LEARNING FROM LARGE-SCALE COMPUTATIONAL FLUID DYNAMICS SIMULATIONS

VSC USER DAY

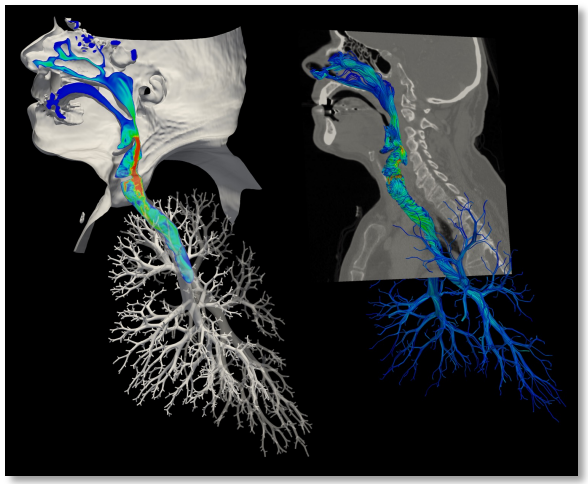
17.12.2025

ANDREAS LINTERMANN | JÜLICH SUPERCOMPUTING CENTRE | FORSCHUNGSZENTRUM JÜLICH GMBH

RE-THINKING SCIENTIFIC WORKFLOWS

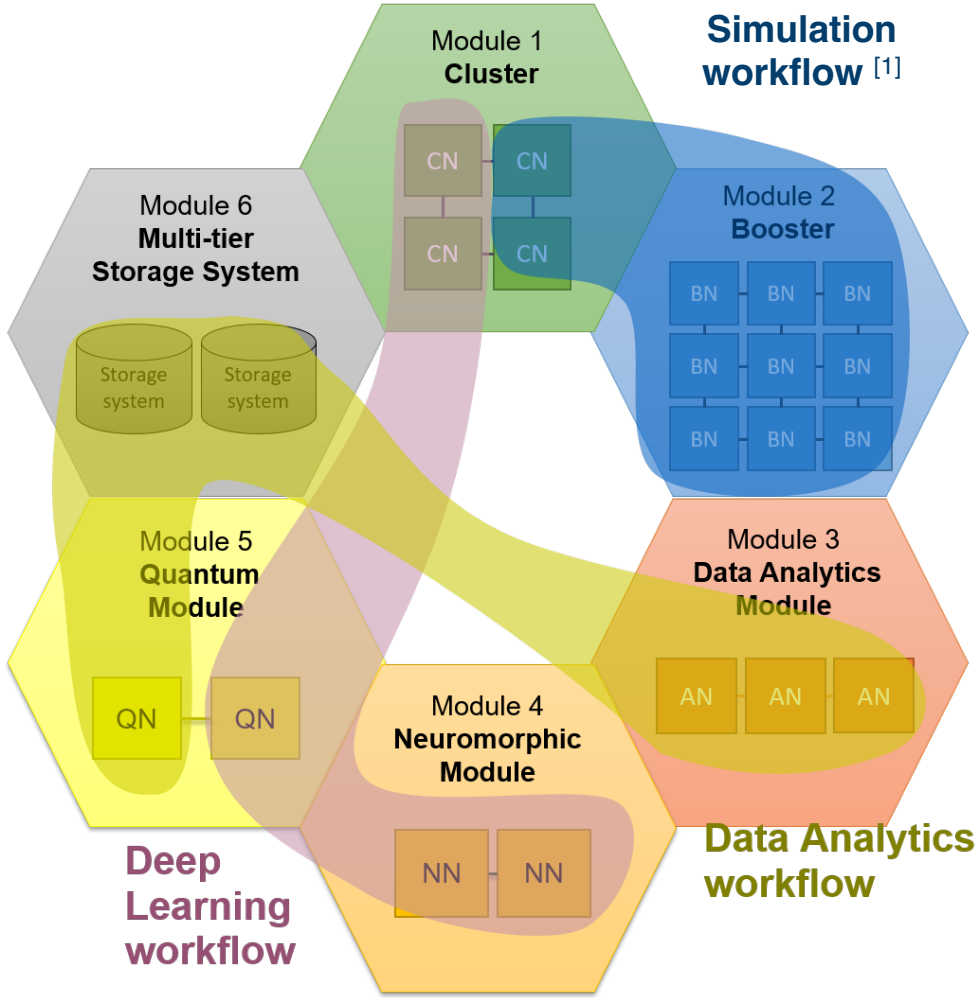
Gaps and opportunities of large-scale heterogeneous HPC

