DNN-MG: Neural network multigrid solver for the Navier-Stokes equations

Christian Lessig[2]

(with Nils Margenberg[1] Robert Jendersie[2] Dirk Hartmann[3] Thomas Richter[2])

[1]Helmut-Schmidt-University Hamburg, [2]Otto-von-Guericke-University Magdeburg, [3]Siemens AG

December 13, 2021

Motivation

- Navier-Stokes equations are central for many applications (atmospheric and ocean dynamics, aerospace engineering, process engineering, ...)
- Existing numerical methods have matured over decades but still very large costs for highly resolved simulations
  - $\approx 10^{11}$ degrees of freedom to resolve flow around airfoil, even more for atmospheric dynamics
- **Can we combine existing numerical methods and machine learning to retain the advantages of the former but overcome their shortcomings?**

Motivation

- Navier-Stokes equations are central for many applications (atmospheric and ocean dynamics, aerospace engineering, process engineering, ...)
- Existing numerical methods have matured over decades but still very large costs for highly resolved simulations
  - $\approx 10^{11}$ degrees of freedom to resolve flow around airfoil, even more for atmospheric dynamics
- **Can we combine existing numerical methods and machine learning to retain the advantages of the former but overcome their shortcomings?**

Motivation

- Navier-Stokes equations are central for many applications (atmospheric and ocean dynamics, aerospace engineering, process engineering, ...)
- Existing numerical methods have matured over decades but still very large costs for highly resolved simulations
  - $\approx 10^{11}$ degrees of freedom to resolve flow around airfoil, even more for atmospheric dynamics
- **Can we combine existing numerical methods and machine learning to retain the advantages of the former but overcome their shortcomings?**

Motivation

- Navier-Stokes equations are central for many applications (atmospheric and ocean dynamics, aerospace engineering, process engineering, ...)
- Existing numerical methods have matured over decades but still very large costs for highly resolved simulations
  - $\approx 10^{11}$ degrees of freedom to resolve flow around airfoil, even more for atmospheric dynamics
- **Can we combine existing numerical methods and machine learning to retain the advantages of the former but overcome their shortcomings?**

The incompressible Navier-Stokes equations

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

$$\nabla \cdot v = 0 \quad \text{on } [0, T] \times \Omega.$$

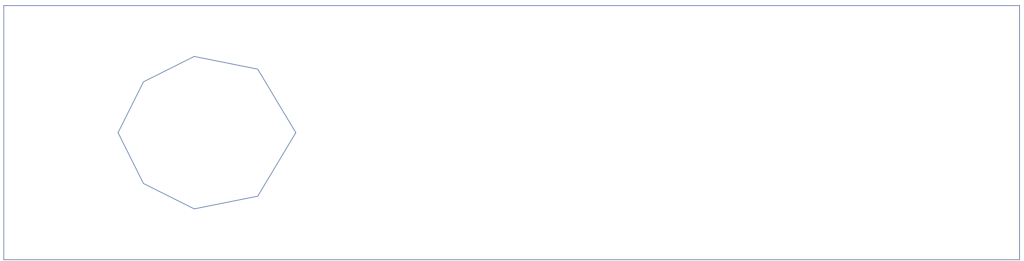
The incompressible Navier-Stokes equations

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

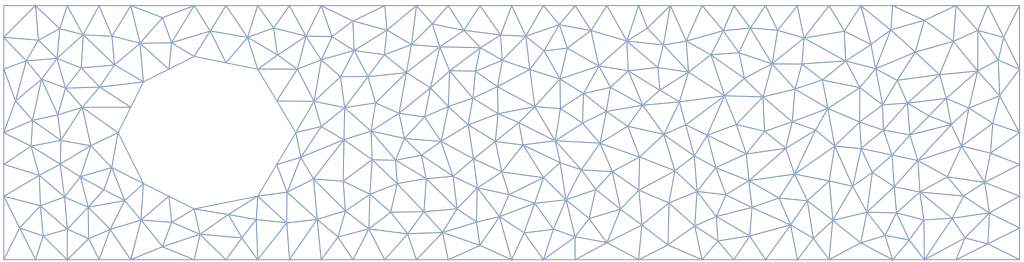
with initial and boundary conditions

$$v(0,\, \cdot) = v_0(\cdot) \quad \text{on } \Omega$$
$$v = v^D \quad \text{on } [0,\, T] \times \Gamma^D$$
$$\frac{1}{Re}(\vec{n} \cdot \nabla)v - p\vec{n} = 0 \quad \text{in } [0,\, T] \times \Gamma^N,$$

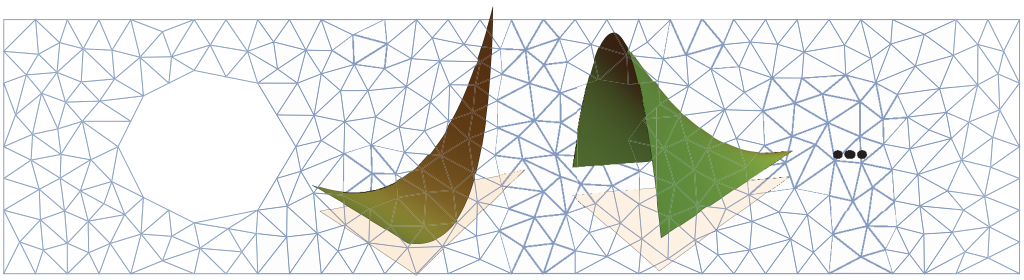
Discretization of incompressible Navier-Stokes equations



Discretization of incompressible Navier-Stokes equations



Discretization of incompressible Navier-Stokes equations



Discretization of incompressible Navier-Stokes equations

Finite element discretization via weak formulation:

$$(\partial_t v_h,\, \phi_h) + (v_h \cdot \nabla v_h,\, \phi_h) + \frac{1}{\mathrm{Re}}(\nabla v_h,\, \nabla \phi_h) - (p_h,\, \nabla \cdot \phi_h) = (f,\, \phi_h)$$

$$(\nabla \cdot v_h,\, \xi_h) + \sum_{T \in \Omega_h} \alpha_T (\nabla(p_h - \pi_h p_h), \nabla(\xi_h - \pi_h \xi_h)) = 0$$

for all $\phi_h \in V_h$, $\xi_h \in L_h$ where $V_h \subset H^1(\Omega)$ is the finite element test function space for the velocity (subject to the boundary conditions) and $L_h \subset L_2(\Omega)$ is those for the pressure.

