionalities and briefly describes the LBM and its implementation. Section 3 shows simple examples of coupling the LBM with machine learning, demonstrates the use of *PyTorch*'s automatic differentiation capabilities in CFD simulations, and provides a computational benchmark. Section 4 presents a summary and conclusions. Scripts for all simulations in this paper are accessible on https://github.com/lettucecfd/lettuce-paper.

2 Software Description

2.1 Software Functionalities

The lattice Boltzmann method (LBM) is based on the kinetic theory of gases, concretely a discretized version of the BGK-Boltzmann equation. It evolves a

discrete particle distribution function $f_i(\mathbf{x}, t)$ according to the lattice Boltzmann equation

$$f_i(\mathbf{x} + \mathbf{c}_i\delta_t, t + \delta_t) = f_i(\mathbf{x}, t) + \Omega_i(\boldsymbol{f}(\mathbf{x}, t)), \tag{1}$$

where δ_t and $\mathbf{c}_i$ denote the time step and discrete particle velocities, respectively. At its core, the LBM involves two operations. First, the so-called streaming step shifts the particle distributions $f_i(\mathbf{x}, t)$ to the neighboring nodes along the trajectories $\mathbf{c}_i$. Second, the collision step introduces interactions between the particles on each node. Among the various collision models available in the literature, $\Omega(\boldsymbol{f})$ is selected based on considerations such as asymptotic behavior, accuracy, memory usage and stability. The most commonly used collision operator is the Bhatnagar-Gross-Krook model:

$$\Omega_i(\boldsymbol{f}) = -\frac{f_i - f_i^{\mathrm{eq}}}{\tau} \tag{2}$$

This operator describes the relaxation of the particle distribution function towards an equilibrium distribution influenced by a single relaxation parameter τ. The equilibrium distribution is given by

$$f_i^{\mathrm{eq}}(\mathbf{x}, t) = w_i\rho\left(1 + \frac{\mathbf{u}\cdot\mathbf{c}_i}{c_s^2} + \frac{(\mathbf{u}\cdot\mathbf{c}_i)^2}{2c_s^4} - \frac{\mathbf{u}\cdot\mathbf{u}}{2c_s^2}\right), \tag{3}$$

where w_i and c_s are the lattice weights and speed of sound, respectively. The density ρ and fluid velocity $\mathbf{u}$ are obtained as

$$\rho(\mathbf{x}, t) = \sum_i f_i(\mathbf{x}, t) \qquad \text{and} \qquad \rho\mathbf{u}(\mathbf{x}, t) = \sum_i \mathbf{c}_i f_i(\mathbf{x}, t). \tag{4}$$

Lettuce is equipped with a variety of frequently used collision models, such as the Bhatnagar-Grook-Krook (BGK) model [2], multi-relaxation time collision models [7,22], the two-relaxation time model [13], the regularized model [23] and entropic two-relaxation time models by Karlin, Bösch and Chikatamarla (KBC) [18]. For the latter, implementations are rare in open software packages. Many of these collision models are implemented in a stencil- and dimension-independent manner.

2.2 Code Example

After *Lettuce* is installed, the user can run simulations with minimal code. The following example demonstrates a lean executable Python script that simulates a three-dimensional Taylor-Green vortex (TGV3D), one of several flows provided in the library. The *Lettuce* library contains various boundary conditions, forcing and initialization schemes, thus covering a wide range of setups. After importing *Lettuce*, the stencil and the hardware are selected. Then, the flow, collision model, and streaming step are chosen and run through the `Simulation` class.

```
import lettuce as lt
lattice = lt.Lattice(stencil=lt.D3Q27, device='cuda') #or 'cpu'
flow = lt.TaylorGreenVortex3D(
    resolution=256,
    reynolds_number=1600,
    mach_number=0.05,
    lattice=lattice)
collision = lt.BGKCollision(
    lattice=lattice,
    tau=flow.units.relaxation_parameter_lu)
streaming = lt.StandardStreaming(lattice)
simulation = lt.Simulation(
    flow=flow,
    lattice=lattice,
    collision=collision,
    streaming=streaming)
simulation.step(num_steps=10000)
```

Lettuce provides various observables that can be reported during the simulation (e.g. kinetic energy, enstrophy, energy spectrum). These observables can be added easily to the `Simulation` class and exported for further analysis as follows:

```
energy = lt.IncompressibleKineticEnergy(lattice, flow)
simulation.reporters.append(
    lt.ObservableReporter(energy, interval=10,))
simulation.reporters.append(
    lt.VTKReporter(
        lattice,
        flow,
        interval=10,
        filename_base="./output"))
```

Besides, *Lettuce* comes with a VTK-reporter based on the PyEVTK library [16]. This reporter exports velocity components and pressure, which both can then be visualized by third-party software. An example is given in Fig. 1, which shows the isosurfaces of the three-dimensional Taylor-Green vortex simulation from the code snippet above.