Complex hardware

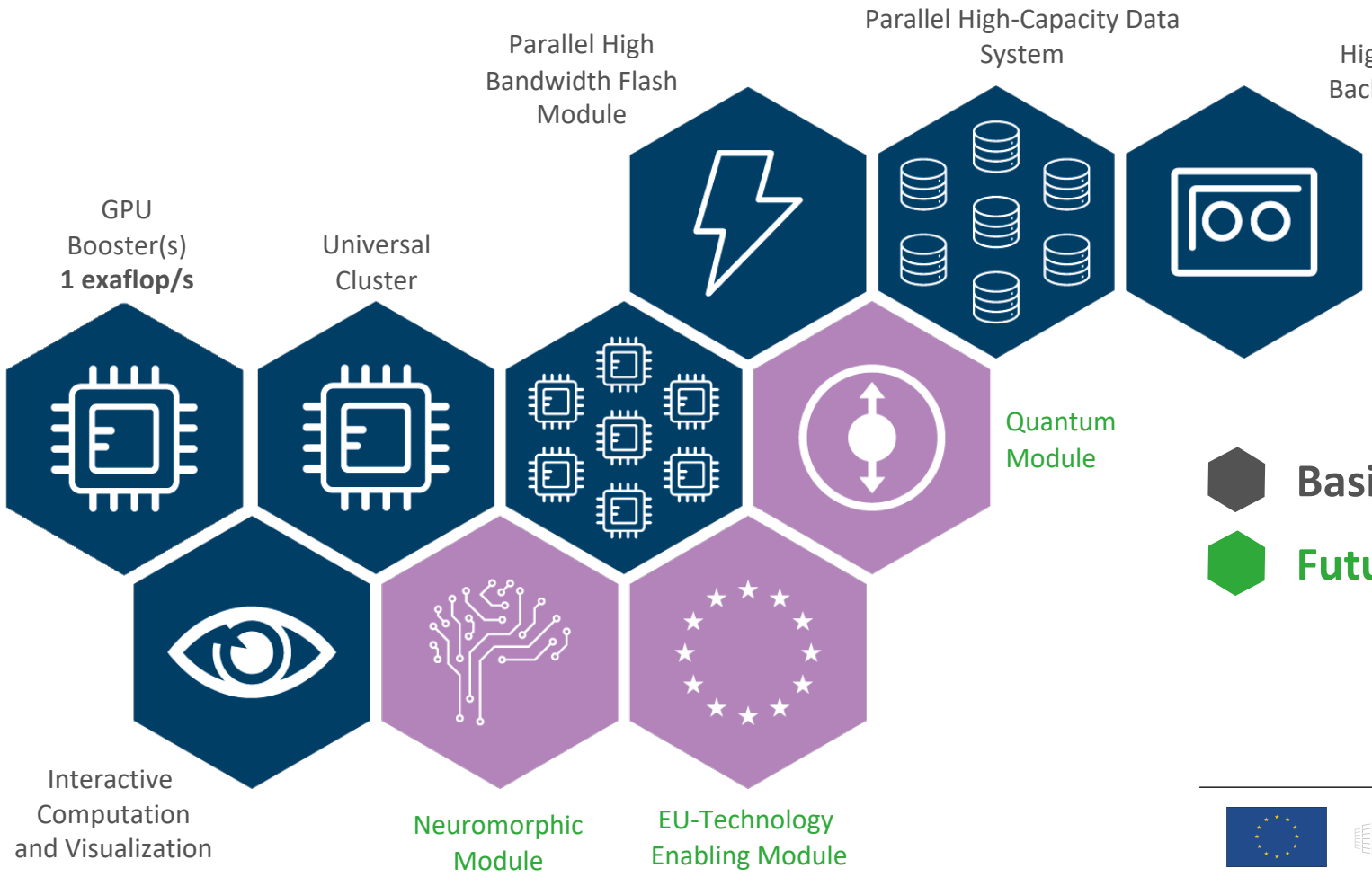


Complex tasks

- User's perspective:
  - Find the most suitable hardware for a specific task
  - Implement cost-effective scaling
  - Enable intertwined AI- and HPC-workflows
- HPC center's perspective:
  - Match the application diversity and allow for effective resource-sharing



JÜLICH Forschungszentrum



Basis Configuration

Future Technology Modules

Co-funded by

SPONSORED BY THE

EuroHPC Joint Undertaking

Federal Ministry of Education and Research

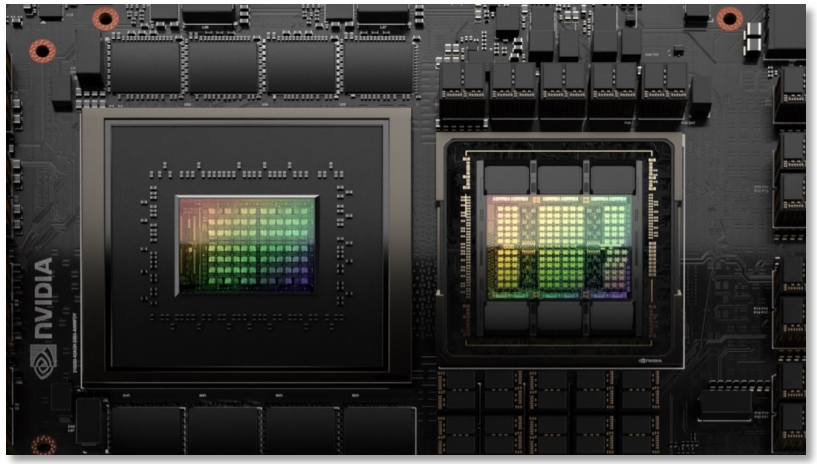
Ministerium für Kultur und Wissenschaft des Landes Nordrhein-Westfalen

