

DEEP NEURAL NETWORKS FOR GEOMETRIC MULTIGRID METHODS

Nils Margenberg¹, Robert Jendersie², Thomas Richter², Christian Lessig²

¹Helmut-Schmidt-University, Holstenhofweg 85, Hamburg, 22043, Germany

²Otto-von-Guericke University, Universitätsplatz 2, Magdeburg, 39106, Germany

Key words: FEM, Geometric Multigrid Methods, Deep Neural Networks, Scientific Machine Learning

Abstract. We investigate scaling and efficiency of the deep neural network multigrid method (DNN-MG), a novel neural network-based technique for the simulation of the Navier-Stokes equations that combines an adaptive geometric multigrid solver with a recurrent neural network with memory. The neural network replaces in DNN-MG one or multiple finest multigrid layers and provides a correction for the classical solve in the next time step. This leads to little degradation in the solution quality while substantially reducing the overall computational costs. At the same time, the use of the multigrid solver at the coarse scales allows for a compact network that is easy to train, generalizes well, and allows for the incorporation of physical constraints. In this work, we investigate how the network size affects training and solution quality and the overall runtime of the computations.

1 Introduction

In the last decade, deep neural networks had great success with tasks such as machine translation and image classification and recently also showed surprising effectiveness for even more challenging problems such as image generation and playing games, cf. [8]. Underlying these results is the ability of deep neural networks (DNNs) to accurately approximate high dimensional mappings when sufficiently deep and complex network architectures are used, training is performed on large or very large amounts of data, and powerful hardware, such as GPUs, are employed.

The aforementioned success of deep neural networks and related techniques leads to a growing interest to apply these also to problems in computational science and engineering, including for the simulation of partial differential equations (PDEs). Raissi, Karniadakis and co-workers [15, 16], for example, proposed physics-informed neural networks (PINNs) that directly learn the mapping from the input of a potentially parametric PDE to its solution. Kasim et al. [7] showed the effectiveness of deep neural networks to approximate a wide range of partial differential equations when also the network architecture is part of the training. Various works [10, 11, 9] recently also developed frameworks for operator learning. A natural connection between neural networks and dynamical systems has been observed for example by E [4] and Haber and Rothotto [5, 3]. This can also be exploited when neural networks are used for the solution of

PDEs, for example to improve the stability of the learning. A more detailed discussion of related work can, for example, be found in [13].

A principle question for the applicability of deep neural networks for PDEs is how the equations themselves and known results on their solutions should be incorporated. Furthermore, there exists already a large number of numerical schemes for partial differential equations and one has to ask under what circumstances a neural network can outperform these. To address these questions, we investigated in our previous work how an adaptive multigrid solver, i.e. one of the most efficient classical techniques for partial differential equations, can be combined with deep neural networks for the solution of the Navier-Stokes equations. The resulting deep neural network multigrid method (DNN-MG) [13] combines a geometric multigrid solver with a recurrent neural network with memory to replace computations on one or multiple finest mesh levels with network based corrections. DNN-MG exploits the typical structure of multigrid methods with a hierarchy of finite element meshes and function spaces. In contrast to the typical application of multigrid, the finite element problem is not solved on the finest mesh level but on an intermediate one. Even finer levels are only processed by the neural network. Through this, DNN-MG achieves a significant speed-up with only a small degradation in quality while its design for small patches ensures that the technique generalizes well and is comparatively easy to train. The divergence freedom of learned corrections for the Navier-Stokes equations was investigated in [12].

In this work, we investigate the scalability with respect to the size of the neural network. Our results show that larger networks are able to capture the flow behavior better and improve important flow quantities. The increased network size has a minor impact on training and test time. DNN-MG compared to a pure numerical simulation can even reduce the simulation time.

2 Deep neural network multigrid solver

In this section we give a short summary of the deep neural network multigrid solver [13]. Although it is applicable to other equations, we will consider it in the context of the incompressible Navier-Stokes equations.

