
Lat-Net: Compressing Lattice Boltzmann Flow Simulations using Deep Neural Networks

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 Computational Fluid Dynamics (CFD) is a hugely important subject with applica-
2 tions in almost every engineering field, however, fluid simulations are extremely
3 computationally and memory demanding. Towards this end, we present Lat-Net, a
4 method for compressing both the computation time and memory usage of Lattice
5 Boltzmann flow simulations using deep neural networks. Lat-Net employs convo-
6 lutional autoencoders and residual connections in a fully differentiable scheme to
7 compress the state size of a simulation and learn the dynamics on this compressed
8 form. The result is a computationally and memory efficient neural network that
9 can be iterated and queried to reproduce a fluid simulation. We show that once
10 Lat-Net is trained, it can generalize to large grid sizes and complex geometries
11 while maintaining accuracy. We also show that Lat-Net is a general method for
12 compressing other Lattice Boltzmann based simulations such as Electromagnetism.

13 1 Introduction

14 Computational fluid dynamics (CFD) is a branch of fluid mechanics that deals with numerically
15 solving and analyzing fluid flow problems such as those found in aerodynamics, geology, biology, etc.
16 CFD simulations are known for their high computational requirements, memory usage, and run times.
17 Because of this, there is an ever growing body of work using simulation data to create reduced order
18 models or surrogate models that can be evaluated with significantly less resources. Towards this end,
19 we develop a neural network approach that both compresses the computation time and memory usage
20 of fluid simulations.

21 We investigate fluid simulations that contain complex time dependent turbulence. Simulations of
22 this form are difficult because they require fluid solver to have high resolution and small times steps.
23 Never the less, they frequently occur in nature and are an important area of study. Motivated by need
24 for these simulations and the recent success of neural network based models in related areas (1) (2)
25 (3), we choice this setting to test our model.

26 The most popular approach to modeling fluid flow is with the Navier stokes equation. The solution
27 to this partial differential equation gives the flow velocity field for a given domain. Recently, a new
28 method for simulating fluid flow has emerged named the Lattice Boltzmann Method (LBM). It is
29 derived from the Boltzmann equation and grew out of Lattice Gas Automaton (LGA) in the late 80s
30 (4). The main advantages of the LBM are its ability to run on complex geometries, its scalability to
31 parallel architectures (particularly GPUs) and applicability to complex flows that contain phenomena
32 such as heat transfer and chemical reactions. Our method is centered around this method of simulating
33 flow.

34 Lat-Net works by compressing the state of a simulation while learning the dynamics of the simulation
35 on this compressed form. The model can be broken up into three pieces, an encoder, compression

36 mapping, and decoder. The encoder compresses both the state of the simulation as well as the given
37 boundary conditions. The compression mapping learns the dynamics on the compressed state that
38 correspond to the dynamics in the fluid simulation. The decoder decompresses the compressed state
39 allowing either the whole simulation state or desired pieces to be extracted.

40 We focus the content of this paper on LBM fluid simulations because this is the most popular use of the
41 LBM, however, this method of simulation is known to be able to solve a large set of partial differential
42 equations (5). In fact, LBM can simulate many physical systems of interest such as Electromagnetism,
43 Plasma, Multiphase flow, Schrödinger equation etc. (6) (7) (8) (9). With this in mind, we keep our
44 method general and show evidence our method works equally well on Electromagnetic simulations.
45 However, because the dominate use of LBM is on fluid flow problems we center discussion on this
46 subject.

47 Our work has the following contributions.

- 48 • It allows for simulations to be generated with less memory then the original flow solver.
49 There is a crucial need for such methods because memory requirements grow cubic to grid
50 size in 3D simulations. In practice, this quickly results in the need for large GPU clusters
51 (10) (11).
- 52 • Once our model is trained, it can be used to generate significantly larger simulations.
53 This allows the model to learn from a training set of small simulations and then generate
54 simulations as much as 16 times bigger with little effect in accuracy.
- 55 • Our method is directly applicable to a variety of physics simulations, not just fluid flow. We
56 show this with our electromagnetic example and note that the changes to our model are
57 trivial.