Discretization of incompressible Navier-Stokes equations

Time discretization with second-order (implicit) Crank-Nicolson scheme:

$$\frac{1}{2}(2/k\, v_n + v_n \cdot \nabla v_n,\, \phi_h) + \frac{1}{2\mathrm{Re}}(\nabla v_n,\, \nabla \phi_h) - (p_n,\, \nabla \cdot \phi_h) = Rhs(v_{n-1}, f_{\{n,n-1\}}, \phi_h)$$

$$(\nabla \cdot v_n,\, \xi_h) + \sum_{T \in \Omega_h} \alpha_T (\nabla(p_n - \pi_h p_n), \nabla(\xi_h - \pi_h \xi_h)) = 0$$

for all $\phi_h \in V_h$, $\xi_h \in L_h$.

Discretization of incompressible Navier-Stokes equations

Time discretization with second order Crank-Nicolson scheme:

$$\frac{1}{2}(2/k\, v_n + v_n \cdot \nabla v_n,\, \phi_h) + \frac{1}{2\mathrm{Re}}(\nabla v_n,\, \nabla \phi_h) - (p_n,\, \nabla \cdot \phi_h) = Rhs(v_{n-1}, f_{\{n,n-1\}}, \phi_h)$$

$$(\nabla \cdot v_n,\, \xi_h) + \sum_{T \in \Omega_h} \alpha_T (\nabla(p_n - \pi_h p_n), \nabla(\xi_h - \pi_h \xi_h)) = 0$$

for all $\phi_h \in V_h$, $\xi_h \in L_h$. Implicit solve for unknown $x_n = (v_n, p_n)$.

Discretization of incompressible Navier-Stokes equations

Solve

$$\mathcal{A}_h(x_n) = Rhs(v_{n-1}, f_{\{n,n-1\}}, \phi_h)$$

for $x_n = (v_n, p_n)$ using Newton iteration

$$\mathcal{A}'_h(x^{(l-1)})w^{(l)} = Rhs(v_{n-1}, f_{\{n,n-1\}}, \phi_h) - \mathcal{A}_h(x^{(l-1)}), \quad x^{(l)} = x^{(l-1)} + w^{(l)}$$

for all $\phi_h \in V_h$, $\xi_h \in L_h$ where $\mathcal{A}'_h$ is the Jacobian of $\mathcal{A}_h$.

Discretization of incompressible Navier-Stokes equations

Solve

$$\mathcal{A}_h(x_n) = Rhs(v_{n-1}, f_{\{n,n-1\}}, \phi_h)$$

for $x_n = (v_n, p_n)$ using Newton iteration

$$\mathcal{A}'_h(x^{(l-1)})w^{(l)} = Rhs(v_{n-1}, f_{\{n,n-1\}}, \phi_h) - \mathcal{A}_h(x^{(l-1)}), \quad x^{(l)} = x^{(l-1)} + w^{(l)}$$

for all $\phi_h \in V_h$, $\xi_h \in L_h$ where $\mathcal{A}'_h$ is the Jacobian of $\mathcal{A}_h$. At each Newton step a linear system needs to be solved.

Discretization of incompressible Navier-Stokes equations

Solve

$$\mathcal{A}_h(x_n) = Rhs(v_{n-1}, f_{\{n,n-1\}}, \phi_h)$$

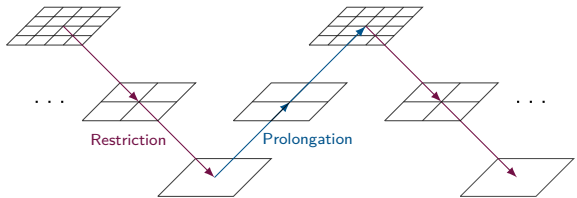
for $x_n = (v_n, p_n)$ using Newton iteration

$$\mathcal{A}'_h(x^{(l-1)})w^{(l)} = Rhs(v_{n-1}, f_{\{n,n-1\}}, \phi_h) - \mathcal{A}_h(x^{(l-1)}), \quad x^{(l)} = x^{(l-1)} + w^{(l)}$$

for all $\phi_h \in V_h$, $\xi_h \in L_h$ where $\mathcal{A}'_h$ is the Jacobian of $\mathcal{A}_h$. At each Newton step a linear system needs to be solved. GMRES with multi-grid pre-conditioner to efficiently obtain robust solution.

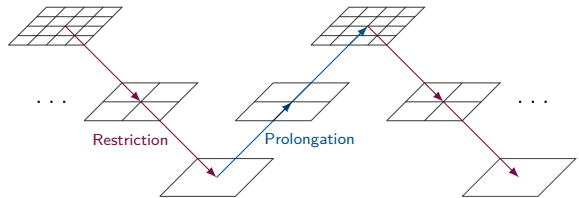
Discretization of incompressible Navier-Stokes equations

Multi-grid pre-conditioner:



Discretization of incompressible Navier-Stokes equations

Multi-grid pre-conditioner:



Solve linear system on a fine mesh level by restricting (projecting) it to coarser and coarser levels, solving directly on the coarsest one, and then prolongating (interpolating) solution back across the levels.