2.1 Finite Element Discretization of the Incompressible Navier-Stokes equations

The incompressible Navier-Stokes equations are given by

$$\nabla \cdot v = 0, \quad \partial_t v + (v \cdot \nabla)v - \frac{1}{\text{Re}} \Delta v + \nabla p = f \quad \text{on } [0, T] \times \Omega \quad (1)$$

Here $v: [0, T] \times \Omega \rightarrow \mathbb{R}^2$ is the velocity, $p: [0, T] \times \Omega \rightarrow \mathbb{R}$ the pressure, $\text{Re} > 0$ the Reynolds number, and f an external force. At time $t = 0$ it holds $v(0, \cdot) = v_0(\cdot)$, on the Dirichlet boundary Γ^D we prescribe $v = v^D$, where $v^D = 0$ on fixed boundaries or v^D indicates an inflow profile and on outflow boundaries Γ^D we consider the do-nothing outflow condition $\frac{1}{\text{Re}}(\vec{n} \cdot \nabla)v - p\vec{n} = 0$, which is a well established model for artificial boundaries [6]. By \vec{n} we denote the outward facing unit normal on the boundary $\partial\Omega$ of the domain.

We use a weak finite element formulation to discretize (1) with $v_h, \phi_h \in V_h = [W_h^{(2)}]^d$ and test functions $p_h, \xi_h \in L_h = W_h^{(r)}$. $W_h^{(r)}$ is here the space of continuous functions which are polynomials of degree r on each element $T \in \Omega_h$ and Ω_h is the mesh domain. The equal order

finite element pair $V_h \times L_h$ that we use does not fulfill the inf-sup condition. We hence use stabilization terms of local projection type [1] with parameter $\alpha_T = \alpha_0 \cdot \text{Re} \cdot h_T^2$ and projection $\pi_h : W_h^{(2)} \rightarrow W_h^{(1)}$ into the space of linear polynomials.

We use the second order Crank-Nicolson method for time discretization and in the n -th time step we determine the state $x_n = (v_n^1, \dots, v_n^d, p_n)$ given by an algebraic system of equations

$$\mathcal{A}_n(x_n) = f_n, \quad (2)$$

where the right hand side f_n also depends on the previous state x_{n-1} , details are given in the previous publications [13, 12]. Equation (2) is a large nonlinear system of algebraic equations that is solved by Newton's method with the initial guess $x_n^{(0)} = (v_{n-1}, p_{n-1})$. Each Newton step takes the form

$$\mathcal{A}'_n(x_n^{(l-1)}) w^{(l)} = f_n - \mathcal{A}_n(x_n^{(l-1)}), \quad x_n^{(l)} = x_n^{(l-1)} + w^{(l)} \text{ for } l = 1, 2, \dots \quad (3)$$

$\mathcal{A}'(x^{(l-1)})$ is the Jacobian of \mathcal{A} at $x^{(l-1)}$, which can be computed analytically for this problem, cf. [17, Sec. 4.4.2].

2.2 Deep Neural Network Multigrid Solver

For the solution of the Newton iteration described above we use GMRES and the geometric multigrid method as preconditioner, leading to a highly efficient solver. However, multiple GMRES steps and one up- and down-sweep of the multigrid method per GMRES step still result in a significant amount of computations. Considering the multigrid hierarchy with levels $1, 2, \dots, L$ most of the computation time (about 75%) is thereby spent on the finest mesh level (denoted as L). Solving on an additional level $L + 1$ therefore leads to a substantial increase in the computational costs.

The principle idea of the deep neural network multigrid solver (DNN-MG) is to advance in time using the Crank-Nicolson method on mesh level L , using the multigrid hierarchy on levels $1, 2, \dots, L$ for solving the linear systems, and then correct this coarse solution of the Navier-Stokes equations on one (or potentially multiple) finest mesh levels $L + 1$ with a neural network. An overview is provided in Algorithm 1. The network correction on level $L + 1$ enters the assembly of the right hand side at the next time step so that the enriched solution is preserved in time (see our previous work [13, Fig. 1] for further details). For such a correction to be meaningful, the network has to be able to (largely) retain the accuracy of the multigrid computation. Through the coarse multigrid solution as well as the nonlinear residual, the network has, however, highly informative input that makes this viable; in other words, the classical coarse solution as well as its error on finer levels provide a strong prior for the network correction.

To ensure our network correction can be computed efficiently and to facilitate generalization, we use the underlying mesh to subdivide our domain into patches, e.g. a mesh element or a structured assembly of a few adjacent elements, and apply the network on these. This greatly aids generalizability since the network no longer needs to correct the general flow and instead only local corrections for each patch are required. Furthermore, the local setup simplifies training and ensures that a large training corpus is given by just a few example flows. For the neural network, we use a recurrent one with memory, more specifically GRUs, so that complex flow behavior can be predicted and coherence in time is ensured.