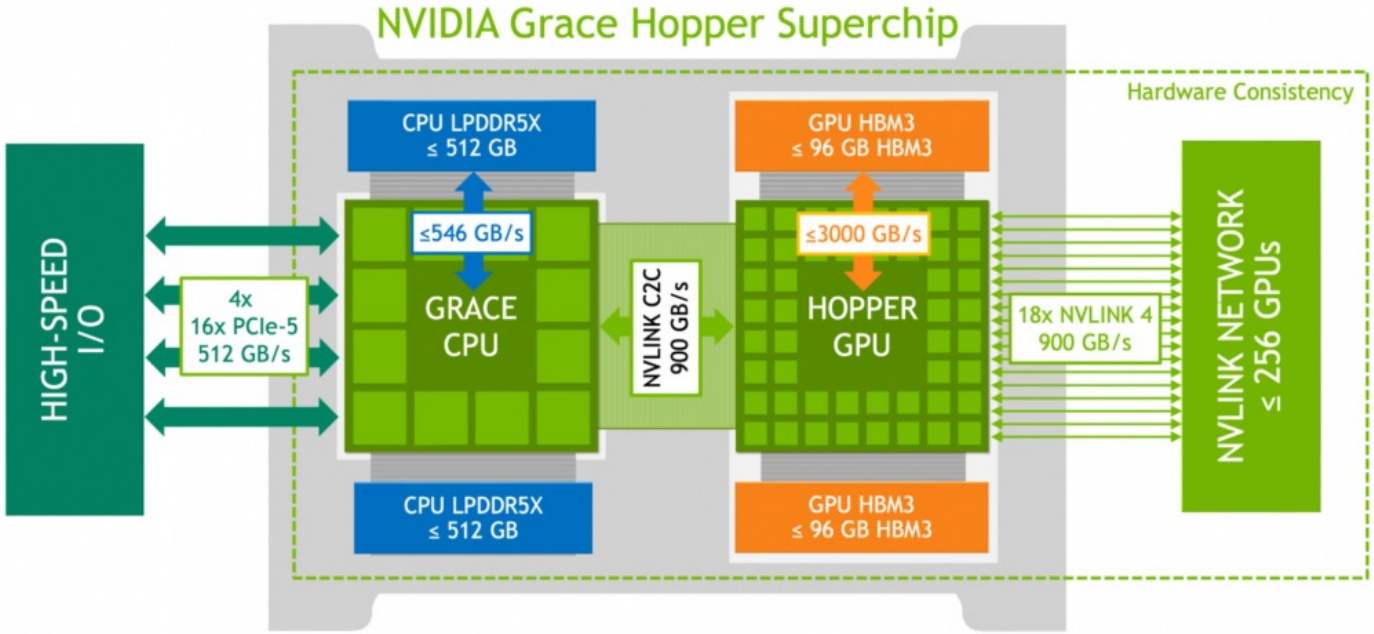
GCS Gauss Centre for Supercomputing

JUPITER - MODULAR ARCHITECTURE

JUPITER – THE BOOSTER

- **1 ExaFLOP/s (FP64, HPL, no. 4 Top500)**
- **NVIDIA Grace-Hopper GH200**
  - 4 chips per compute node
  - ~6,000 nodes
- **NVIDIA Mellanox NDR**
  - 4 x NDR200 NICs per compute node
- **BullSequana XH3000**
  - Direct Liquid Cooled blades
  - 2 compute node per blade





120 GB LPDDR5X

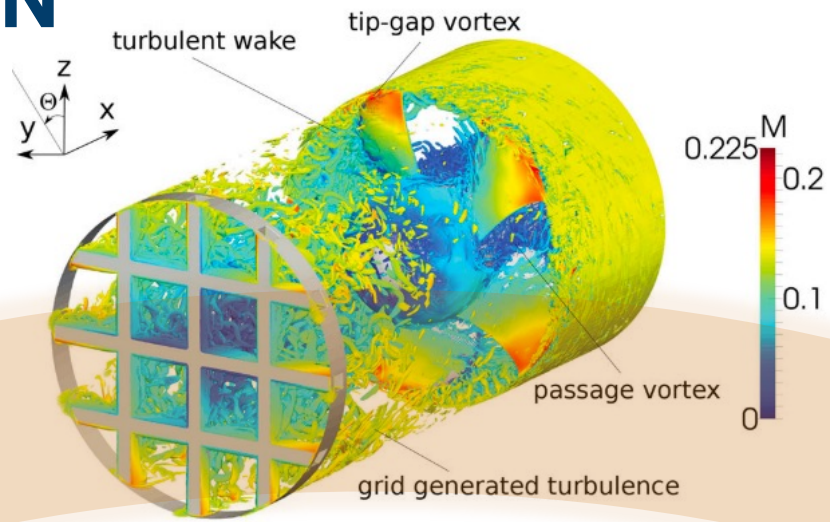
EVIDEN an atos business

NVIDIA

JÜLICH Forschungszentrum

CFD/AI INTEGRATION

Expensive DNS / LES



prediction

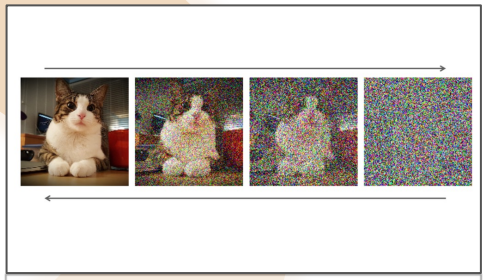
energy

performance

modeling

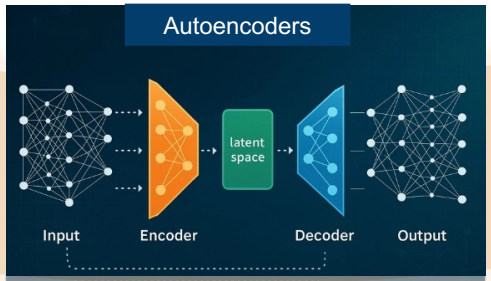
Transformers

Diffusion Models



Convergence of CFD + AI

Autoencoders

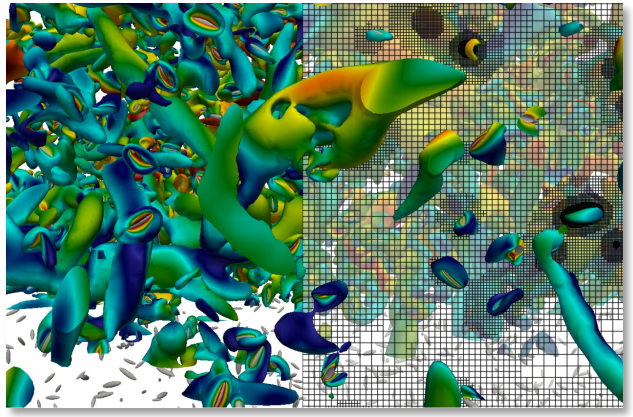


MULTI-PHYSICS SIMULATIONS WITH M-AIA

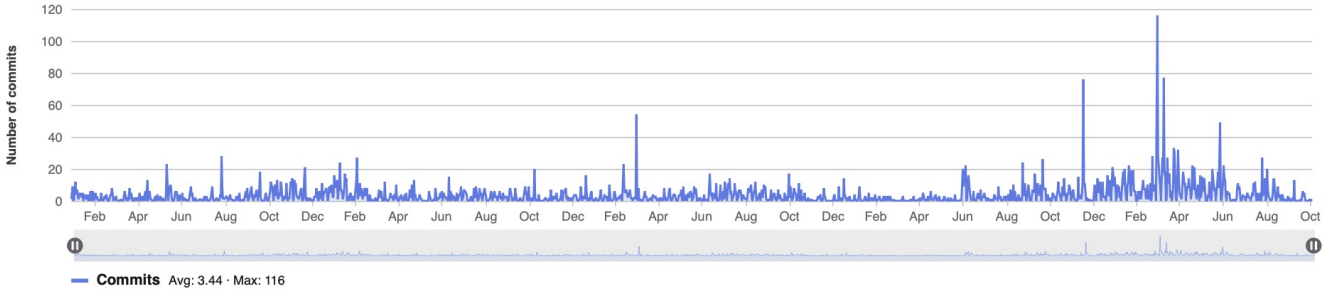
M-AIA

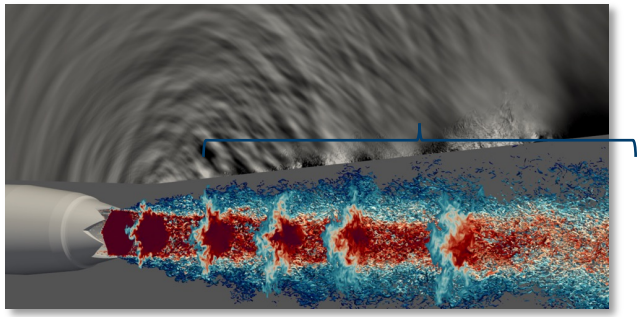
M-AIA (multiphysics Aerodynamisches Institut Aachen) [2]