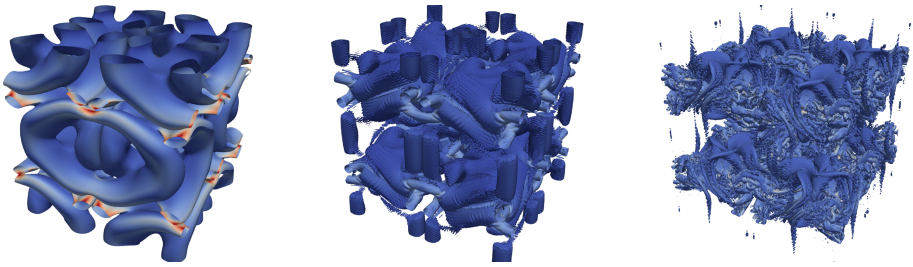

Fig. 1. Q-criterion isosurfaces of a three-dimensional Taylor-Green Vortex at time step t = 5000, 7000 and 10000 colored by streamwise velocity. The Reynolds number and grid resolution were 1600 and 256^3, respectively. (Color figure online)

Figure 2 shows the energy dissipation that is obtained from the kinetic energy k by calculating $-dk/dt$ ($=\nu\langle\epsilon\rangle$ in isotropic turbulence) using finite differences. That way it includes numerical dissipation effects and spurious contributions from LBM, too. The data is compared to the reference taken from Brachet [3], who used a spectral code with a resolution of 256^3 grid points. The dissipation, $-dk/dt$, shows excellent agreement with the reference data up to a Reynolds number of 1600. For Reynolds numbers of $Re = 3000$ and higher the maximum dissipation rate deviates slightly due to under-resolution.

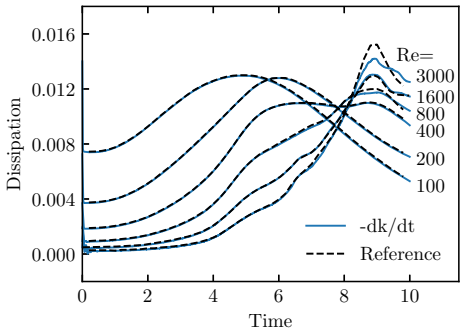

Fig. 2. Energy dissipation rate $\epsilon(t) = -dk/dt$ of a three-dimensional Taylor-Green-Vortex using the BGK collision model. Reference is taken from Brachet [3].

3 Advanced Functionalities

3.1 Machine Learning

The collision model constitutes the core of the LBM as it determines the macroscopic system of PDEs and thereby encodes the solver's physics. It has long been known that the choice of collision model for a given PDE is ambiguous, which is related to the discrepancy between the number of degrees of freedom (discrete

distribution functions) and macroscopic variables of interest. For example, the standard D2Q9 stencil uses nine degrees of freedom per lattice node to encode six physically relevant moments (density, momentum, and stress tensor). The remaining degrees of freedom (represented by higher-order moments or cumulants) are usually propagated in a way that offers certain numerical advantages such as improved stability [18,20,23] or accuracy [12].

In the following, we want to exploit this ambiguity to define neural collision operators as a more accurate alternative to classical collision models. For this purpose, we define a collision model that relaxes the moments m_i towards their respective equilibria m_i^{eq} by individual relaxation rates $\mathbf{S} = \mathrm{diag}\,(\tau_0, \tau_1, ..., \tau_{Q-1})$, where Q is the number of discrete velocities per grid point. A transformation matrix $\mathbf{M}$ (according to Dellar [7]) maps the distribution function $\boldsymbol{f}$ to the moments $\mathbf{m} = \mathbf{M}\boldsymbol{f} = (\rho, \rho\mathbf{u}, \mathbf{\Pi}, \mathscr{N}, \boldsymbol{\mathscr{J}})^T$, where ρ is the density, $\mathbf{u} = (u, v)$ is the fluid velocity, $\mathbf{\Pi}$ is the momentum flux, and $\mathscr{N}$ and $\boldsymbol{\mathscr{J}} = (\mathscr{J}_0, \mathscr{J}_1)$ are non-hydrodynamic moments. These moments are relaxed towards the moment equilibrium $\mathbf{m}^{\mathrm{eq}} = \mathbf{M}\boldsymbol{f}^{\mathrm{eq}}$ with the relaxation rates given by $\mathbf{S}$:

$$\Omega(\boldsymbol{f}) = -\boldsymbol{M}^{-1}\boldsymbol{S}^{-1}(\boldsymbol{M}\boldsymbol{f} - \boldsymbol{m}^{\mathrm{eq}}). \tag{5}$$