58 2 Related Work

59 Recently, there have been several papers applying neural networks to fluid flow problems. Guo etc.
60 (2) proposed to use a neural network to learn a mapping from boundary conditions to steady state
61 flow. Most related to our own work, Yang etc. (3) and Tompson etc. (1) use a neural network to solve
62 the Poisson equation in order to accelerate Eulerian fluid simulations. The key difference between
63 this and Lat-Net is its ability to compress the memory usage and the generality of our method to other
64 physics simulations.

65 There has also been an increasing body of work applying neural networks to other physics modeling
66 problems. For example, neural networks have been readily adopted in many chemistry applications
67 such as predicting molecular properties from descriptors, protein contact prediction and computational
68 material design (12). Very recently, neural networks have been applied to quantum mechanics
69 problems as seen in Mills etc. (13) and Giuseppe etc. (14) where neural networks are used to
70 approximate solutions to the Schrödinger equation. In high energy Physics, Paganini etc. (15) uses a
71 generative adversarial networks (GAN)(16) to model electromagnetic showers in a longitudinally
72 segmented calorimeter. Many of these applications are relatively recent and indicate a resurgence of
73 interest in applications of neural networks to modeling physics.

74 Reduced order Modeling is an area of research that focuses on techniques to reduce the dimensionality
75 and computational complexity of mathematical models. A Reduced order model (ROM) is constructed
76 from high-fidelity simulations and can subsequently be used to generate simulations for lower
77 computation. The most popular ROM method for fluid dynamics is Galerkin projection (17) (18).
78 This method uses Proper Orthogonal Decomposition to reduce the dimensionality of flow simulations
79 and then finds the dynamics on this reduced space. There are other methods that build on this such
80 as reduced basis methods and balanced truncation (19) (20). While these approaches are centered
81 around the Navier stokes equation and thus not directly comparable to our own, we note that the
82 compression mapping present in these methods is typically quite simple. Given the recent success
83 neural networks have had in creating well structured encodings (such as Variational Autoencoders
84 (21) (22)), we feel our approach is well justified.

85 3 Deep Neural Networks for Compressed Lattice Boltzmann

86 In this section, we present our model for compressing Lattice Boltzmann simulations.

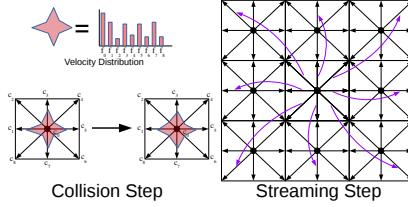


Figure 1: Illustration of the Lattice Boltzmann update steps

87 3.1 Review: The Lattice Boltzmann Method

88 In a Lattice Boltzmann simulation, the domain is discretized into an equal sized Cartesian grid. Each
 89 cell of this grid contains a velocity distribution function f_i that describes the velocity of flow at that
 90 point. f_i has values ranging over i that correspond to the $\{\vec{c}_i\}$ directions of flow. In our 2 dimensional
 91 simulations, there are 9 such directions (D2Q9 scheme). A figure showing this grid structure is seen
 92 in 1. From the distribution function f_i , one can calculate the density (p) and velocity (\vec{u}) of the fluid
 93 flow with the following equations.

$$p = \sum_i f_i \quad \text{and} \quad \vec{u} = \sum_i \vec{c}_i f_i \quad (1)$$

94 The lattice states are updated with two separate steps, the collision step and the streaming step. The
 95 collision step mimics the flow interacting with itself and is updated in the following way,

$$f_i^t(x, t + \delta_t) = f_i(x, t) + \frac{1}{\tau}(f_i^{eq} - f_i) \quad (2)$$

96 where τ is the relaxation constant and f_i^{eq} is the flow equilibrium. For our simulations, we use the
 97 f_i^{eq} from the Lattice Bhatnagar-Gros-Krook (LBGK) scheme (23). After the collision step is applied,
 98 the flow propagates to adjacent cells following the streaming step.

$$f_i(x + c_i, t + \delta_t) = f_i^t(x, t + \delta_t) \quad (3)$$

99 This step will contain bounce back if one of the adjacent cells is a boundary. Figure 1 illustrates these
 100 steps for the 2 dimensional case.

101 It is interesting to note the simplicity of this method and its similarity to convolutional neural networks.
 102 In fact, if we treat the lattice state f as a tensor of size $(n_x, n_y, 9)$, as we do for the remainder of this
 103 paper, the streaming operator can be mimicked with a 3 by 3 convolution and the collision step can
 104 be performed with a 1 by 1 convolution (D2Q9). This offers a unique way to interpret our method. In
 105 a some sense, we are taking a large convolutional neural network and compressing it onto a much
 106 smaller and more memory efficient network. With this mental picture in mind, we now describe our
 107 approach.