Discretization of incompressible Navier-Stokes equations

For each time step:

Discretization of incompressible Navier-Stokes equations

For each time step:

– Newton solve for $x_n = (v_n, p_n)$; for each Newton step:

Discretization of incompressible Navier-Stokes equations

For each time step:

– Newton solve for $x_n = (v_n, p_n)$; for each Newton step:

  – Solve linear system with GMRES; for each GMRES step:

Discretization of incompressible Navier-Stokes equations

For each time step:

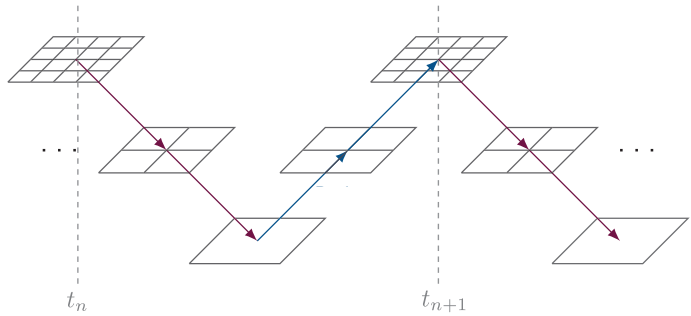
- Newton solve for $x_n = (v_n, p_n)$; for each Newton step:
  - Solve linear system with GMRES; for each GMRES step:
    - Use one sweep of geometric multi-grid as pre-conditioner

Discretization of incompressible Navier-Stokes equations

For each time step:

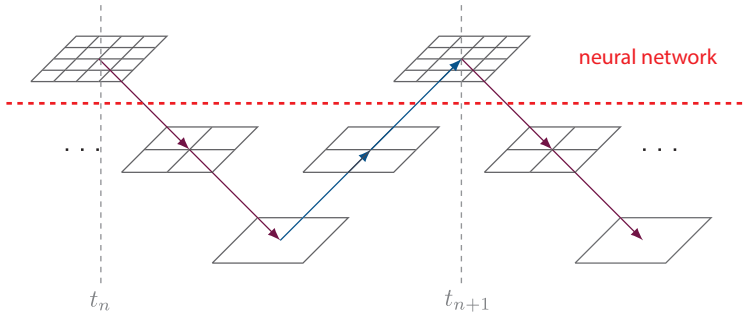
- Newton solve for $x_n = (v_n, p_n)$; for each Newton step:
  - Solve linear system with GMRES; for each GMRES step:
    - Use one sweep of geometric multi-grid as pre-conditioner
- **Correct v_n using neural network and use in next $RHS(v_n, f_{\{n,n-1\}})$**

Idea of deep neural network multigrid solver



– Perform classical computations up to level L

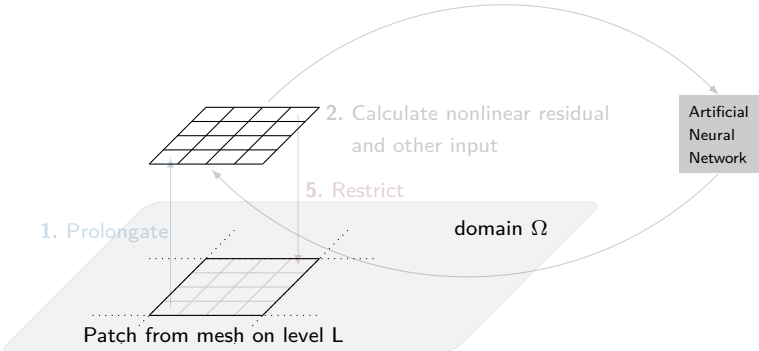
Idea of deep neural network multigrid solver



– Perform classical computations up to level L resulting in $(\tilde{v}_n, \tilde{p}_n)$

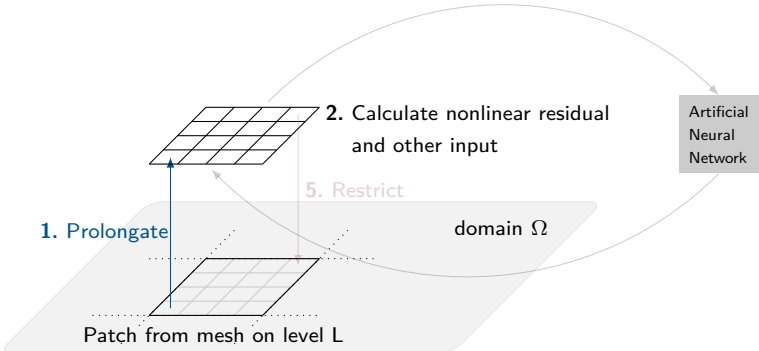
– Neural network computes correction $\hat{v}_n$ using mesh level $L+1$ s.t. $v_n = \tilde{v}_n + \hat{v}_n$