- Open-source CFD / multi-physics simulation framework developed for +25 years
- Developed at RWTH (AIA) and Forschungszentrum Jülich (JSC)
- ~ 360k lines of code (C/C++, Markdown, Python, etc.) with > 50 active branches
- C++ / MPI / OpenMP / PSTL / HIP / parallel I/O

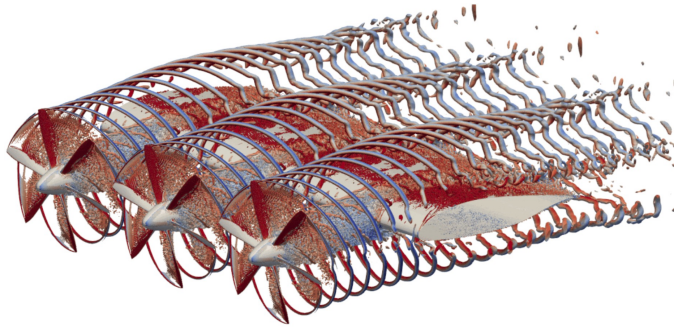


Fully-resolved particle simulation

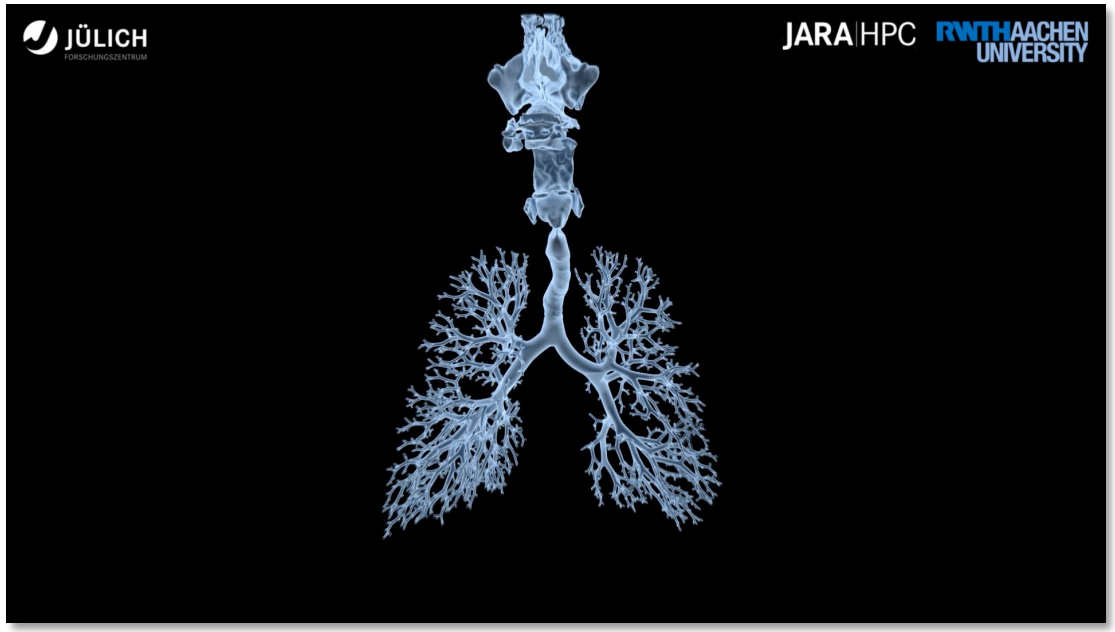




Jet noise (engine with chevrons)