The relaxation rates for the conserved moments ρ and $\rho\boldsymbol{u}$ have no effect and are thus set to unity. Since the shear relaxation rates are related to the kinematic viscosity ν, they are set to $\tau_n = \nu c_s^{-2}\delta_t^{-1} + 0.5$, $n = 3, 4, 5$, which recovers the Navier-Stokes equations in the weakly compressible regime. A neural network provides the relaxation rates τ_n, $n = 6, 7, 8$, for the higher moments. For this purpose, a shallow network with one hidden layer and 530 parameters is introduced to keep the computational cost feasible. The network determines the higher-order relaxation rates based on local moments and is optimized to reproduce a finer-resolved reference simulation. Moreover, we want to ensure that the collision operator is stable. For this purpose, an exponential function is applied to the output $\tilde{\tau}_n$ of the neural network: $\tau_n = \exp(\tilde{\tau}_n) + 0.5$, $n = 6, 7, 8$. This operation renders the output larger than 0.5, which prevents excessive over-relaxation. The relaxation parameters are not upper bounded as $\tau_n \to \infty$ yields the identity and is usually uncritical in terms of stability.

Training data was generated by simulating a doubly periodic shear layer at a Reynolds number of 5000 on a domain of size 128^2 using the BGK model [4]. The shear layer provides both large gradients and smooth areas which need to be detected. Most relevant engineering flows have features that are locally present in shear layer turbulence, such that good performance of a model in this setup will likely transfer to other flows. Instead, training with isotropic flows only would likely hamper transferability.

Depending on the information contained in the local moments, the network should adjust the local relaxation parameters. The training procedure optimizes the network parameters $\boldsymbol{\vartheta}$ to minimize the discrepancy between a low-resolution and a high-resolution simulation. Therefore, the discrete distributions from the training trajectory are mapped onto a coarser grid with 64^2 grid points. Batches of short simulations with the neural collision model are started from each snapshot. After 100 simulation steps, the flow fields are compared with the finer-resolved reference simulation based on energy E, vorticity ω, and velocity u,

which are all computed on the coarse grid. The mean-squared error (MSE) of these quantities from the reference are minimized using the Adam optimizer and the loss function

$$\begin{aligned} L\left(t;\boldsymbol{\vartheta}\right) = w_u \sum_i^N \mathrm{MSE}\left(u(\mathbf{x}_i,t;\boldsymbol{\vartheta}),\tilde{u}(\mathbf{x}_i,t)\right) \\ + w_\omega \sum_i^N \mathrm{MSE}\left(\omega(\mathbf{x}_i,t;\boldsymbol{\vartheta}),\tilde{\omega}(\mathbf{x}_i,t)\right) \\ + w_E\ \mathrm{MSE}\left(E(t;\boldsymbol{\vartheta}),\tilde{E}(t)\right), \end{aligned} \tag{6}$$

where N is the number of grid points. The weights are hyperparameters that were selected as $w_u := 0.6$, $w_\omega := w_E := 0.2$. This choice emphasizes the optimization of dissipation effects, which are critical in under-resolved turbulence. Such flows exhibit large gradients that occur intermittently, leading to locally under-resolved spatial structures. Therefore, the model has to strike a balance between retaining the physical structures on small scales and adding numerical dissipation for stabilization.

The fluid velocity is the most natural target as it directly measures numerical errors. The kinetic energy tracks the dissipation globally but does not resolve the spatial heterogeneity. In contrast, including the vorticity as a target stresses the finest resolved structures; the enstrophy, i.e., the integral over the vorticity magnitude, measures the viscous shear stresses that induce dissipation. In homogeneous turbulence, it peaks at high wave numbers around the Kolmogorov scale. Consequently, optimizing the loss function (6) deliberately targets the dissipation occurring on small scales. A detailed hyperparameter optimization, the inclusion of other target properties, and the incorporation of multiple flows with different boundary conditions in the training set will likely further improve the model. These next steps as well as a systematic study of transferability are beyond the scope of this proof-of-concept and will be left for future work.

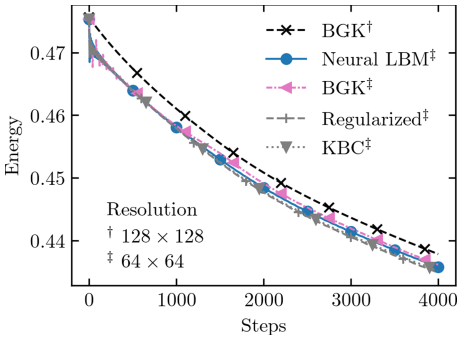

Fig. 3. Comparison of the kinetic energy evolution of various collision models for the doubly periodic shear layer at $Re = 5000$.