Prediction with the neural net



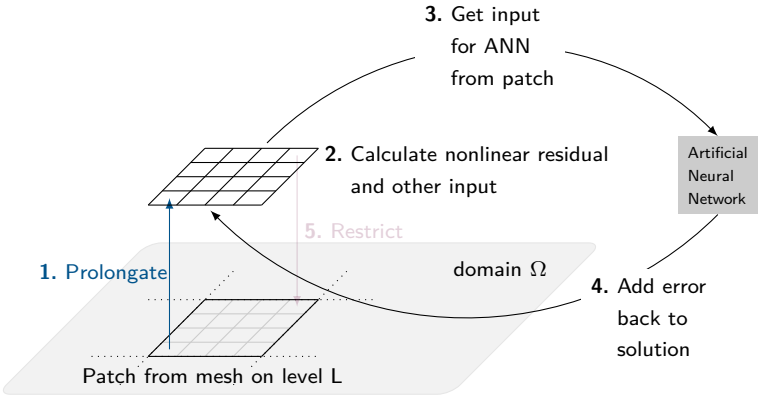
- Prediction is *local*, independent for patches on level L

Prediction with the neural net



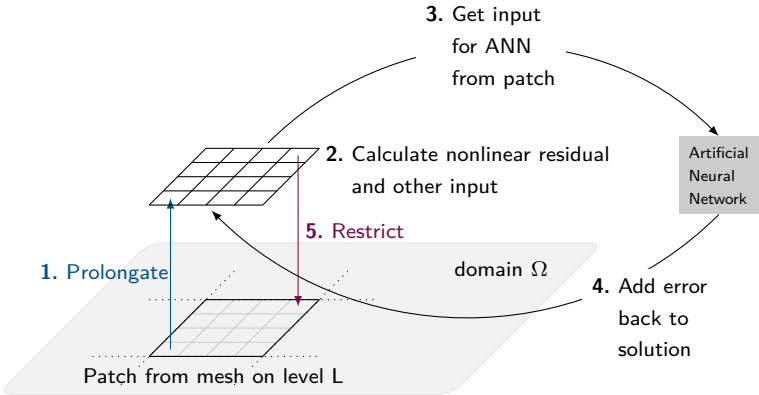
- Prediction is *local*, independent for patches on level L

Prediction with the neural net



- Prediction is *local*, independent for patches on level L

Prediction with the neural net



- Prediction is *local*, independent for patches on level L

Network input and output

Input

- Nonlinear residual of the velocity on the patch
- Peclet number
- Prolongated velocity field $P(\tilde{v}_L)$
- Mesh information (element size, aspect ratio, ...)

Output

- Velocity correction $\hat{v}_{L+1}$ for the patch (overlaps are averaged)

Network input and output

Input

- Nonlinear residual of the velocity on the patch
- Peclet number
- Prolongated velocity field $P(\tilde{v}_L)$
- Mesh information (element size, aspect ratio, ...)

Output

- Velocity correction $\hat{v}_{L+1}$ for the patch (overlaps are averaged)

Network input and output

Input

- Nonlinear residual of the velocity on the patch
- Peclet number
- Prolongated velocity field $P(\tilde{v}_L)$
- Mesh information (element size, aspect ratio, ...)

Output

- Velocity correction $\hat{v}_{L+1}$ for the patch (overlaps are averaged)

Network input and output

Input

- Nonlinear residual of the velocity on the patch
- Peclet number
- Prolongated velocity field $P(\tilde{v}_L)$
- Mesh information (element size, aspect ratio, ...)

Output

- Velocity correction $\hat{v}_{L+1}$ for the patch (overlaps are averaged)

Network input and output

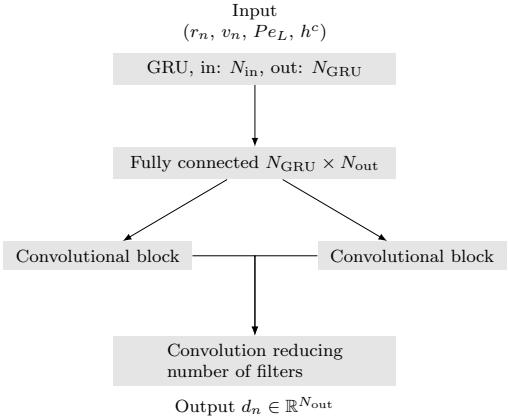
Input

- Nonlinear residual of the velocity on the patch
- Peclet number
- Prolongated velocity field $P(\tilde{v}_L)$
- Mesh information (element size, aspect ratio, ...)

Output

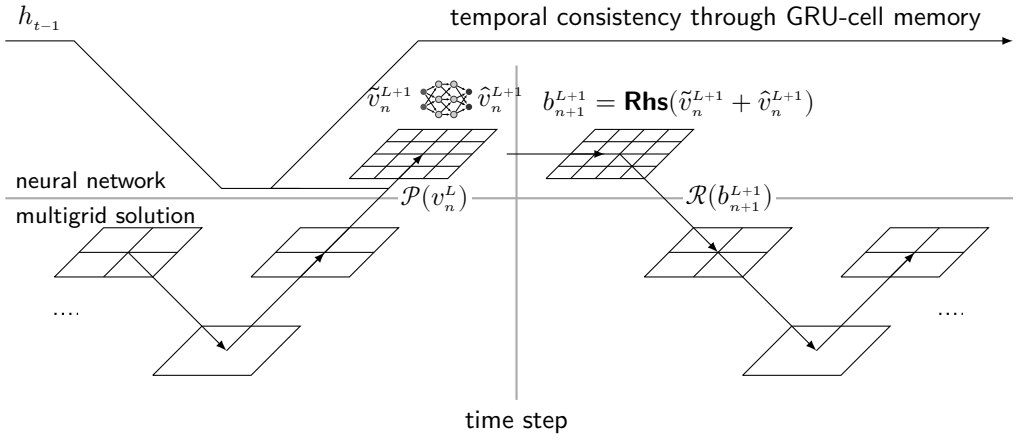
- Velocity correction $\hat{v}_{L+1}$ for the patch (overlaps are averaged)