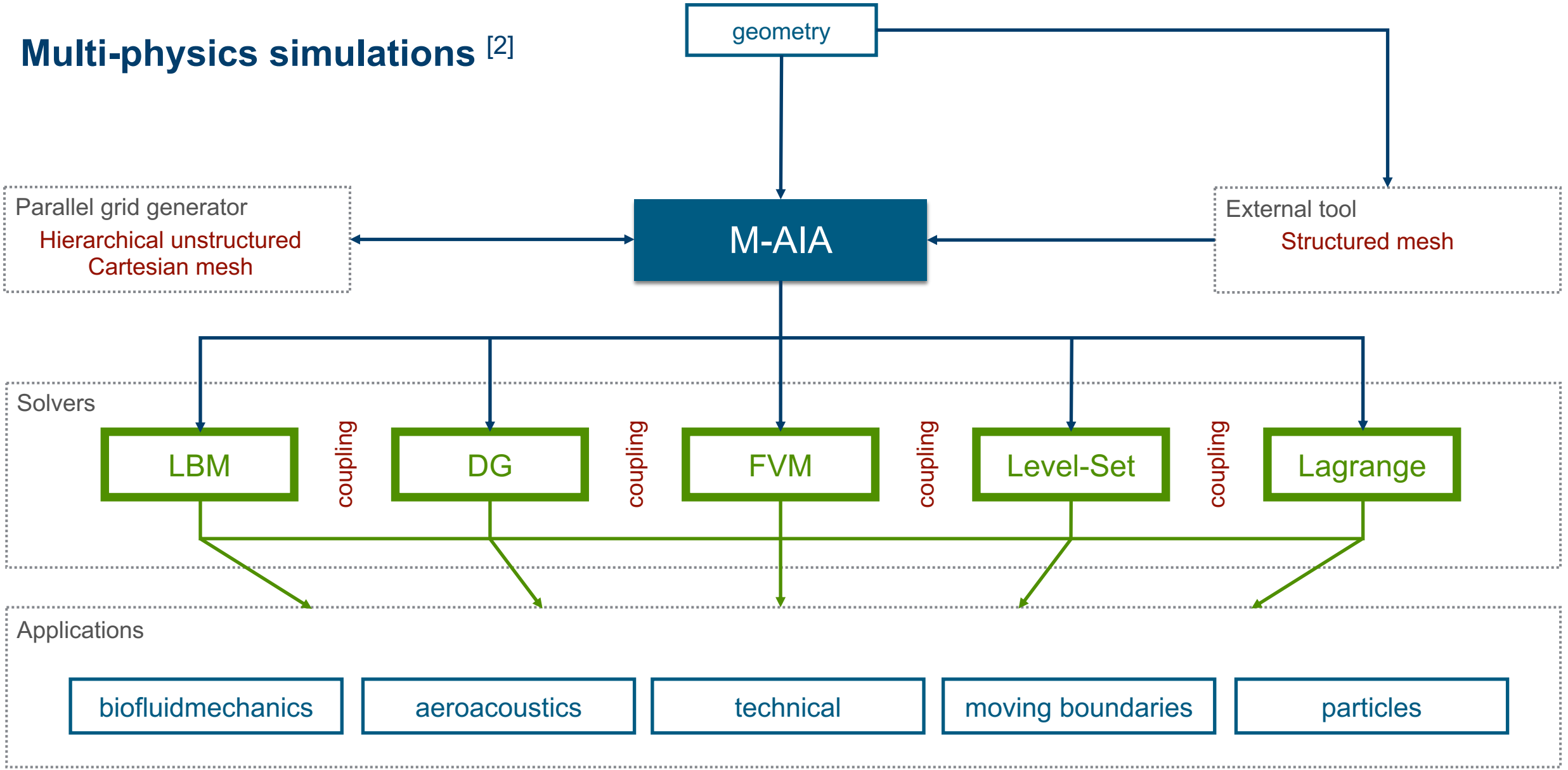
Propeller flow



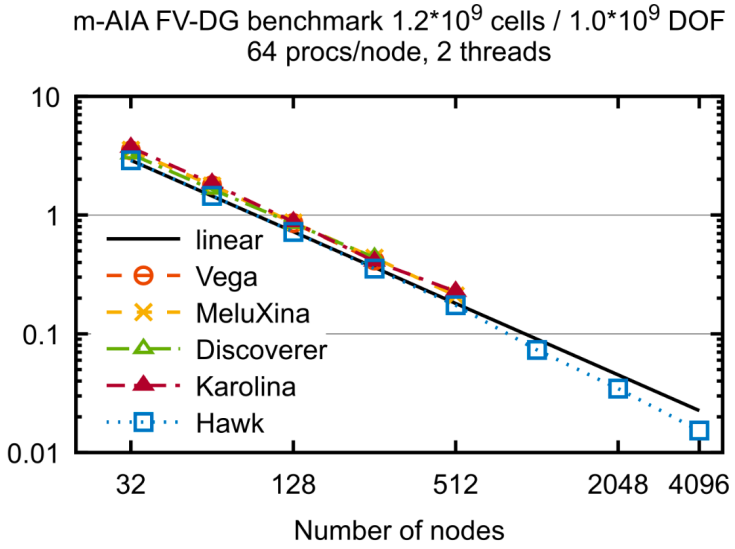
Particle flow in the human respiratory tract

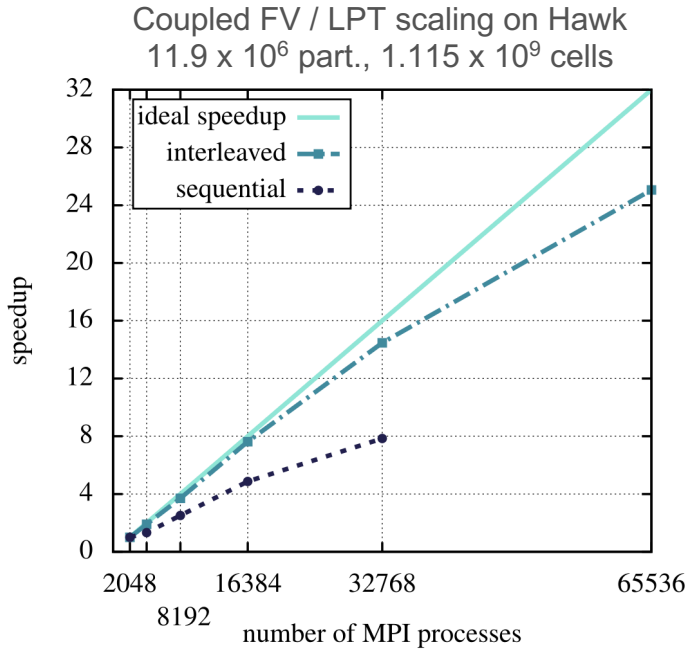
M-AIA

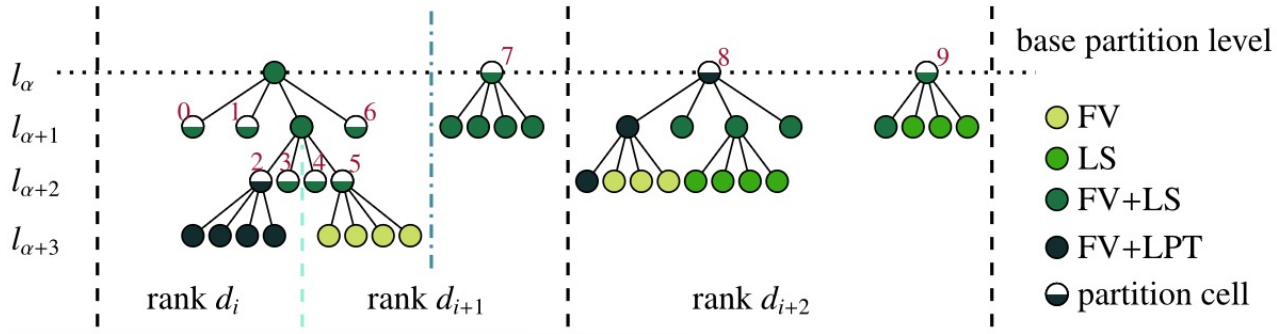
Multi-physics simulations [2]