Algorithm 1 DNN-MG for the solution of the Navier-Stokes equations. Lines 6–9 (blue) provide the modifications of the DNN-MG method compared to a classical Newton-Krylov simulation with geometric multigrid preconditioning.

```

1: for all time steps  $n$  do
2:   while not converged do ▷ Newton's method for Eq. 2
3:      $\delta z_i \leftarrow \text{MULTIGRID}(L, A_L^n, b_L^n, \delta z_i)$  ▷ Geometric multigrid
4:      $z_{i+1} \leftarrow z_i + \epsilon \delta z_i$ 
5:   end while
6:    $\tilde{v}_n^{L+1} \leftarrow \mathcal{P}(v_n^L)$  ▷ Prolongation on level  $L + 1$ 
7:    $d_n^{L+1} \leftarrow \mathcal{N}(\tilde{v}_n^{L+1}, \Omega_L, \Omega_{L+1})$  ▷ Prediction of velocity correction
8:    $b_{n+1}^{L+1} \leftarrow \text{Rhs}(\tilde{v}_n^{L+1} + d_n^{L+1}, f_n, f_{n+1})$  ▷ Set up rhs for next time step
9:    $b_{n+1}^L \leftarrow \mathcal{R}(b_{n+1}^{L+1})$  ▷ Restriction of rhs to level  $L$ 
10: end for

```

In summary, the key features that make DNN-MGs efficient and flexible are:

- (a) patchwise operation to ensure generalizability to unseen flow regimes and meshes;
- (b) GRUs with memory to capture complex flow behavior and ensure coherence in time;
- (c) use of the nonlinear residual in Eq. 2 as network input to have rich information about the sought correction.

Algorithmically, DNN-MG works as follows, see Alg. 1. The neural network-based correction is applied at the end of every time step after we computed an updated velocity v_n^L on level L (Alg. 1, l. 2-4). For this, v_n^L is first prolonged to level $L + 1$, yielding $\tilde{v}_n^{L+1} := \mathcal{P}(v_n^L)$. Then we compute the input to the neural network, which is evaluated individually for each patch P_i (a mesh element on level L in the present work). The inputs to the neural network are thereby also entirely local and include the residual of Eq. (2) on level $L + 1$, the prolonged velocity \tilde{v}_n^{L+1} , geometric properties such as the patch's aspect ratio, and the Péclet number over the patch. The network predicts a velocity correction $d_{n,i}^{L+1}$ and notably does not include the pressure. It enters only through the fine mesh residual, which is part of the input.

At the end of the neural network-based correction a provisional right hand side $b_{n+1}^{L+1} = \text{Rhs}(\tilde{v}_n^{L+1} + d_n^{L+1}, f_n, f_{n+1})$ of (3) is computed on level $L + 1$ and then restricted to level L . The corrected b_{n+1}^L is then used in the next time step, which is again improved by a neural network-based correction at the end of the step. Through this, the neural network based correction propagates back into the Newton solver and improves the numerical solution.

Before we can use DNN-MG, we train the network with a high fidelity finite element solution on the fine mesh level $L + 1$ using the loss function

$$\mathcal{L}(v^L, v^{L+1}; d^{L+1}) := \sum_{n=1}^N \sum_{P_i \in \Omega_{L+1}} \|v_{n+1}^{L+1} - (v_n^L + d_n^{L+1})\|_{l^2(P_i)}^2. \quad (4)$$

Here N is the number of time-steps in the training data. Since DNN-MG operates strictly local on patches, the loss accumulates the local residuals for each patch in the second sum. For a detailed description and results showing the generalizability, efficiency and efficacy, we refer to [13, 12].

3 Numerical experiments

In this section we investigate the scalability of DNN-MG with respect to the size of the neural network both in terms of its computational efficiency and the quality of the results.

3.1 Setup

As test problem we considered the classic benchmark of a laminar flow around an obstacle [19] governed by the Navier-Stokes equations. Analogous to [13] and different from [19] we use elliptic obstacles with varying aspect ratios instead of discs. This avoids that the neural networks can memorize flows and perform well in the experiments.

The neural networks of DNN-MG used for our experiments had the same overall structure as in our previous work [13] with GRU cells with memory at their heart. In contrast to [13], however, we replaced the convolutional layer with a dense one since it slowed down training considerably with only a negligible effect on the effectiveness of the network. To investigate the effect of the neural network size we parameterized the network by the hidden state size of the GRU-cells (32, 64) and the number of GRUs stacked on top of each other (1, 2, 3). The resulting network sizes are reported in Tab. 1.