108 3.2 Proposed Architecture

109 Figure 2 shows a sketch of the model. The figure can be understood by following the arrows starting
 110 from the flow state f_t and the boundary b . We treat f_t as a tensor with shape $(n_x, n_y, 9)$ for the
 111 2D case and $(n_x, n_y, n_z, 15)$ for the 3D case. The boundary is treated as a binary tensor of shape
 112 $(n_x, n_y, 1)$ and $(n_x, n_y, n_z, 1)$ with the value being 1 if the cell is solid. Bellow we walk through
 113 each step of our method.

114 First, we compress both the state of the fluid simulation f_t and the boundary conditions b using
 115 two separate neural networks ϕ_{enc} and ϕ'_{enc} respectively. The result from ϕ_{enc} is a compressed
 116 representation of the flow g_t and the result of ϕ'_{enc} are two tensors b_{mul} and b_{add} of equal size to g_t .
 117 These three tensors represent the entirety of the compressed state of the simulation.

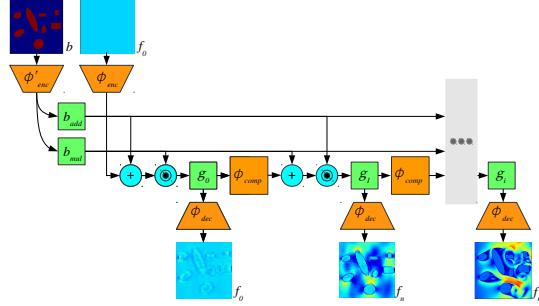


Figure 2: Illustration of the Lat-Net architecture

118 In a Lattice Boltzmann solver, the boundary conditions are used at each time-step to add bounce
 119 back to the streaming step. In a similar way, our model applies the compressed boundary to the
 120 compressed state every time-step. We do this in the following way,

$$g_t = (g_t \odot b_{mul}) + b_{add} \quad (4)$$

121 This method proved extremely successful at keeping the boundary information firmly planted through
 122 the duration of the simulation. This method of applying boundary conditions was inspired by (24)
 123 where they use a similar method to combine foreground and background information in video
 124 prediction. After the boundary is applied to g_t , we can run the state through another neural network
 125 to emulate the dynamics, i.e. $\phi_{comp} : g_t \rightarrow g_{t+1}$. Each step of ϕ_{comp} is equivalent to n time-steps
 126 of the Lattice Boltzmann solver. For example, in the 2 dimensional simulation, each step of ϕ_{comp}
 127 mimics 120 steps of the Lattice Boltzmann solver. Once g_t is computed, we can extract out the
 128 generated state of the simulation with a decoder network ϕ_{dec} .

129 3.3 Network Implementation Details

130 We implement 2 networks trained on 2 dimensional and 3 dimensional lattice simulations. The
 131 encoder, compression mapping and decoder pieces of the 2D network are each a series of 3 by 3
 132 residual blocks with the sequences 4x(down res-res)-res, 4x(res), and 3x(transpose conv-res-res)-
 133 transpose conv, respectively (25). For the 3D network, the sequences are 2x(down res)-res, 3x(res),
 134 and (transpose conv-res-transpose conv) where 3 by 3 by 3 convolutions are used. The down residual
 135 blocks are created by changing the first convolution to have kernel size 4 by 4, stride 2 and double the
 136 filter size. The up sampling is achieved with transpose convolutions of kernel size 4 by 4, stride 2 and
 137 half the filter size. For the last residual block on the 3D network, the filter size is halved once again.

138 As mentioned above, each network is kept entirely convolutional. Fluid flow is inherently spatially
 139 correlated so using convolutional layers allows this spatial information to be preserved. Keeping the
 140 network convolutional also allows different input sizes to be used. This is how our model is able to
 141 train on small simulations and then generate larger simulations.

142 Residual connections have been used in many deep learning architectures with much success. Adding
 143 residual connections allows for much deeper networks to be trained, often resulting in improved
 144 results (25). When training our model, it is necessary to unroll the compression network over several
 145 time-steps. This has the same effect as making the network deeper. For this reason, it seems logical to
 146 take advantage of this network architecture. We have seen that removing these residual connections
 147 results in much slower convergence and worse accuracy.