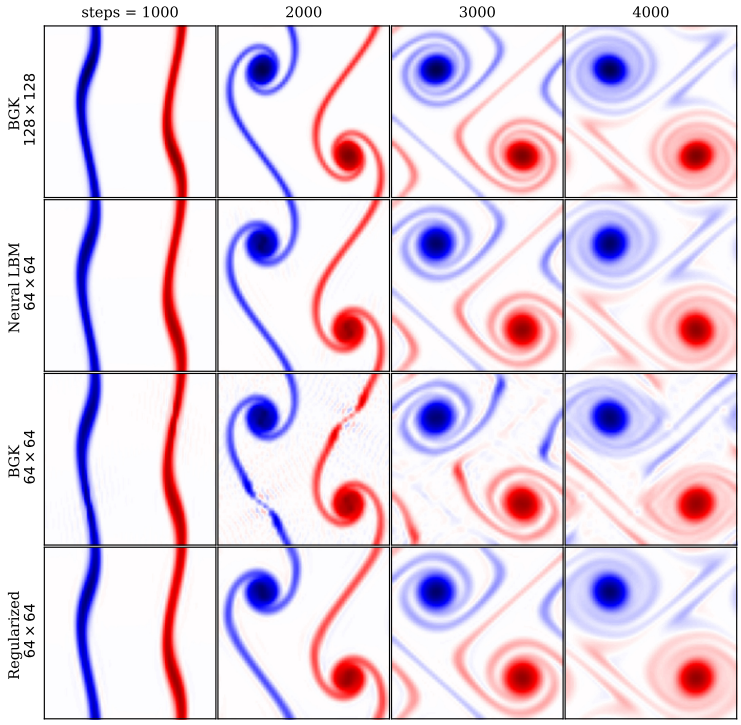

Fig. 4. Evolution of vorticity fields for a doubly periodic shear layer flow for Reynolds number 5000 using several collision models [4].

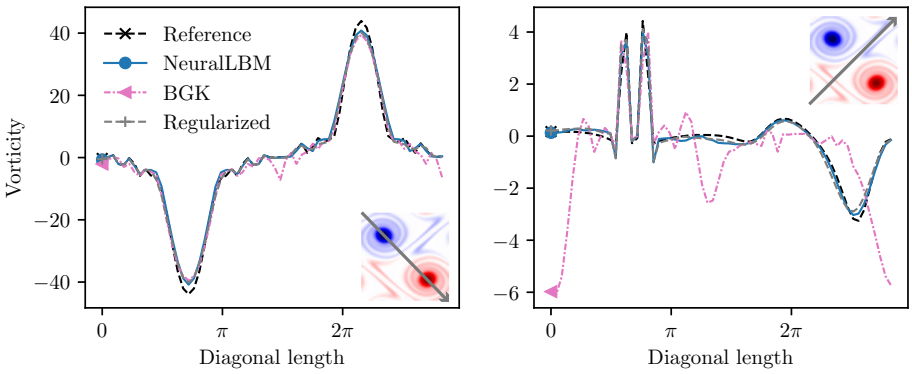

Fig. 5. Vorticity along a diagonal line for a doubly periodic shear layer flow after 4000 steps for Reynolds number 5000.

Turning to results, Fig. 3 compares the evolution of kinetic energy for the doubly periodic shear layer. Several collision models were used for this assessment. In the beginning, the energy of the lower-resolved simulations dropped due to under-resolution. Then, all collision models produced similar energies. However, the vorticity fields depicted in Fig. 4 clearly show that the simulation using the BGK operator can no longer capture the vortices from the finer-resolved reference simulation. In contrast, the neural collision model accurately reproduces these structures, as shown in more detail in Fig. 5 by the vorticity along the diagonals. The improvement compared to the BGK operator becomes clear while still providing less dissipation than the regularized model.

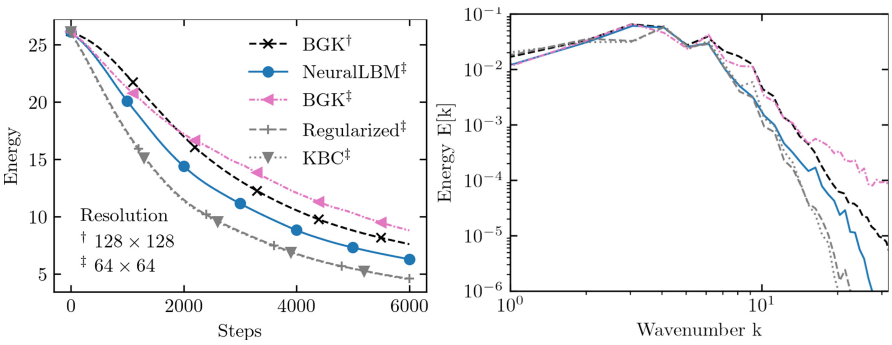

Fig. 6. Enstrophy evolution (*left*) and energy spectrum (*right*) of an isotropic decaying turbulence at Reynolds number 30000. Colors and line types are equal for both plots. (Color figure online)