Network architecture

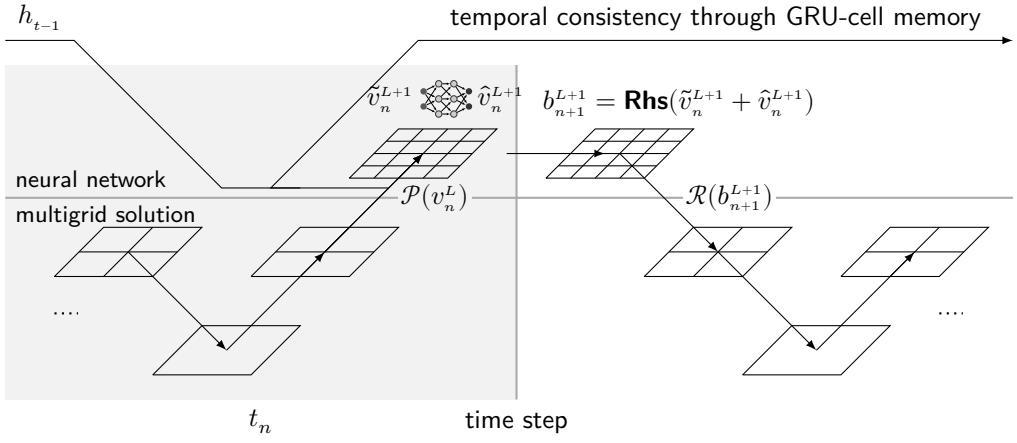


In total 8634 trainable parameters (Up to $\approx 80,000$ improves results and consistency)

Integration with the time stepping

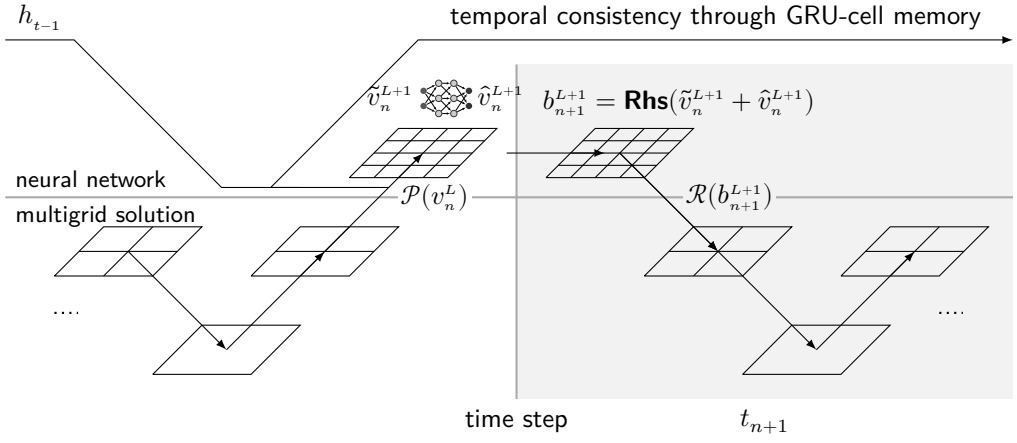


Integration with the time stepping



t_n: Prolongate numerical solution, predict error

Integration with the time stepping

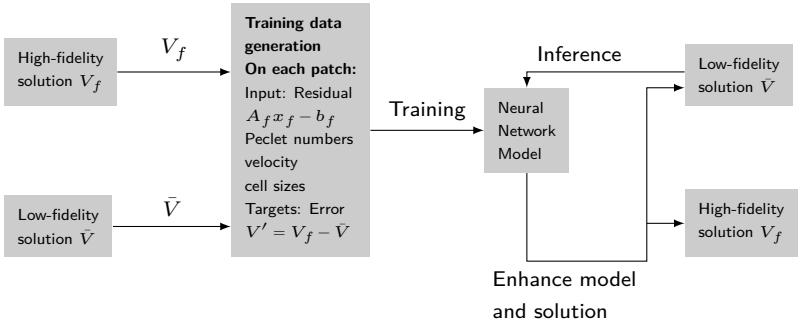


t_{n+1}: Assemble an improved RHS on fine level and restrict back to coarse level

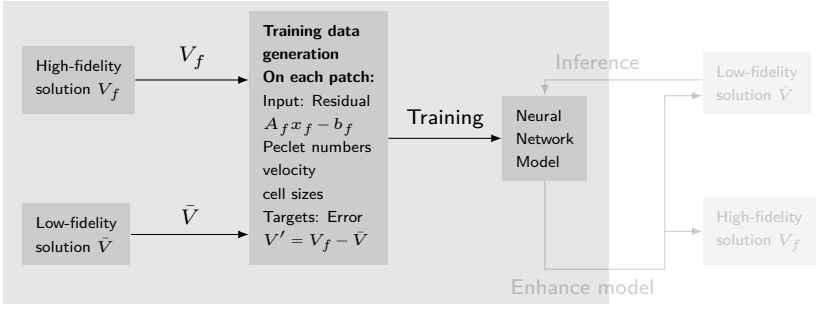
DNN-MG algorithm

for all time steps n **do**
  while not converged **do** ▷ Newton-GMRES method for Eq. 6
    $\delta z_i \leftarrow \text{MULTIGRID}(L, A_L^n, b_L^n, \delta z_i)$ ▷ Algo. 1 as preconditioner
    $z_{i+1} \leftarrow z_i + \epsilon\, \delta z_i$
  end while
  $\tilde{v}_n^{L+1} \leftarrow \mathcal{P}(v_n^L)$ ▷ Prolongation on level $L+1$
  $d_n^{L+1} \leftarrow \mathcal{N}(\tilde{v}_n^{L+1}, \Omega_L, \Omega_{L+1})$ ▷ Prediction of velocity correction
  $b_{n+1}^{L+1} \leftarrow \mathbf{Rhs}(\tilde{v}_n^{L+1} + d_n^{L+1}, f_n, f_{n+1})$ ▷ Set up rhs of Eq. 6 for next time step
  $b_{n+1}^{L} \leftarrow \mathcal{R}(b_{n+1}^{L+1})$ ▷ Restriction of rhs to level L
end for