148 3.4 Training Details

149 Lat-Net is trained by unrolling the network and comparing the generated flow with the true. Our
 150 loss function is Mean Squared Error (MSE) with Image Gradient Difference Loss (GDL) (26). The
 151 GDL is multiplied by λ_{GDL} and then added to the MES. In all our experiments, λ_{GDL} is set to 0.2.
 152 Removing the GDL tended to produce less accurate models. Lat-Net is unrolled 5 time-steps and
 153 then trained with the Adam optimizer (27).

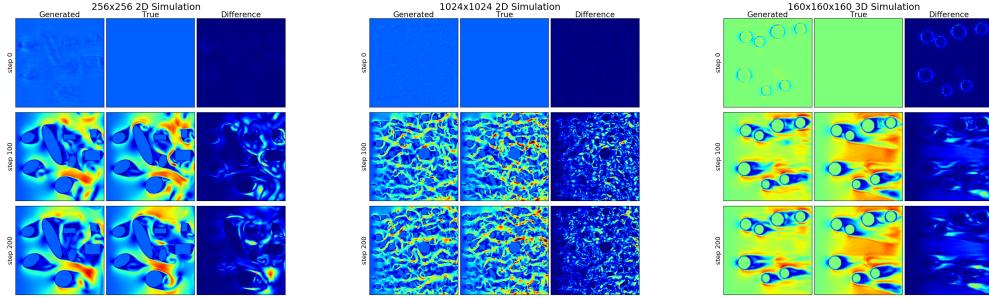


Figure 3: A visual comparison of flows generated by Lat-Net and the Lattice Boltzmann method. Each figure shows the Generated, True, and Difference of the flow for various time-steps.

154 4 Experiments

155 In this section, we describe our experiments testing Lat-Net on a variety of problems. Our experiments
 156 are designed to test our models ability to generate large simulations as well as its transferability to
 157 new boundary geometries. We also explore computation time and working memory usage. Finally,
 158 we briefly show results applying this method to electromagnetic simulations.

159 4.1 Dataset Generation

160 In order to train and test our model, we generate sets of fluid and electromagnetic simulations. All
 161 simulations were generated with the MechSys library (28).

162 The train set for the 2D fluid simulations consists of 50 runs of grid size 256 by 256 and 9 directional
 163 flows in the lattice Boltzmann solver (D2Q9 scheme)(23). The simulations use periodic boundary
 164 conditions on top and bottom as well as uniform inlet flow and outlet flow of 0.04 from the left and
 165 right. 8 Objects are placed randomly with height and width sizes ranging from 140 to 20 cells. The
 166 test set for the 2 dimensional simulations consists of 10 runs of size 256 by 256, and 5 runs of size
 167 1024 by 1024 with the same boundary conditions and object densities. We also generate a test set of
 168 size 256 by 512 with vehicle cross sections as objects. There are 28 cross sections used ranging from
 169 trucks to minivans. For all 2 dimensional simulations, the ratio of network steps to Lattice Boltzmann
 170 steps is 1 to 120.

171 The train set for the 3D fluid simulations consists of 50 runs of grid size 40 by 40 by 160 and 15
 172 directional flows in the lattice Boltzmann solver (D3Q15 scheme)(23). Similar to the 2D simulations,
 173 periodic boundary conditions are used with same inlet and outlet flow. 4 spheres are randomly placed
 174 with height and width 24. The reason different object geometries and sizes were not explored was
 175 due to the fact that smaller objects or objects with complex geometries tended to have too course
 176 a resolution for the lattice Boltzmann solver and larger objects required too large a simulation size.
 177 The test set comprises 10 runs of 40 by 40 by 160 and 5 runs of 160 by 160 by 160 simulations with
 178 the same object density. The ratio of network steps to Lattice Boltzmann steps is 1 to 60.

179 The train set for the electromagnetic simulations are grid size 256 by 256 with periodic boundaries.
 180 An electromagnetic wave is initialized in the top of the simulations and proceeds to interact with
 181 randomly placed objects of different dielectric constants. When the wave hits these objects, the
 182 reflection and refraction phenomenon is seen. The test set consists of simulations of size 512 by 512
 183 with the same object density.