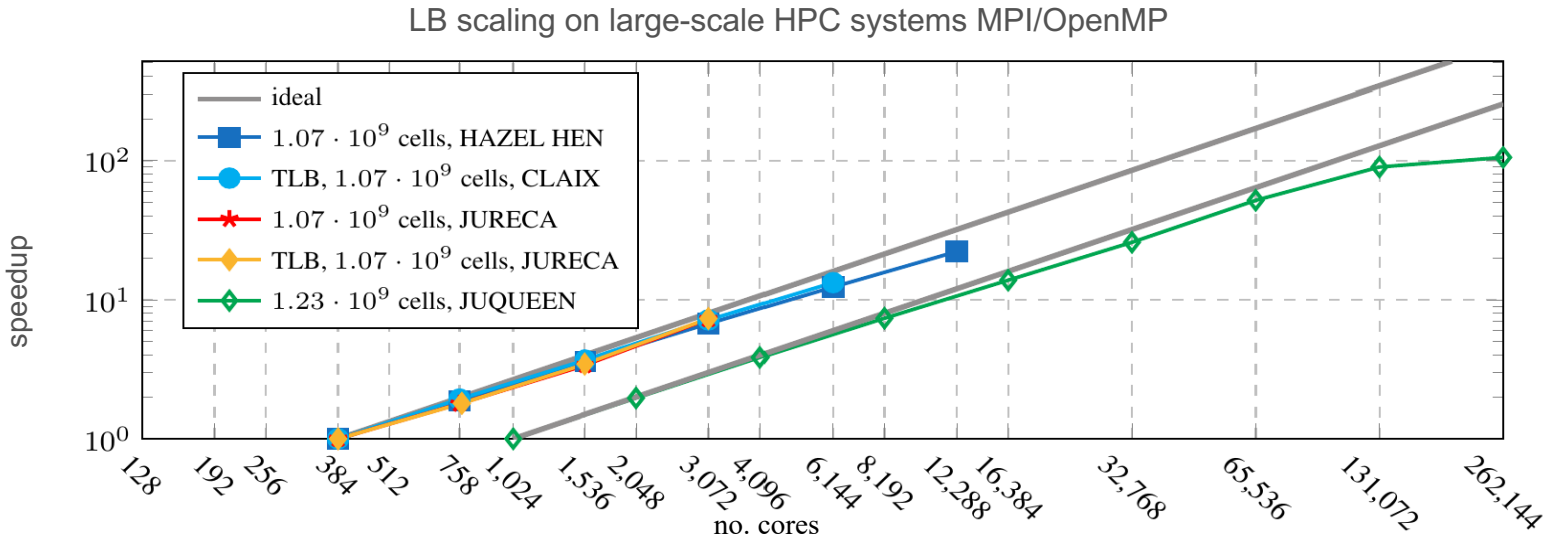


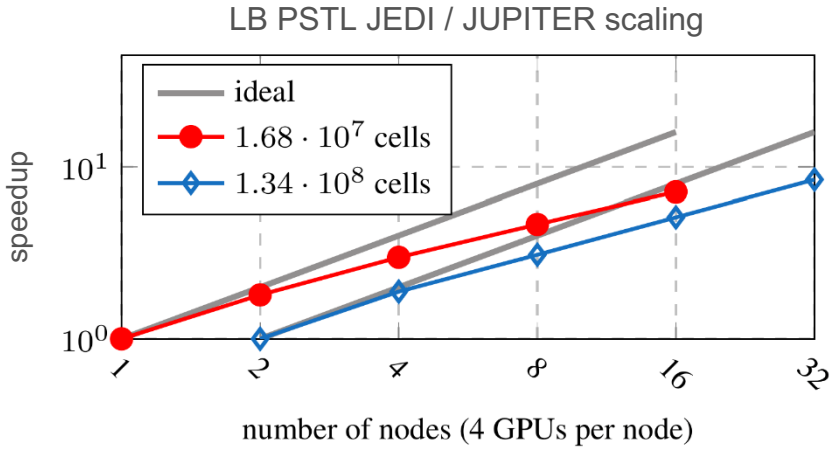
M-AIA SCALINGS [2,6]