We use the finite element library Gascoigne 3d [2] for numerical simulations and PyTorch [14] for the neural networks. The training data was obtained in the same manner described in previous works by running resolved finite element simulations consisting of 2560 elements and 8088 degrees of freedom and localizing the data to the time steps and to the patches of the mesh. The simulations were carried out on a coarser mesh with 640 elements resulting 2124 degrees of freedom. The training of the networks was performed on a NVIDIA Tesla V100. The numerical simulations were performed on the same machine equipped with 2 Intel Xeon E5-2640 v4 CPUs and running on 20 cores.

3.2 Training performance

The convergence behavior of the training for the different network configurations is reported in Fig. 1. We see that all networks show a similar behavior although the larger networks converge slightly faster and the loss is generally lower. The strongest effect on this behavior has the size of the hidden state with the GRUs of size $64 \times n$ performing best.

In Tab. 1 we see that the training time scales very well with the network size. Although the number of parameters is increased by almost one magnitude between the networks 32×1 and 64×3 , the training time grows only by 35%.

3.3 Application performance

Once trained, the network is used in the DNN-MG solver described in Section 2. Table 2 collects the runtimes for the overall simulation over 1050 time steps as well as the time required

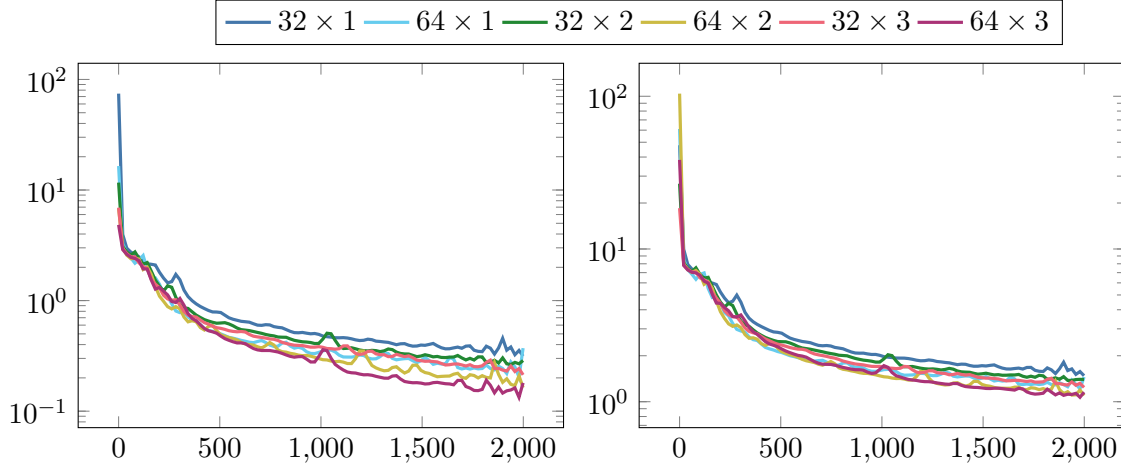


Figure 1: Average training (left) and validation (right) loss of different GRU-cells per patch (smoothed with gnuplots smooth bezier curve): A cell with dimension $m \times n$ consists of n GRUs stacked on top of each other, where the hidden states h are of size m .

for evaluating the neural network in DNN-MG. In most cases, the neural network contributes less than 4% to the overall computational time. We further note that about 1.5s of the network evaluation time must be attributed to data processing and communication between the finite element library and PyTorch. In an optimized implementation this time could most likely be reduced substantially.

The observed scaling in Tab. 2 does not always agree with what one would expect from the neural networks sizes in Tab. 1. There are multiple reasons for this. First, when the neural network correction is highly inaccurate then we need more Newton iterations in the next time step, resulting in higher runtimes such as for the 32×1 network. High quality predictions, on the other hand, improve the runtimes and for the 64×3 GRU-cell, for example, the time associated with the assembly routines was reduced by a factor of 4. Second, if the network deteriorates the solution and convergence of the Newton iteration is too slow we switch off the prediction for this time step. If this happens often, it leads to significantly less network evaluations and thus reduces runtimes of the network (this causes the spikes e.g. for 64×1 , 64×2) although at the price of a loss in accuracy.

In summary, the networks take up only a small fraction of the runtime and may even reduce the overall runtime through faster convergence of the Newton iteration. For the largest network 64×3 , for example, we achieve a speedup of almost 100% compared to the solution without a network. However, an ill-suited network can also negatively affect the convergence and hence slow down computations.