The crucial question is whether the optimized network is transferable to other flows. Figure 6 shows the vorticity evolution and energy spectrum for an isotropic decaying turbulence simulation at a Reynolds number of 30000 [37]. Although trained on a different flow, the neural collision model reproduced the vortex field far better, while other collision models were either unstable or overly dissipative, as shown in Fig. 7. The BGK model was not able to handle the high Reynolds number and introduced unphysical small-scale oscillations. These numerical artefacts are visible in the energy spectrum, revealing a lot of unphysical energy accumulated at high wavenumbers. By contrast, the KBC and regularized collision models are more dissipative at larger wavenumbers, resulting in much faster energy and enstrophy decay. In comparison to these baseline models, the ML-enhanced simulation produced the best match with the reference simulation. This example demonstrates generalization capabilities and the potential benefit of using collision models based on neural networks.

A promising future direction of research is to target the current limitations of the LBM, including high Mach number compressible flows. These flows require higher-order moments so that current compressible LBMs usually resort to larger stencils [6,11,24,43], off-lattice streaming [5,43,44], or non-local collision models

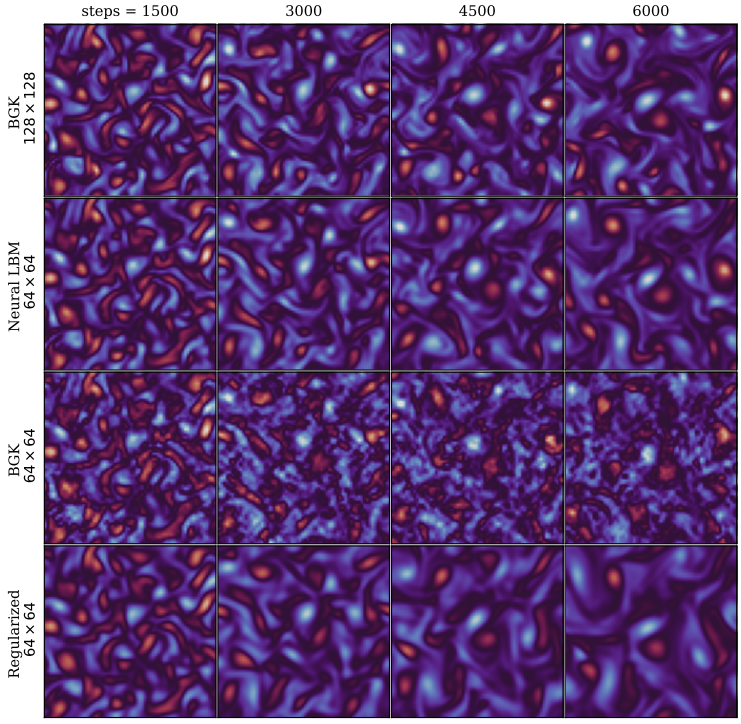

Fig. 7. Evolution of vorticity fields for a isotropic decaying turbulence flow for Reynolds number 30000 using several collision models.

[34,35] that introduce additional numerical approximations. In this application, neural collisions could help reduce numerical errors and advance the state of the art.

3.2 Flow Control Through Automatic Differentiation

The availability of an automatic differentiation framework within a CFD simulation engine has additional advantages (besides machine learning). *PyTorch* provides analytic derivatives for all numerical operations, which is, for example, useful in flow control and optimization.

As a demonstration of these capabilities, forced isotropic turbulence is simulated, i.e., the energy spectrum is maintained throughout the simulation using a forcing term as detailed below. A cost functional R is introduced as the relative deviation of the instantaneous spectrum $\sigma(\mathbf{u})$ from the target spectrum σ_0 with a cutoff at $c := 2 \cdot 10^{-5}$. R is defined as

$$R(\boldsymbol{u}) = \| \ln \max(\sigma(\boldsymbol{u}), c) - \ln \max(\sigma_0, c) \|_2^2,$$

where the logarithm and maximum are taken elementwise.

To incorporate this restraint into the simulation, the equilibrium distribution $f^{\mathrm{eq}}(\rho, \mathbf{u} + \Delta\mathbf{u})$ is expanded around a velocity that is locally shifted by a forcing term $\Delta\mathbf{u} := -\kappa \cdot \nabla_{\mathbf{u}} R$ with a force constant $\kappa = 5 \cdot 10^{-5}$. Computing the gradient requires differentiating through various numeric operations, including a Fast Fourier Transform, which is easily done within *PyTorch* due to the automatic differentiation facility.