184 4.2 Generating Simulations

185 A key component of our model is its ability to generate larger simulations than those trained on. To
 186 test its effectiveness in doing so, we compare the accuracy of generated 2D and 3D simulations to
 187 ground truth simulations.

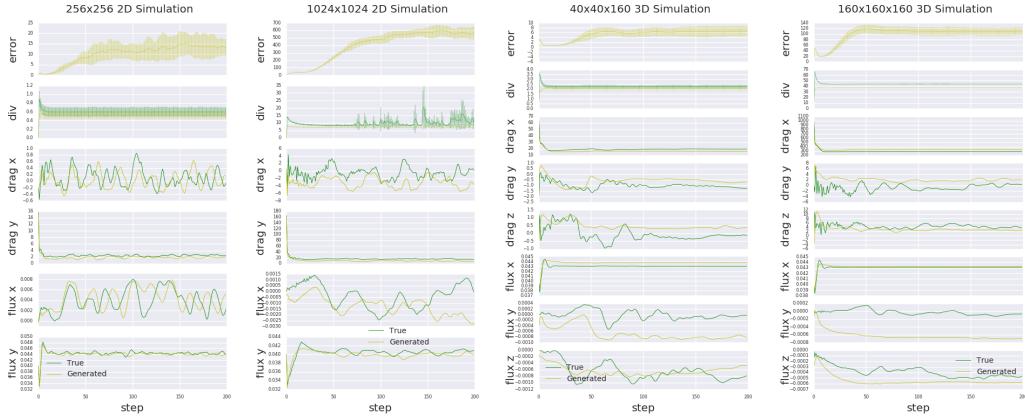


Figure 4: Comparison plot of the flows generated by Lat-Net and the Lattice Boltzmann method. Each plot shows the average mean squared error of the true and generated generated along with the average divergence of the velocity vector field for both simulations. The standard deviation is also displayed. In addition, the calculated values for drag and flux are displayed for a single simulation run. For the 1024 by 1024 simulation, the flow produced by the Lattice Boltzmann solver tended to produce instabilities resulting in the chaotic divergence observed.

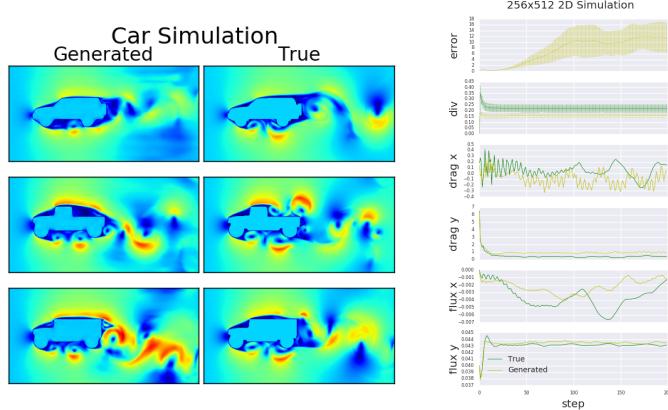


Figure 5: Comparison of generated flows for the vehicle cross section dataset. The images show the generated and true flow at step 100 for 3 different cars in the dataset. The plot shows the same values as described in 4.

Comparing the accuracy of our simulations require some consideration. The naive approach is to compare the MSE between the generated and true simulation at various time-steps. The problem with this approach is that fluid flow is a chaotic dynamical system and small perturbations in flow quickly compound leading to dramatic differences at latter times. For this reason, we compare a variety of metrics in evaluating the generated flows accuracy. Similar to (1), we compare the divergence of the generated and true velocity vector field to test our models stability. We also compare the computed values of drag and flux. Flow simulations are often run to calculate such values so comparing this is a strong indicator of our models real world applicability. The drag is calculated directly from the lattice state via the momentum transfer method (23). The flux value is the average of the flux in each non-boundary cell. These values can be used to calculate important quantities such as the drag coefficient and Reynolds number. Lastly, we visually inspect the produced flow to check for instabilities and blurring effects.