3.4 Accuracy of the networks

To compare the approximation power of the different neural network configurations, we evaluate the solution to the benchmark problem with respect to three functionals and also compute the error in the computed velocities and pressures. The first functional we consider is the squared

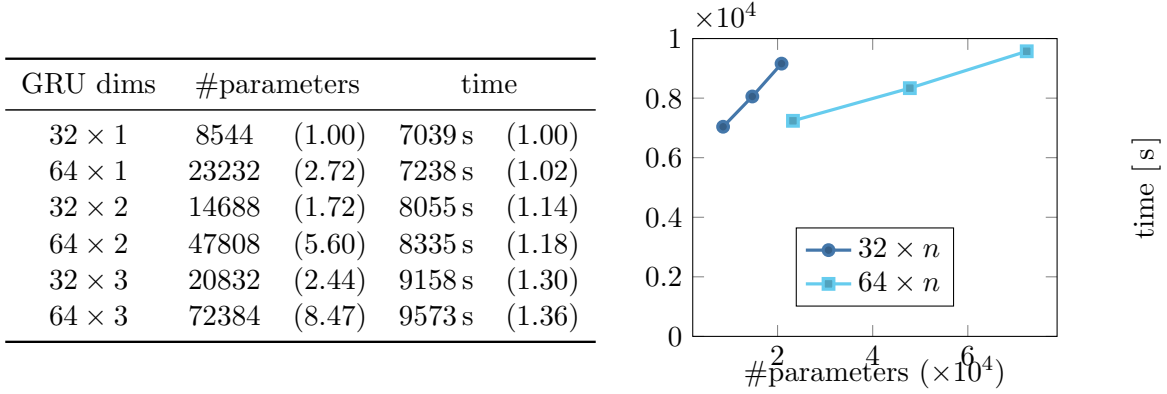


Table 1: GRU dimensions, parameters and the training time. In parentheses we list the factor compared to GRU 32×1 .

divergence

$$J_{\text{div}}(v(t)) = \int_{\Omega} |\nabla \cdot v(t)|^2 dx. \quad (5)$$

We know that the exact solution should satisfy $J_{\text{div}}(v) = 0$. However, since our finite element approach is not strictly divergence free and since we are adding artificial stabilization terms, the numerical solution will carry a divergence error that is also present in the training data. Hence, also the neural network correction will not be exactly divergence free although it should not deteriorate it further. As suggested in [19], the second and third functional we are considering are the drag and lift forces acting on the obstacle

$$J_{\text{drag/lift}}(v(t), p(t)) = - \int_{\partial\Gamma} \left(\frac{1}{Re} \nabla v(t) - p(t)I \right) \vec{n} \cdot \vec{e}_{d/l} ds, \quad (6)$$

where we denote by Γ the boundary of the obstacle, by \vec{n} the unit normal vector pointing into the obstacle and by $\vec{e}_d = \vec{e}_x$ and $\vec{e}_l = \vec{e}_y$ the Cartesian unit vectors.

Figure 2 shows all three functionals over the temporal interval $[9\text{ s}, 10\text{ s}]$, where the solution reached the periodic limit cycle (the phase were adjusted so that they agree on the first maximum in $[9\text{ s}, 10\text{ s}]$). We use a high resolution simulation $\text{MG}(L+1)$ as reference and $\text{MG}(L)$, that is the uncorrected simulation on level L , as base line. We observe that most functional outputs are significantly improved by DNN-MG with the best performing neural networks (cf. Table 3). Nearly exact values are obtained in case of the lift functional (right) but also drag (middle) is getting close to the reference value, in particular for 64×3 . For the divergence J_{div} the improving is more modest although compared to our previous results [12] ([12, Fig. 1] in the corresponding preprint) with a network with 32×1 GRU cells we still obtain a lower divergence.

In Fig. 3 we show the relative l_2 errors of the velocity and pressure with respect to the fine mesh solution $\text{MG}(L+1)$. The results demonstrate that the largest network 64×3 is able to produce robust and accurate predictions for large time intervals. This network is also able to substantially reduce the phase error that results from the interplay between spatial and temporal discretization [18, 13] and leads to lower frequency oscillations on coarser meshes. Our previous results and the smaller networks show the same effect for the DNN-MG approach (low frequency

type	runtime	NN-Eval	
$\text{MG}(L+1)$	538.45 s		(0%)
$\text{MG}(L)$	453.06 s		(0%)
32×1	705.53 s	21.84 s	(3.1%)
64×1	284.63 s	2.85 s	(1.0%)
32×2	257.72 s	7.44 s	(2.9%)
64×2	224.95 s	4.14 s	(1.8%)
32×3	200.42 s	11.01 s	(5.5%)
64×3	230.6 s	5.47 s	(2.3%)