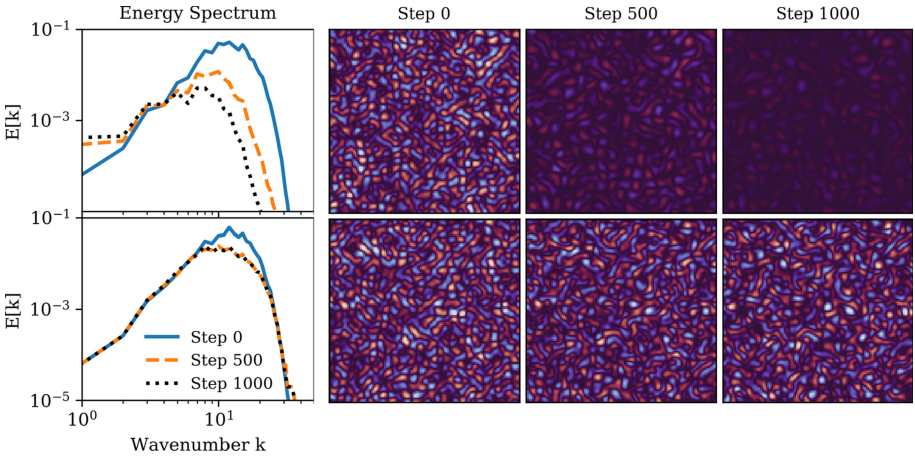

Fig. 8. Energy spectrum $E[k]$ (left column) and evolution of vorticity fields in isotropic turbulence. Upper row: free simulation; lower row: restrained simulation.

Figure 8 shows the vorticity fields and energy spectrum for a simulation at a resolution of 128^2 grid points with $Re = 2000$ and $Ma = 0.1$. While the unrestrained simulation decays, the restrained simulation maintains the spectrum after an initial adjustment phase took place, starting from the artificial initialization field. This example shows that complicated forces are easily incorporated into simulations through automatic differentiation. This feature can be useful in many other applications.

3.3 Benchmark

Lettuce attains a satisfactory performance due to the GPU operations provided by *PyTorch*. The optimized backend code enables fast CUDA-driven simulations on both cluster GPUs and even standard GPUs for workstations. We evaluated the performance of *Lettuce* by simulating a Taylor-Green vortex in both 2D (D2Q9) and 3D (D3Q19). The results are compared for both an NVIDIA Tesla V100 and an NVIDIA RTX2070S in single and double precision. Figure 9 compares the performance in MLUPS (Million Lattice Updates Per Second) for different resolutions.

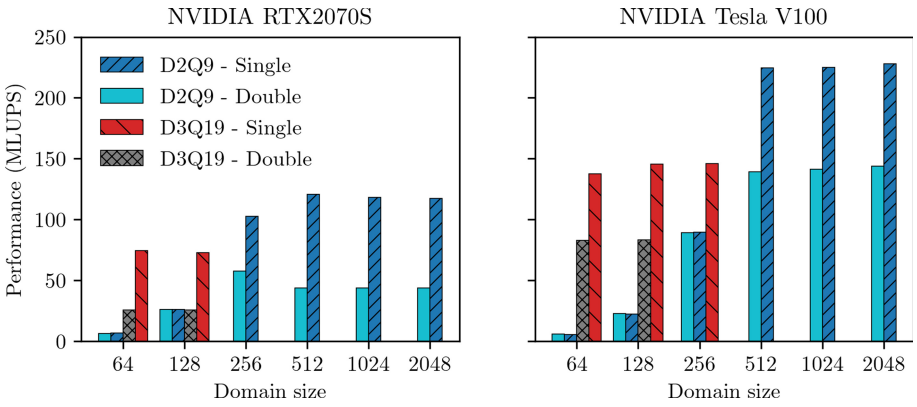

Fig. 9. Performance of Lettuce for simulating a Taylor-Green vortex.

With increasing domain size $64^2; 128^2; 256^2; 512^2$, the simulation speed increases to 140 MLUPS in two dimensions and 83 MLUPS in three dimensions using an NVIDIA Tesla V100 in double precision. The simulation speed increases by over 60% to 75% when calculations are performed in single precision. Using an off-the-shelve NVIDIA RTX2070S, the computational efficiency was lower, as expected. Performance peaked at 58 MLUPS using a domain size of 256^2 in two dimensions. For higher resolutions, the performance decreased to 44 MLUPS. By using single precision, the performance can be increased by 180% for higher resolutions. This comparison shows that even on a commercially available consumer-grade GPU high-performance simulations can be performed in an acceptable time. Further speedups will be obtained by implementing custom C++ and CUDA extensions, for which *PyTorch* offers a modular and well-documented interface. Such extensions as well as a distributed multi-GPU implementation through `torch.distributed` are natural enhancements that are currently in progress.

4 Conclusion

We have introduced *Lettuce*, a *PyTorch*-based lattice Boltzmann code that bridges lattice Boltzmann simulations and machine learning. We have demonstrated how simulations can be set up and run with minimal use of source code. This eases code development significantly and flattens the learning curve for beginners. Scientists and engineers can run GPU-accelerated three-dimensional simulations even on local workstations, which benefits rapid prototyping of lattice Boltzmann models. Besides machine learning routines, the framework supports automatic differentiation for flow control and optimization. As an example, a forced isotropic turbulence simulation was run with a maintained energy spectrum. Furthermore, we have defined a neural collision model to demonstrate the benefits of incorporating neural networks into the lattice Boltzmann method.