Neural network of DNN-MG

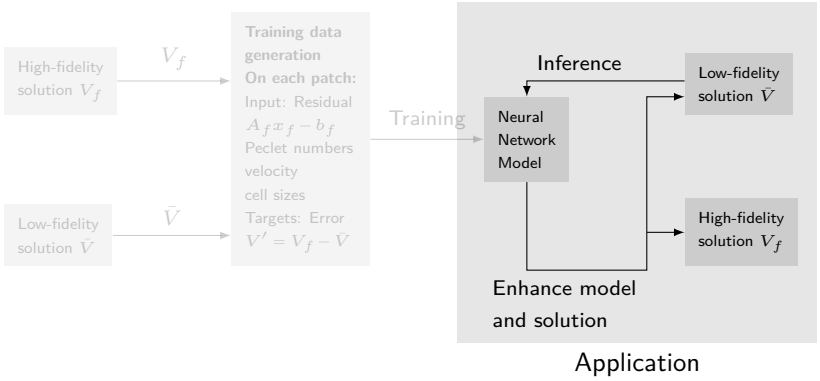


Neural network of DNN-MG

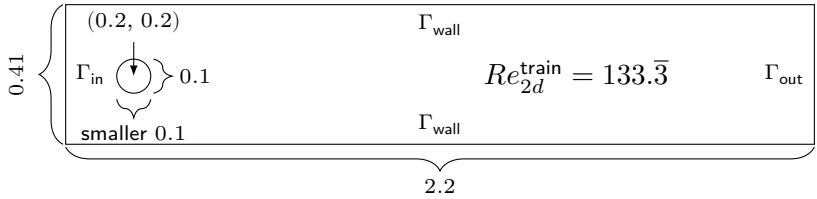


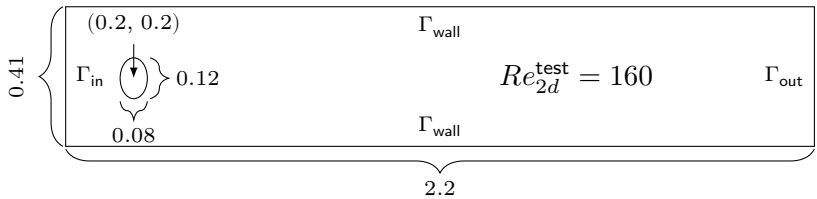
Training phase

Neural network of DNN-MG

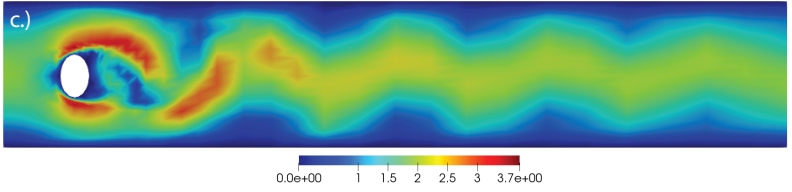


Training and test setup



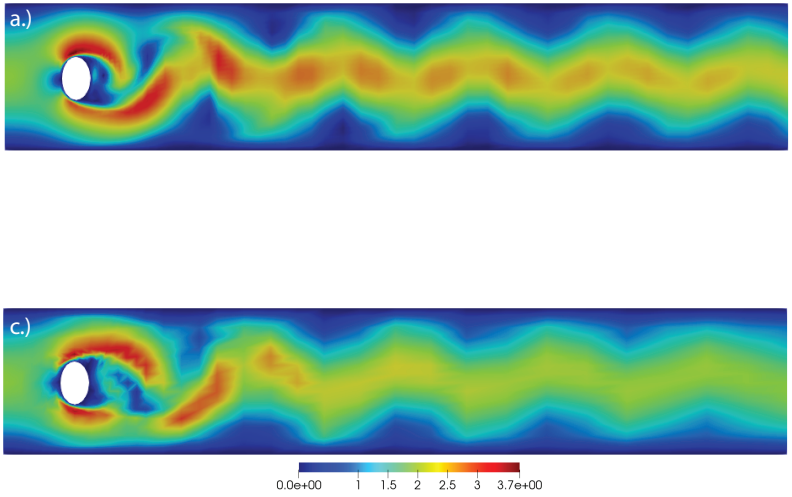


Results: velocity fields



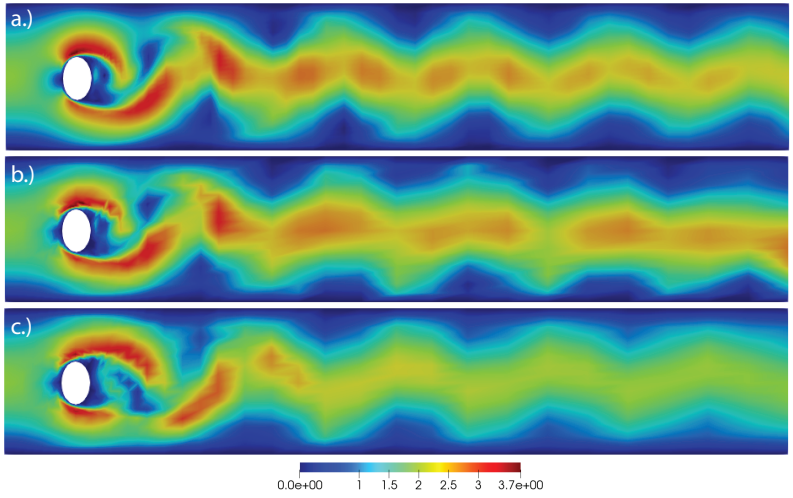
(a) multigrid solution on $L+1$ levels, (b) DNN-MG, (c) multigrid solution on L levels