Table 1: Computation Time of Network

Simulation	Comp. Size	Comp. Mapping	Full State	Plane	Line	Point
(1024, 1024, 9)	(64, 64, 128)	2.7 ms	36.2 ms	NA	6.7 ms	6.6 ms
(160, 160, 160, 15)	(40, 40, 40, 64)	23.1 ms	272.1 ms	38.2 ms	25.6 ms	24.1 ms

200 In figure 4, we see the predicted values for different grid sizes in 2D and 3D simulations. In the 2D
 201 simulations, Lat-Net is able to effectively transfer to larger domain sizes with very similar calculated
 202 values of drag and flux. The generated flow also maintains its stability even after hundreds of steps. In
 203 the 3D simulation we see that, while our model predicts realistic values for the 40x40x160 simulation,
 204 it tends to have a slight bias in the direction of flow that manifests itself in the 160x160x160
 205 simulation.

206 When visually inspecting our produced flows (figure 3), we see a slight blurring effect but overall
 207 similar structure in the 2D flows. We attribute this blur effect to the dimensionality reduction and use
 208 of MSE. This can possibly be overcome with the use of generative adversarial network (16) where
 209 the loss is derived from a discriminator network. Another solution may be to craft a loss function that
 210 takes advantage of the statistical properties of flow (7). We leave these pursuits to future work.

211 There is a distinct difference in the generated and true flow for the 3D 160x160x160 simulation.
 212 While the generated flow appears accurate close to the objects, in regions far between objects the
 213 network tends to underestimate the flow velocity. We believe this is due to these types of regions not
 214 being present in the train set and is the probable cause for the biases seen in the drag and flux. As
 215 mentioned above, our 3D train set is limited due to memory constraints and so developing a diverse
 216 train set to overcome this proved difficult.

217 The boundaries used in the above evaluation are drawn from the same distribution as the train set.
 218 This motivates the question of how our model performs on drastically different geometries. To test
 219 this, we apply our model to predicting flow around vehicle cross sections. Surprisingly, even though
 220 our model is only trained on flows around simple shapes (ovals and rectangles) it can effectively
 221 generalize to this distinctly different domain. In figure 5, we see the predicted flows are quite similar
 222 but with the same blurring effect. Calculating the same values as above, we see the flow is stable and
 223 produces similar drag and flux.

224 4.3 Computation and Memory Compression

225 In this section, we investigate the computational speed-up of our model. The standard performance
 226 metric for Lattice Boltzmann Codes is Million Lattice Updates per Second (MLUPS). This metric is
 227 calculated by the following equation,

$$MLUP = \frac{n_x \times n_y \times n_z \times 10^{-6}}{\text{Compute Time}} \quad (5)$$

228 where n_x , n_y , and n_z are the dimensions of the simulation. For 3 dimensional simulations like the
 229 ones seen in this paper, a speed of 1,200 MLUPS can be achieved with a Nvidia K20 GPU and single
 230 precision floats (29). We use this as our benchmark value to compare against.

231 The computation time and memory usage of the encoder can be neglected because this is a one time
 232 cost for the simulation. In addition, if the simulation is started with uniformly initialized flow as seen
 233 in our experiments, the computation to compress the flow is extremely redundant and can easily be
 234 optimized.

235 As seen in table 1, the computation time of the compression mapping is 23.1 ms for a 3D simulation of
 236 grid size 160 by 160 by 160. Because each step of the compression mapping is equivalent to 60 Lattice
 237 Boltzmann steps, this equates to 10,600 MLUPS and a roughly 9x speed increase (a similar speed-up
 238 is seen with the 2D simulation). This does not give a complete picture though. Once the compressed
 239 states have been generated, the flow must be extracted with the decoder. Unfortunately, this requires
 240 considerable amounts of computation and memory because it involves applying convolutions to
 241 the full state size. Fortunately, there are ways around this. In many applications of CFD, it is not
 242 necessary to have the full state information of the flow at each time-step. For example, calculating
 243 the drag only requires integrating over the surface of the object. By using the convolutional nature
 244 of the decoder, we can extract specific pieces of the flow without needing to compute the full state.

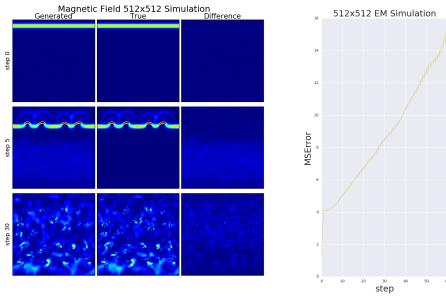


Figure 6: Comparison of generated Electromagnetic fields. The images show the true and generated magnetic field at various time-steps. The reflection and refraction phenomena can clearly be seen in both. The plot shows the mean squared error of the true and generated simulation.