The presented results indicate that neural collision models can outperform traditional collision operators and reduce numerical errors, which motivates further research in this direction.

References

1. Bauer, M., et al.: waLBerla: a block-structured high-performance framework for multiphysics simulations. Comput. Math. with Appl. **81**, 478–501 (2021)
2. Bhatnagar, P.L., Gross, E.P., Krook, M.: A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems. Phys. Rev. **94**(3), 511–525 (1954)
3. Brachet, M.E., Meiron, D.I., Orszag, S.A., Nickel, B., Morf, R.H., Frisch, U.: Small-scale structure of the Taylor-green vortex. J. Fluid Mech. **130**, 411–452 (1983)
4. Brown, D.L.: Performance of under-resolved two-dimensional incompressible flow simulations. J. Comput. Phys. **122**(1), 165–183 (1995)
5. Chen, T., Wen, X., Wang, L.P., Guo, Z., Wang, J., Chen, S.: Simulation of three-dimensional compressible decaying isotropic turbulence using a redesigned discrete unified gas kinetic scheme. Phys. Fluids **32**(12), 125104 (2020)
6. Coreixas, C., Latt, J.: Compressible lattice Boltzmann methods with adaptive velocity stencils: an interpolation-free formulation. Phys. Fluids **32**(11), 116102 (2020)
7. Dellar, P.J.: Incompressible limits of lattice Boltzmann equations using multiple relaxation times. J. Comput. Phys. **190**(2), 351–370 (2003)
8. Diwan, S.S., Ravichandran, S., Govindarajan, R., Narasimha, R.: Understanding transmission dynamics of Covid-19-type infections by direct numerical simulations of cough/sneeze flows. Trans. Indian Natl. Acad. Eng. **5**, 255–261 (2020)
9. Fabregat, A., Gisbert, F., Vernet, A., Dutta, S., Mittal, K., Pallarès, J.: Direct numerical simulation of the turbulent flow generated during a violent expiratory event. Phys. Fluids **33**(3), 035122 (2021)
10. Font, B., Weymouth, G.D., Nguyen, V.T., Tutty, O.R.: Deep learning of the spanwise-averaged Navier-Stokes equations. J. Comput. Phys. **434**, 110199 (2021)
11. Frapolli, N., Chikatamarla, S.S., Karlin, I.V.: Entropic lattice Boltzmann model for compressible flows. Phys. Rev. E Stat. Nonlinear Soft Matter Phys. **92**(6), 061301 (2015)
12. Geier, M., Pasquali, A., Schönherr, M.: Parametrization of the cumulant lattice Boltzmann method for fourth order accurate diffusion. Part I: Derivation and validation. J. Comput. Phys. **348**, 862–888 (2017)
13. Ginzburg, I., Verhaeghe, F., D'Humières, D.: Two-relaxation-time Lattice Boltzmann scheme: about parametrization, velocity, pressure and mixed boundary conditions. Commun. Comput. Phys. **3**(2), 427–478 (2008)
14. Godenschwager, C., Schornbaum, F., Bauer, M., Köstler, H., Rüde, U.: A framework for hybrid parallel flow simulations with a trillion cells in complex geometries. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis - SC 2013, New York, NY, USA, pp. 1–12. ACM Press (2013)
15. Hennigh, O.: Lat-Net: compressing lattice Boltzmann flow simulations using deep neural networks. arXiv preprint arXiv:1705.09036 (2017)
16. Herrera, P.: pyevtk 1.2.0. PyPI (2021). https://pypi.org/project/pyevtk/