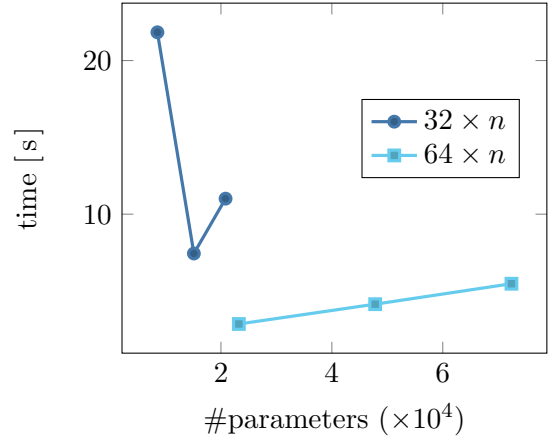


Table 2: Timing results for the simulations with the different GRUs we tested with *runtime* referring to the timing of the whole simulation and *NN-eval* the time spent on evaluating the neural network. In general ≈ 1.5 s of the latter is spent on data processing. In parentheses we specify the time taken relative to the runtime in percent.

oscillations in Fig. 3) but using a larger neural network cures this defect. We also refer to Fig. 2 which shows (e.g. in the case of the lift functional) that the GRU 64×3 solution is almost perfectly in phase with the fine mesh solution $\text{MG}(L+1)$ while a lower frequency is found for the coarse mesh $\text{MG}(L)$.

4 Discussion

The presented results show the scalability of the DNN-MG method with respect to the network size. We observed that larger networks lead to DNN-MG simulations that capture flow behavior more accurately. At the same time, the increased number of parameters in the network has little impact on the training times and can (through the faster convergence of the Newton solve) even improve the runtime performance compared to a standard MG solver. Smaller networks are often not able to capture the flow behavior and even render the solution useless at times. In some cases, they also have a negative impact on the runtime performance due to the interaction of the neural network and the Newton solve.

The model problem of a laminar flow around an obstacle that we considered is relatively simple and it might appear that a multigrid solver is not necessary in this case. However, the structure of the multigrid solver with different mesh levels is a necessary ingredient for the DNN-MG method. The multigrid structure will, furthermore, ensure optimal scaling for larger problems and especially in 3d, regimes we will consider in future work.

5 Conclusion

We have investigated the scalability of the DNN-MG method with respect to the neural network size in terms of accuracy, runtime, and training time. We showed that larger networks are able to more accurately predict flow features and this allowed us to substantially improve over previous results for the DNN-MG method. In particular, our largest network 64×3 is

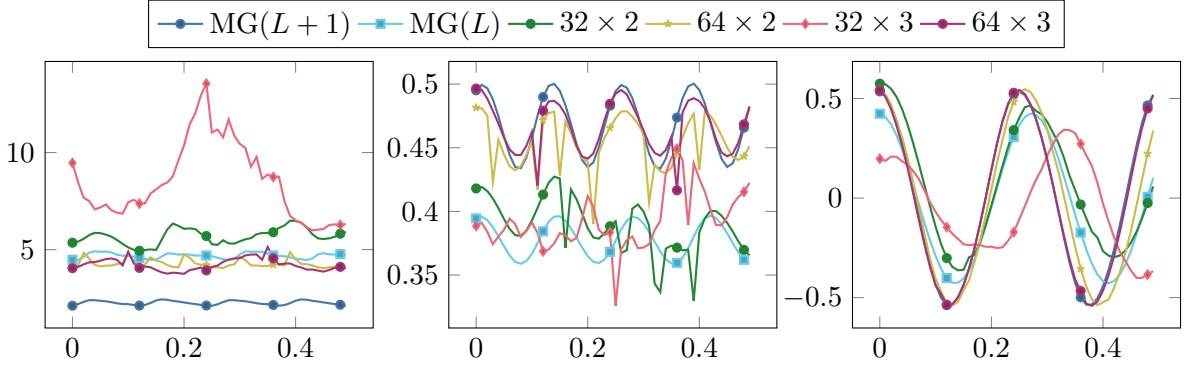


Figure 2: Divergence $J_{\text{div}}(v(t))$, drag $J_{\text{drag}}(v(t), p(t))$, lift $J_{\text{lift}}(v(t), p(t))$ (from left to right) for the coarse multigrid solution $\text{MG}(L)$, a high fidelity finite element reference solution $\text{MG}(L+1)$ and DNN-MG solutions for different GRU configurations.

able to recover the flow frequency of the high fidelity simulations and through this we obtained velocity and pressure errors not attainable before. The divergence is now also at least as good as for the standard multigrid simulation without any additional effort, cf. [12]. Importantly, the improved accuracy comes at virtually no additional cost in terms of runtime and are now able to improve over a standard multigrid simulation by almost 100%.