245 In table 1, we show computation times for extracting flow information of a plane, line and single
 246 point. While these computations can still be somewhat expensive, they do not necessarily need to be
 247 performed at every time-step and require very little working memory.

248 Unfortunately, there are some measurements that do require the full state information to compute
 249 such as the average flux seen in our tests. Our method is currently unable to handle these without
 250 requiring high run-times and large working memory. A possible solution is training a separate neural
 251 network that takes in the compressed state and predicts the desired measurement. This would negate
 252 the need to extract out the full state and keep memory usage low. We leave this and similar ideas for
 253 future work.

254 While Lat-Net compresses the simulation state size by more than an order of magnitude, the working
 255 memory requirements for the compression network must be considered. A typical GPU based Lattice
 256 Boltzmann solver requires around 1.5 times as much working GPU memory as the memory size
 257 of the lattice (29). For example, the maximum sized D3Q15 lattice that can fit on a GPU with 8
 258 Gigabytes is 446^3 . We have observed that in our implementation of Lat-Net, the maximum 3D
 259 compression network we can run with an 8 Gigabyte GPU corresponds to a lattice size of 672^3 . This
 260 represents a 3.4x efficiency gain in working memory usage. While this is certainly a nontrivial gain,
 261 we feel that further improvements can be realized with a more memory efficient implementation of
 262 the compression mapping.

263 4.4 Electromagnetic Results

264 Finally, we illustrate the generality of our method by applying it to electromagnetic simulations. The
 265 same neural network architecture is used as in the 2D flow simulations with the only difference being
 266 the filter size on the compression is half of that in the flow network. The loss is kept identical however
 267 the lattice values are scaled up by a factor of 10 so they are on the same range as the flow lattice
 268 values. In figure 6, we see very similar waves formed with the same reflection and refraction.

269 5 Conclusion

270 Fluid Simulations are incredibly important for a variety of tasks however they are extremely computa-
 271 tion and memory intensive. In this work, we have developed a unique method to overcome this using
 272 deep neural networks. We have demonstrated it is capable of accurately reconstructing a variety of
 273 simulations under different conditions with significantly less computation and memory. We have also
 274 shown that our method can be readily applied to other physics simulations such as electromagnetic
 275 simulations. While our method has proved successful on the problems in this paper, there is still
 276 significant room for improvement. A loss function that either takes into account the statistical nature
 277 of the flow or uses recent advances in GANs could produce sharper, more realistic flow. Training
 278 a network to extract desired measurements from the compressed state such as average flux would
 279 overcome the current memory limitation for such a task. We leave these and other improvements for
 280 future work.