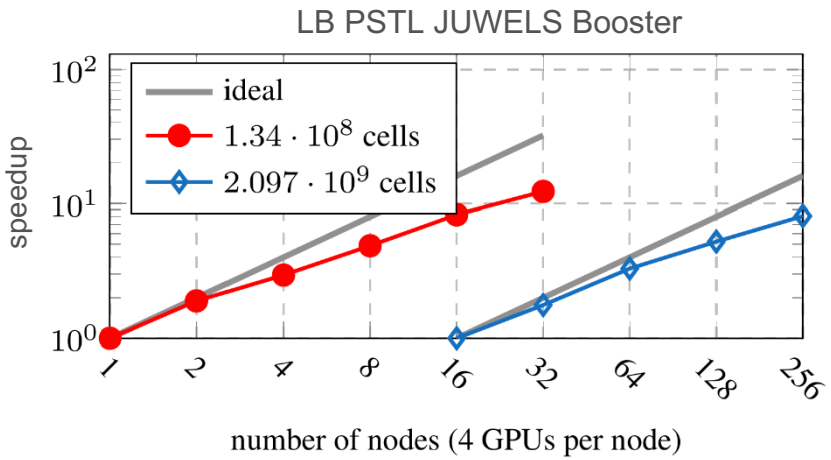










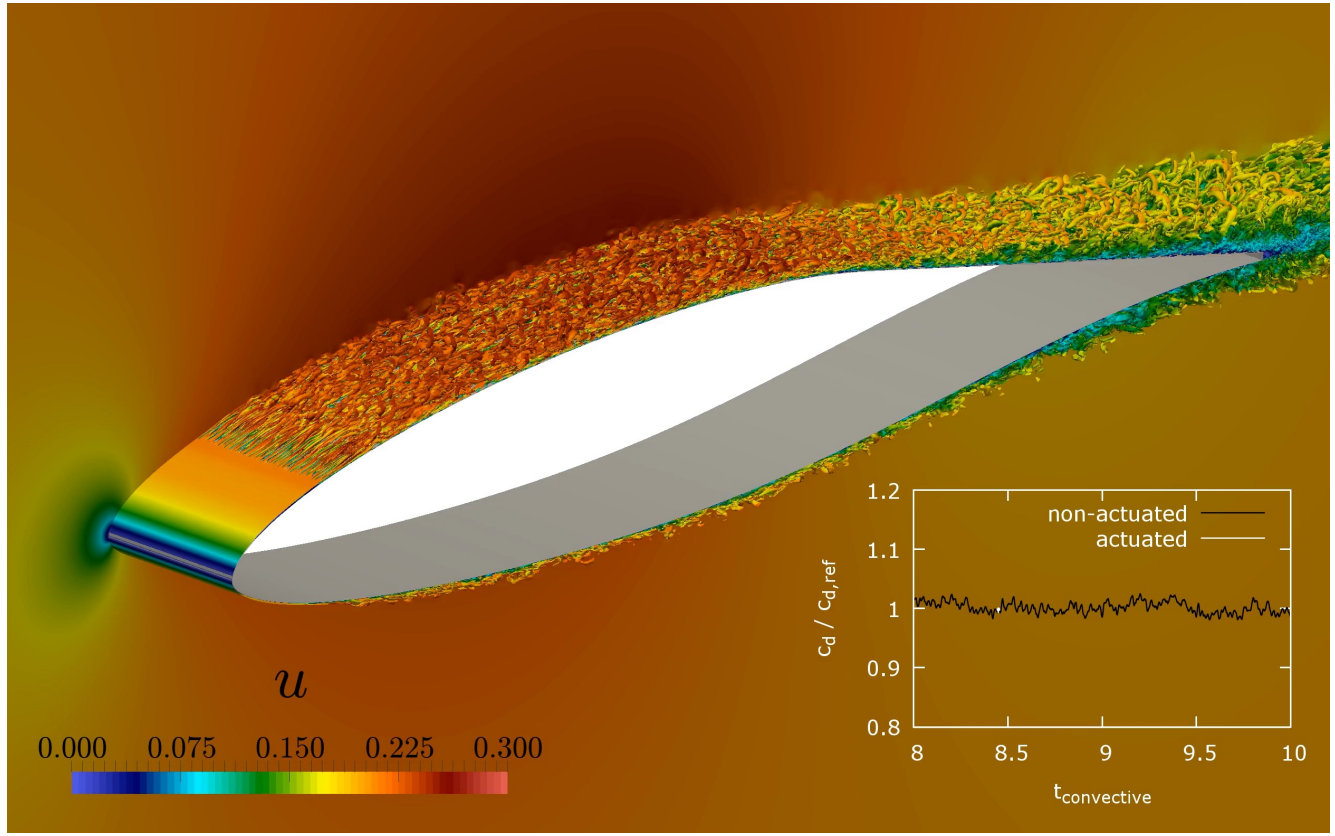


AI FOR CFD APPLICATIONS

AERODYNAMIC PREDICTIONS

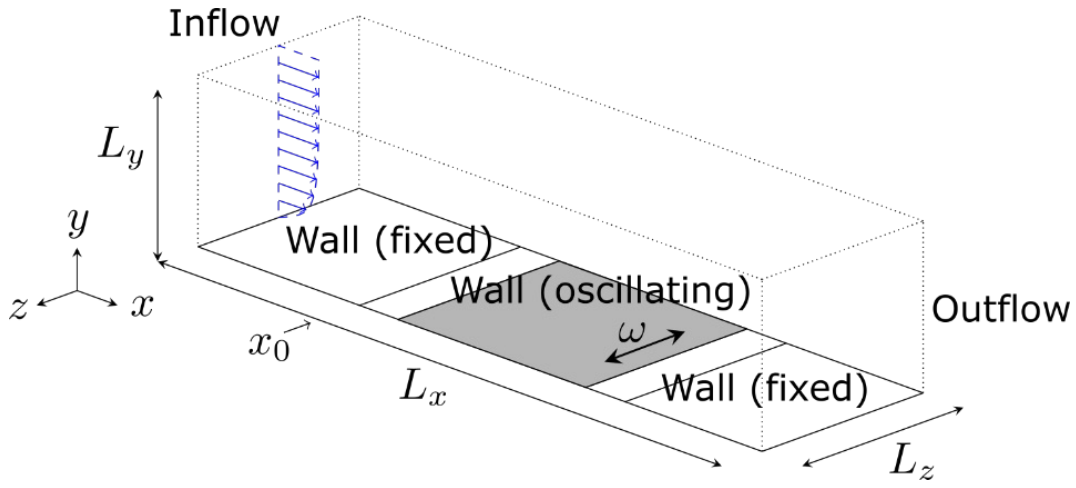
Active Drag Reduction (ADR)

- ADR spanwise traveling transversal surface waves
- Reduce energy consumption and emissions:
  - Drag reduction
  - Net power savings
- Vision:
  - Short/middle-term: Understand mechanisms of ADR, reduce simulation costs, and optimize the design choices
  - Long-term: Application in airplanes in cruise flight, highspeed trains, etc.
- Simulation-based analysis requires HPC resources