17. Heuveline, V., Krause, M.J.: OpenLB: towards an efficient parallel open source library for lattice Boltzmann fluid flow simulations. In: International Workshop on State-of-the-Art in Scientific and Parallel Computing, PARA, vol. 9 (2010)
18. Karlin, I.V., Bösch, F., Chikatamarla, S.: Gibbs' principle for the lattice-kinetic theory of fluid dynamics. Phys. Rev. E Stat. Nonlinear Soft Matter Phys. **90**(3), 1–5 (2014)
19. Kochkov, D., Smith, J.A., Alieva, A., Wang, Q., Brenner, M.P., Hoyer, S.: Machine learning accelerated computational fluid dynamics. Proc. Nat. Acad. Sci. **118**(21), e2101784118 (2021).
20. Krämer, A., Wilde, D., Küllmer, K., Reith, D., Foysi, H.: Pseudoentropic derivation of the regularized lattice Boltzmann method. Phys. Rev. E **100**(2), 023302 (2019)
21. Krüger, T., Kusumaatmaja, H., Kuzmin, A., Shardt, O., Silva, G., Viggen, E.M.: The Lattice Boltzmann Method: Principles and Practice. Springer, Heidelberg (2017)
22. Lallemand, P., Luo, L.S.: Theory of the lattice Boltzmann method: dispersion, dissipation, isotropy, Galilean invariance, and stability. Phys. Rev. E Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top. **61**(6), 6546–6562 (2000)
23. Latt, J., Chopard, B.: Lattice Boltzmann method with regularized pre-collision distribution functions. Math. Comput. Simul. **72**(2–6), 165–168 (2006)
24. Latt, J., Coreixas, C., Beny, J., Parmigiani, A.: Efficient supersonic flow simulations using lattice Boltzmann methods based on numerical equilibria. Philos. Trans. R. Soc. A Math. Phys. Eng. Sci. **378**(2175), 20190559 (2020)
25. Latt, J., et al.: Palabos: parallel lattice Boltzmann solver. Comput. Math. Appl. **81**, 334–350 (2021)
26. Lenz, S., et al.: Towards real-time simulation of turbulent air flow over a resolved urban canopy using the cumulant lattice Boltzmann method on a GPGPU. J. Wind Eng. Ind. Aerodyn. **189**, 151–162 (2019)
27. McNamara, G.R., Zanetti, G.: Use of the Boltzmann equation to simulate lattice-gas automata. Phys. Rev. Lett. **61**(20), 2332–2335 (1988)
28. Mora, P., Morra, G., Yuen, D.A.: A concise python implementation of the lattice Boltzmann method on HPC for geo-fluid flow. Geophys. J. Int. **220**(1), 682–702 (2020)
29. Obrecht, C., Kuznik, F., Tourancheau, B., Roux, J.J.: Scalable lattice Boltzmann solvers for CUDA GPU clusters. Parallel Comput. **39**(6), 259–270 (2013)
30. Pastewka, L., Greiner, A.: HPC with python: an MPI-parallel implementation of the lattice Boltzmann method. In: Proceedings of the 5th bwHPC Symposium (2019)
31. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., D'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019)
32. Porté-Agel, F., Bastankhah, M., Shamsoddin, S.: Wind-turbine and wind-farm flows: a review. Boundary Layer Meteorol. **174**(1), 1–59 (2020)
33. Rüttgers, M., Koh, S.-R., Jitsev, J., Schröder, W., Lintermann, A.: Prediction of acoustic fields using a lattice-Boltzmann method and deep learning. In: Jagode, H., Anzt, H., Juckeland, G., Ltaief, H. (eds.) ISC High Performance 2020. LNCS, vol. 12321, pp. 81–101. Springer, Cham (2020)
34. Saadat, M.H., Hosseini, S.A., Dorschner, B., Karlin, I.V.: Extended lattice Boltzmann model for gas dynamics. Phys. Fluids **33**(4), 046104 (2021)

35. Saadat, M.H., Bösch, F., Karlin, I.V.: Lattice Boltzmann model for compressible flows on standard lattices: variable Prandtl number and adiabatic exponent. Phys. Rev. E **99**(1), 013306 (2019)
36. Sagaut, P.: Large Eddy Simulation for Incompressible Flows. An Introduction. Springer, Heidelberg (2006)
37. Samtaney, R., Pullin, D.I., Kosović, B.: Direct numerical simulation of decaying compressible turbulence and shocklet statistics. Phys. Fluids **13**(5), 1415–1430 (2001)
38. Schmieschek, S., et al.: LB3D: a parallel implementation of the lattice-Boltzmann method for simulation of interacting amphiphilic fluids. Comput. Phys. Commun. **217**, 149–161 (2017)
39. Brunton, S.L., Noack, B.R., Koumoutsakos, P.: Machine learning for fluid mechanics. Annu. Rev. Fluid Mech. **52**, 477–508 (2020)
40. Um, K., Fei, Y.R., Holl, P., Brand, R., Thuerey, N.: Solver-in-the-loop: learning from differentiable physics to interact with iterative PDE-solvers. In: 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada, vol. 1, no. c, pp. 1–37 (2020)
41. Wichmann, K.R., Kronbichler, M., Löhner, R., Wall, W.A.: A runtime based comparison of highly tuned lattice Boltzmann and finite difference solvers. Int. J. High Perform. Comput. Appl. (2021).
42. Wilcox, D.C.: Turbulence Modeling for CFD. DCW Industries, CA (1993)
43. Wilde, D., Krämer, A., Bedrunka, M., Reith, D., Foysi, H.: Cubature rules for weakly and fully compressible off-lattice Boltzmann methods. J. Comput. Sci. **51**, 101355 (2021)
44. Wilde, D., Krämer, A., Reith, D., Foysi, H.: Semi-Lagrangian lattice Boltzmann method for compressible flows. Phys. Rev. E **101**(5), 53306 (2020)