281 **References**

- 282 [1] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, “Accelerating eulerian fluid simulation
283 with convolutional networks,” *arXiv preprint arXiv:1607.03597*, 2016.
- 284 [2] X. Guo, W. Li, and F. Iorio, “Convolutional neural networks for steady flow approximation,” in
285 *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and
286 Data Mining*, pp. 481–490, ACM, 2016.
- 287 [3] C. Yang, X. Yang, and X. Xiao, “Data-driven projection method in fluid simulation,” *Computer
288 Animation and Virtual Worlds*, vol. 27, no. 3-4, pp. 415–424, 2016.
- 289 [4] G. R. McNamara and G. Zanetti, “Use of the boltzmann equation to simulate lattice-gas
290 automata,” *Physical review letters*, vol. 61, no. 20, p. 2332, 1988.
- 291 [5] S. Galindo-Torres, A. Scheuermann, and R. Puscasu, “A lattice boltzmann solver for maxwell
292 equations in dielectric media,”
- 293 [6] M. Mendoza and J. Munoz, “Three-dimensional lattice boltzmann model for electrodynamics,”
294 *Physical Review E*, vol. 82, no. 5, p. 056708, 2010.
- 295 [7] T. Kim, N. Thürey, D. James, and M. Gross, “Wavelet turbulence for fluid simulation,” in *ACM
296 Transactions on Graphics (TOG)*, vol. 27, p. 50, ACM, 2008.
- 297 [8] L. Zhong, S. Feng, P. Dong, and S. Gao, “Lattice boltzmann schemes for the nonlinear
298 schrödinger equation,” *Physical Review E*, vol. 74, no. 3, p. 036704, 2006.
- 299 [9] X. Shan and H. Chen, “Lattice boltzmann model for simulating flows with multiple phases and
300 components,” *Physical Review E*, vol. 47, no. 3, p. 1815, 1993.
- 301 [10] N. Onodera, T. Aoki, T. Shimokawabe, and H. Kobayashi, “Large-scale les wind simulation
302 using lattice boltzmann method for a 10 km× 10 km area in metropolitan tokyo,” *TSUBAME
303 e-Science Journal Global Scientific Information and Computing Center*, vol. 9, pp. 1–8, 2013.
- 304 [11] W. Xian and A. Takayuki, “Multi-gpu performance of incompressible flow computation by
305 lattice boltzmann method on gpu cluster,” *Parallel Computing*, vol. 37, no. 9, pp. 521–535,
306 2011.
- 307 [12] G. B. Goh, N. O. Hodas, and A. Vishnu, “Deep learning for computational chemistry,” *Journal
308 of Computational Chemistry*, 2017.
- 309 [13] K. Mills, M. Spanner, and I. Tamblyn, “Deep learning and the schr\“ odinger equation,” *arXiv
310 preprint arXiv:1702.01361*, 2017.
- 311 [14] G. Carleo and M. Troyer, “Solving the quantum many-body problem with artificial neural
312 networks,” *Science*, vol. 355, no. 6325, pp. 602–606, 2017.
- 313 [15] M. Paganini, L. de Oliveira, and B. Nachman, “CaloGAN: Simulating 3D High Energy Particle
314 Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks,”
315 *ArXiv e-prints*, May 2017.
- 316 [16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and
317 Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*,
318 pp. 2672–2680, 2014.
- 319 [17] C. W. Rowley, T. Colonius, and R. M. Murray, “Model reduction for compressible flows using
320 pod and galerkin projection,” *Physica D: Nonlinear Phenomena*, vol. 189, no. 1, pp. 115–129,
321 2004.
- 322 [18] M. F. Barone, I. Kalashnikova, M. R. Brake, and D. J. Segelman, “Reduced order modeling
323 of fluid/structure interaction,” *Sandia National Laboratories Report, SAND No*, vol. 7189,
324 pp. 44–72, 2009.
- 325 [19] K. Veroy and A. Patera, “Certified real-time solution of the parametrized steady incompressible
326 navier–stokes equations: rigorous reduced-basis a posteriori error bounds,” *International
327 Journal for Numerical Methods in Fluids*, vol. 47, no. 8-9, pp. 773–788, 2005.

- 328 [20] C. W. Rowley, “Model reduction for fluids, using balanced proper orthogonal decomposition,”
329 *International Journal of Bifurcation and Chaos*, vol. 15, no. 03, pp. 997–1013, 2005.
- 330 [21] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint*
331 *arXiv:1312.6114*, 2013.
- 332 [22] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, “Embed to control: A locally
333 linear latent dynamics model for control from raw images,” in *Advances in Neural Information*
334 *Processing Systems*, pp. 2746–2754, 2015.
- 335 [23] Z. Guo and C. Shu, *Lattice Boltzmann method and its applications in engineering*, vol. 3. World
336 Scientific, 2013.
- 337 [24] C. Vondrick, H. Pirsiavash, and A. Torralba, “Generating videos with scene dynamics,” in
338 *Advances In Neural Information Processing Systems*, pp. 613–621, 2016.
- 339 [25] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in
340 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778,
341 2016.
- 342 [26] M. Mathieu, C. Courville, and Y. LeCun, “Deep multi-scale video prediction beyond mean square
343 error,” *arXiv preprint arXiv:1511.05440*, 2015.
- 344 [27] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint*
345 *arXiv:1412.6980*, 2014.
- 346 [28] D. Pedrosso, R. Durand, and S. Galindo, “Mechsys, multi-physics simulation library,” 2015.
- 347 [29] M. Januszewski and M. Kostur, “Sailfish: A flexible multi-gpu implementation of the lattice
348 boltzmann method,” *Computer Physics Communications*, vol. 185, no. 9, pp. 2350–2368, 2014.