The presented results lead to further research directions we would like to investigate in the future. One question is when the network size saturates and how the required amount of training data scales with the number of network parameters. The presented results in 2D provide also a promising basis for a hybrid finite element / neural network simulation for the Navier-Stokes equations in 3D which we would like to consider in the future. Finally, our practical findings provide a strong motivation to address more theoretical questions such as the stability, consistency and convergence of DNN-MG.

Acknowledgement NM acknowledges support by the Helmholtz-Gesellschaft grant number HIDSS-0002 DASHH.

Conflict of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

type	Drag				Lift				Freq	Divergence	
	min	max	mean	amp	min	max	mean	amp		max $\nabla \cdot v$	mean $\nabla \cdot v$
MG($L+1$)	0.4332	0.5005	0.4668	0.0673	-0.5394	0.5358	-0.0018	1.0752	3.9823	2.4384	2.0343
MG	0.3588	0.3966	0.3777	0.0378	-0.4281	0.4242	-0.0019	0.8523	3.6530	5.6041	4.2733
32×1	0.3565	0.4505	0.4035	0.0941	-0.5101	0.4479	-0.0311	0.9580	3.6437	5.7798	4.5833
48×1	0.3697	0.4814	0.4256	0.1116	-0.5463	0.5362	-0.0051	1.0824	3.8627	10.5135	7.1490
64×1	0.3683	0.5176	0.4430	0.1493	-0.5380	0.5505	0.0062	1.0884	3.9301	12.5491	8.2422
32×2	0.3296	0.4273	0.3785	0.0977	-0.3725	0.6077	0.1176	0.9802	3.5533	8.0235	5.3722
48×2	0.3737	0.5029	0.4383	0.1292	-0.5352	0.5494	0.0071	1.0846	3.8462	10.2259	5.5768
64×2	0.3807	0.4827	0.4317	0.1020	-0.5444	0.5563	0.0060	1.1007	3.7559	5.3619	3.9398
32×3	0.3080	0.4604	0.3842	0.1525	-0.4108	0.3984	-0.0062	0.8092	3.2710	14.9148	7.0485
48×3	0.3813	0.4795	0.4304	0.0982	-0.4950	0.5019	0.0035	0.9969	3.8095	4.9454	3.3824
64×3	0.3648	0.4965	0.4306	0.1317	-0.5466	0.5455	-0.0006	1.0921	3.9130	7.8864	3.9800

Table 3: Maximum, minimum values, mean and amplitude of oscillation for drag and lift functionals, mean and maximum values of the divergence as well as frequency of the periodic solution. We indicate the results for the different finite element solution on coarse and fine mesh finite element simulation and for the different hybrid finite element / deep neural network approaches.

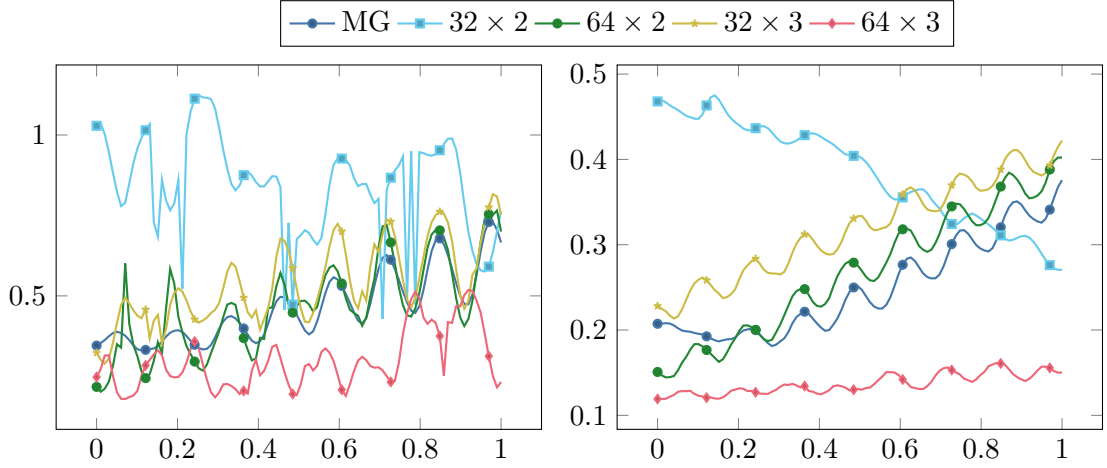


Figure 3: Relative errors for pressure (left) and velocity (right) compared to the reference solution on level $L+1$ for the coarse solution on level L and DNN-MG with different neural network configurations.