Results: velocity fields



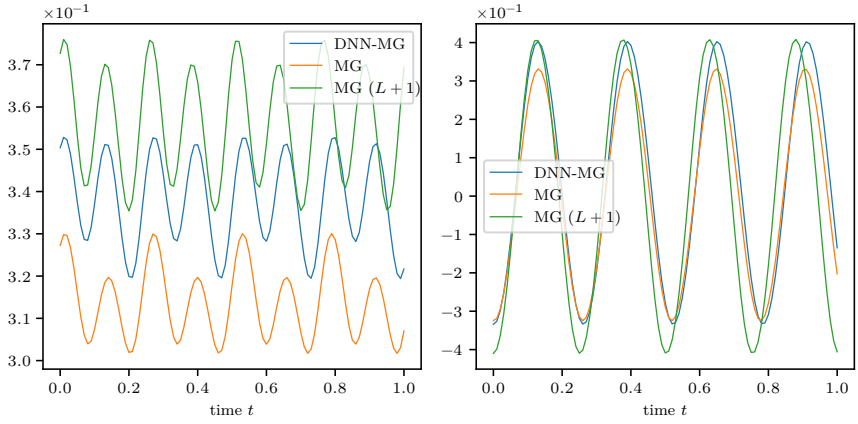
(a) multigrid solution on $L+1$ levels, (b) DNN-MG, (c) multigrid solution on L levels

Results: velocity fields

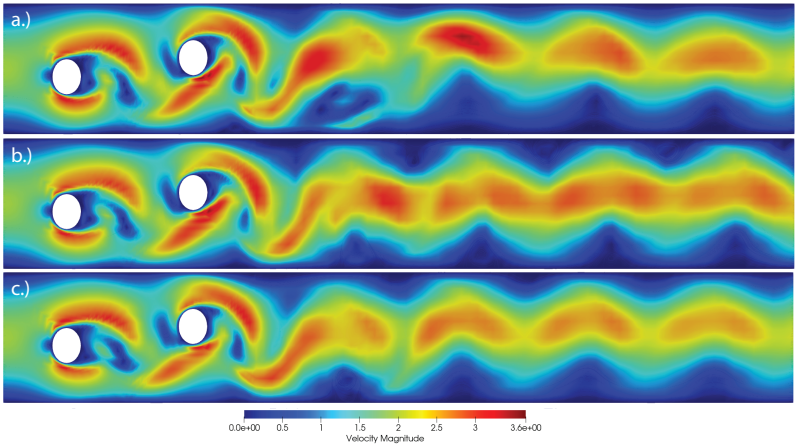


(a) multigrid solution on $L+1$ levels, (b) DNN-MG, (c) multigrid solution on L levels

Drag and lift functionals

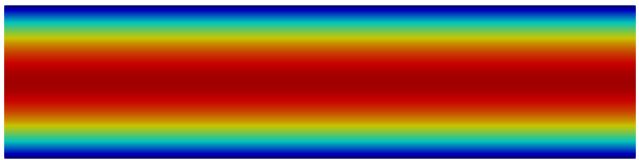


Generalization



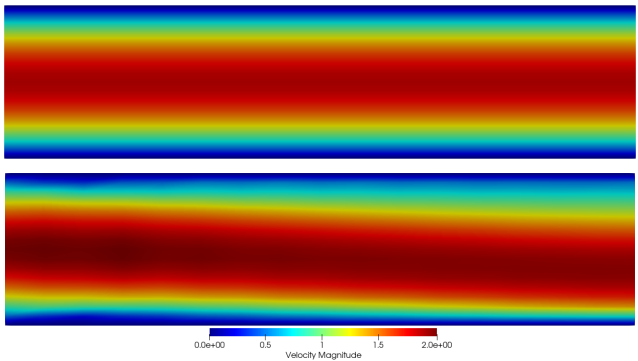
(a) multigrid solution on $L+1$ levels, (b) DNN-MG, (c) multigrid solution on L levels

Generalization



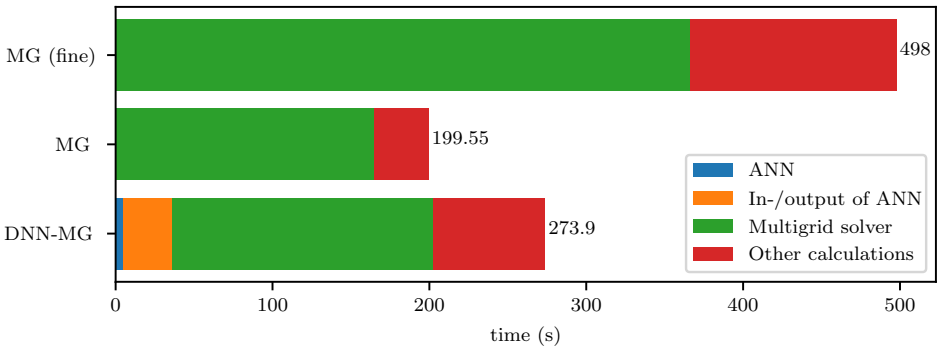
top: multigrid solution on $L+1$ levels, bottom: DNN-MG

Generalization

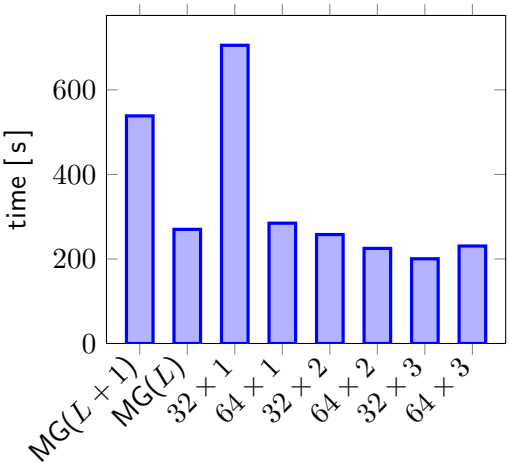


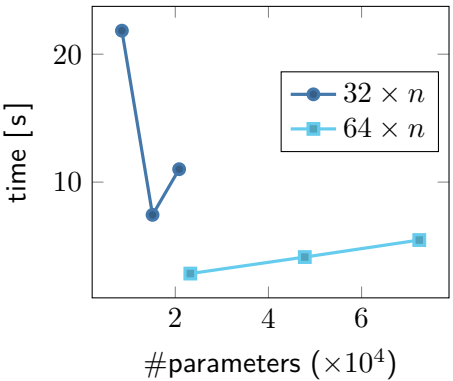
top: multigrid solution on $L+1$ levels, bottom: DNN-MG

Computation time



Simulation time





- Simulation time (left) and time spent on network evaluation

Results

- DNN-MG can improve coarse solution
- DNN-MG saves time compared to fine mesh solution by using prior knowledge
- Local approach enables application of one network to different domains
  - Efficient evaluation, simple training data generation, effective generalization
- DNN-MG generalizes well to other situations
- Applicable to general domains and boundary conditions

Results

- DNN-MG can improve coarse solution
- DNN-MG saves time compared to fine mesh solution by using prior knowledge
- Local approach enables application of one network to different domains
  - Efficient evaluation, simple training data generation, effective generalization
- DNN-MG generalizes well to other situations
- Applicable to general domains and boundary conditions

Results

- DNN-MG can improve coarse solution
- DNN-MG saves time compared to fine mesh solution by using prior knowledge
- Local approach enables application of one network to different domains
  - Efficient evaluation, simple training data generation, effective generalization
- DNN-MG generalizes well to other situations
- Applicable to general domains and boundary conditions

Results

- DNN-MG can improve coarse solution
- DNN-MG saves time compared to fine mesh solution by using prior knowledge
- Local approach enables application of one network to different domains
  - Efficient evaluation, simple training data generation, effective generalization
- DNN-MG generalizes well to other situations
- Applicable to general domains and boundary conditions

Results

- DNN-MG can improve coarse solution
- DNN-MG saves time compared to fine mesh solution by using prior knowledge
- Local approach enables application of one network to different domains
  - Efficient evaluation, simple training data generation, effective generalization
- DNN-MG generalizes well to other situations
- Applicable to general domains and boundary conditions

Results

- DNN-MG can improve coarse solution
- DNN-MG saves time compared to fine mesh solution by using prior knowledge
- Local approach enables application of one network to different domains
  - Efficient evaluation, simple training data generation, effective generalization
- DNN-MG generalizes well to other situations
- Applicable to general domains and boundary conditions

Outlook

- Stability and approximation rate
- Extend DNN-MG to integrate better with the MG and FEM framework
  - Use Residual in the loss function to try unsupervised approaches
- Implement DNN-MG with a semi-supervised approach
  - Generate new data and retrain if necessary
- Test on 3d cases, more problems and other equations

Outlook

- Stability and approximation rate
- Extend DNN-MG to integrate better with the MG and FEM framework
  - Use Residual in the loss function to try unsupervised approaches
- Implement DNN-MG with a semi-supervised approach
  - Generate new data and retrain if necessary
- Test on 3d cases, more problems and other equations

Outlook

- Stability and approximation rate
- Extend DNN-MG to integrate better with the MG and FEM framework
  - Use Residual in the loss function to try unsupervised approaches
- Implement DNN-MG with a semi-supervised approach
  - Generate new data and retrain if necessary
- Test on 3d cases, more problems and other equations

Outlook

- Stability and approximation rate
- Extend DNN-MG to integrate better with the MG and FEM framework
  - Use Residual in the loss function to try unsupervised approaches
- Implement DNN-MG with a semi-supervised approach
  - Generate new data and retrain if necessary
- Test on 3d cases, more problems and other equations

Outlook

- Stability and approximation rate
- Extend DNN-MG to integrate better with the MG and FEM framework
  - Use Residual in the loss function to try unsupervised approaches
- Implement DNN-MG with a semi-supervised approach
  - Generate new data and retrain if necessary
- Test on 3d cases, more problems and other equations

Outlook

- Stability and approximation rate
- Extend DNN-MG to integrate better with the MG and FEM framework
  - Use Residual in the loss function to try unsupervised approaches
- Implement DNN-MG with a semi-supervised approach
  - Generate new data and retrain if necessary
- Test on 3d cases, more problems and other equations

Thank you!

- N. Margenberg, D. Hartmann, C. Lessig, and T. Richter. A neural network multigrid solver for the navier-stokes equations. Submitted to Journal of Computational Physics, 2020.
- N. Margenberg, R. Jendersie, T. Richter, and C. Lessig. Deep neural networks for geometric multigrid methods. In ECCOMAS 2021, 2021.
- N. Margenberg, C. Lessig, and T. Richter. Structure preservation for the deep neural network multigrid solver. Electronic Transactions of Numerical Analysis, 2020.