References

- [1] R. Becker and M. Braack. “A Finite Element Pressure Gradient Stabilization for the Stokes Equations Based on Local Projections”. In: *Calcolo* 38.4 (2001), pp. 173–199.
- [2] R. Becker et al. *The Finite Element Toolkit* GASCOIGNE 3D.
- [3] B. Chang et al. “Reversible Architectures for Arbitrarily Deep Residual Neural Networks”. In: *AAAI Conference on Artificial Intelligence*. 2018.
- [4] W. E. “A Proposal on Machine Learning via Dynamical Systems”. In: *Communications in Mathematics and Statistics* 5.1 (2017), pp. 1–11. URL: <https://doi.org/10.1007/s40304-017-0103-z>.
- [5] E. Haber and L. Ruthotto. “Stable architectures for deep neural networks”. In: *Inverse Problems* 34.1 (Dec. 2017), p. 014004. URL: <https://doi.org/10.1088%2F1361-6420%2Faa9a90>.
- [6] J. G. Heywood, R. Rannacher, and S. Turek. “Artificial Boundaries and Flux and Pressure Conditions for the Incompressible Navier-Stokes Equations”. In: *Int. J. Numer. Math. Fluids*. 22 (1992), pp. 325–352.
- [7] M. F. Kasim et al. “Up to two billion times acceleration of scientific simulations with deep neural architecture search”. In: (Jan. 2020). eprint: 2001.08055.
- [8] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836. URL: <http://www.nature.com/articles/nature14539>.
- [9] Z. Li et al. *Fourier Neural Operator for Parametric Partial Differential Equations*. 2020. arXiv: 2010.08895 [cs.LG].
- [10] Z. Li et al. *Neural Operator: Graph Kernel Network for Partial Differential Equations*. 2020. arXiv: 2003.03485 [cs.LG].
- [11] L. Lu, P. Jin, and G. E. Karniadakis. *DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators*. 2020. arXiv: 1910.03193 [cs.LG].
- [12] Nils Margenberg, Christian Lessig, and Thomas Richter. “Structure Preservation for the Deep Neural Network Multigrid Solver”. In: *etna* 56 (2021), pp. 86–101. ISSN: 1068-9613, 1068-9613. DOI: 10.1553/etna_vol156s86.
- [13] Nils Margenberg et al. “A Neural Network Multigrid Solver for the Navier-Stokes Equations”. In: *Journal of Computational Physics* 460 (July 1, 2022), p. 110983. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2022.110983.
- [14] Adam Paszke et al. “Automatic Differentiation in PyTorch”. In: *NIPS Autodiff Workshop*. 2017.
- [15] M. Raissi and G. E. Karniadakis. “Hidden physics models: Machine learning of nonlinear partial differential equations”. In: *Journal of Computational Physics* 357 (Mar. 2018), pp. 125–141. ISSN: 0021-9991. URL: <https://www.sciencedirect.com/science/article/pii/S0021999117309014>.
- [16] M. Raissi, P. Perdikaris, and G.E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. URL: <http://www.sciencedirect.com/science/article/pii/S0021999118307125>.

- [17] T. Richter. *Fluid-Structure Interactions. Models, Analysis and Finite Elements*. Vol. 118. Lecture Notes in Computational Science and Engineering. Springer, 2017.
- [18] Thomas Richter and Nils Margenberg. “Parallel Time-Stepping for Fluid-Structure Interactions”. In: *Mathematical Modelling of Natural Phenomena* (Jan. 14, 2021). ISSN: 0973-5348, 1760-6101.
- [19] M. Schäfer and S. Turek. “Benchmark computations of laminar flow around a cylinder”. In: *Flow Simulation with High-Performance Computers II*. Ed. by E.H. Hirschel. Notes Numer. Fluid Mech. 52. Vieweg, Wiesbaden, 1996, pp. 547–566.