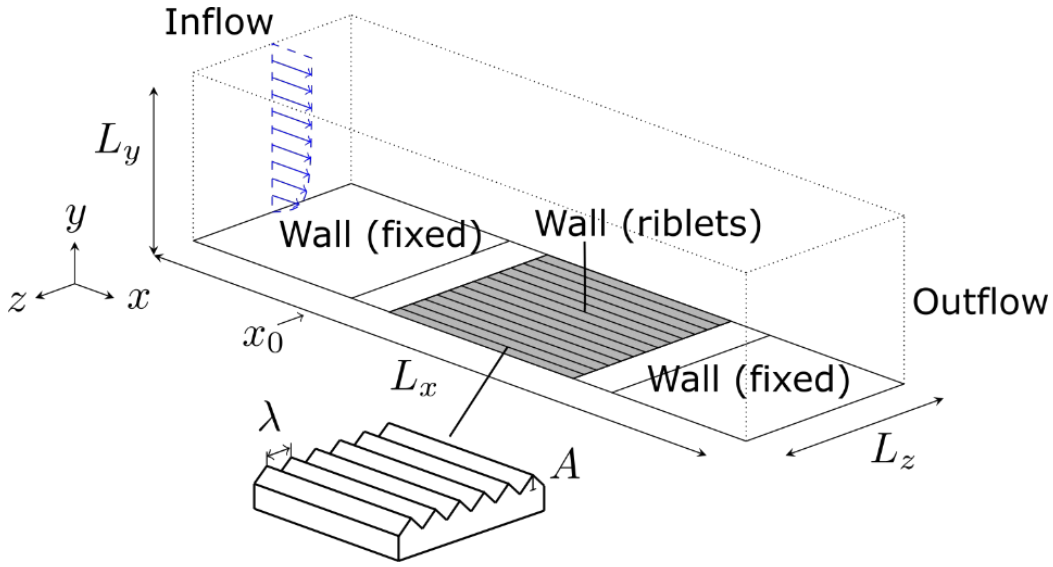


ACTIVE VS. PASSIVE DRAG REDUCTION

Example: Spanwise oscillating wall



Example: Riblet surface

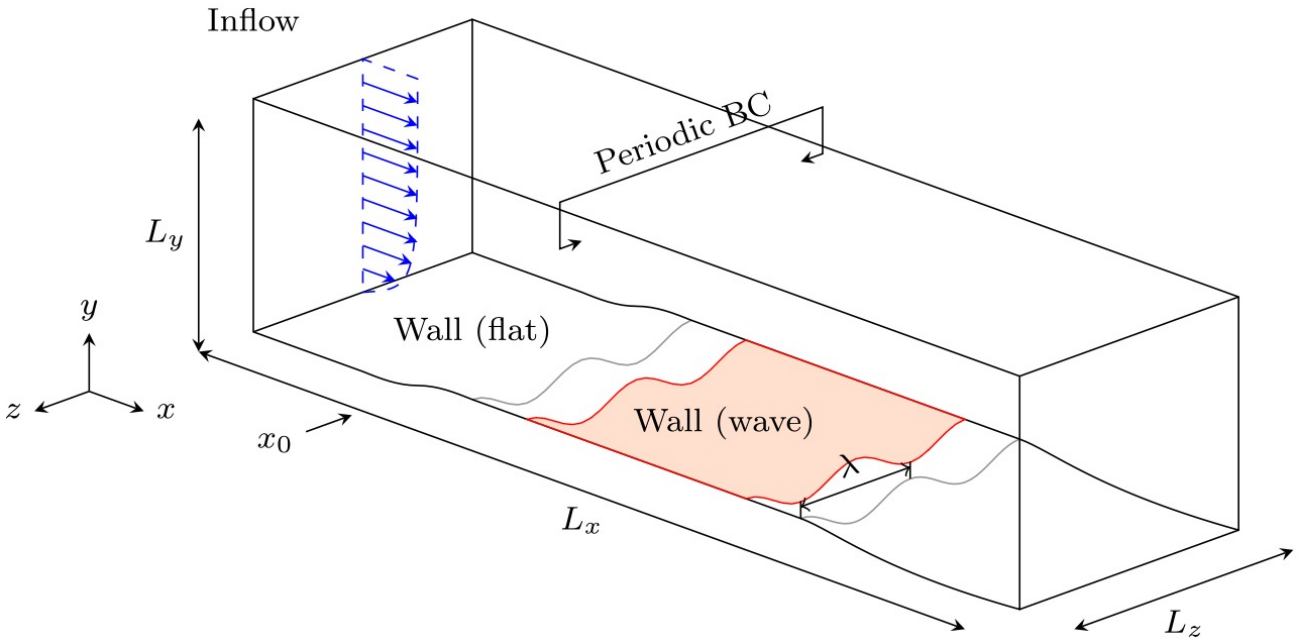


Active DR, e.g., oscillating wall	Passive DR, e.g., riblet surface
⊕ Flexible to flow conditions	⊖ Tailored to flow conditions, hence less flexible
⊕ Higher DR / net power savings	⊖ Lower DR / net power savings
⊖ Power input and technological overhead	⊕ No power input
⊖ Low technological readiness level	⊕ Medium technological readiness level

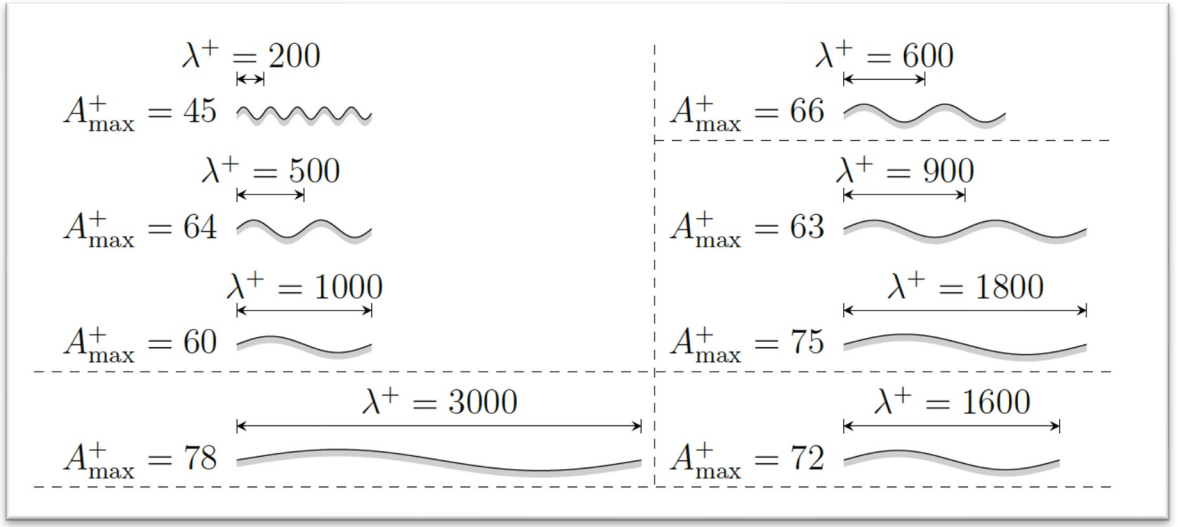
SPANWISE TRAVELLING SURFACE WAVE SIMULATIONS

Research questions

- What is the drag/net power (C_D, P_{net}) dependence on the actuation parameters?
- What are C_D, P_{net} for non-simulated cases?



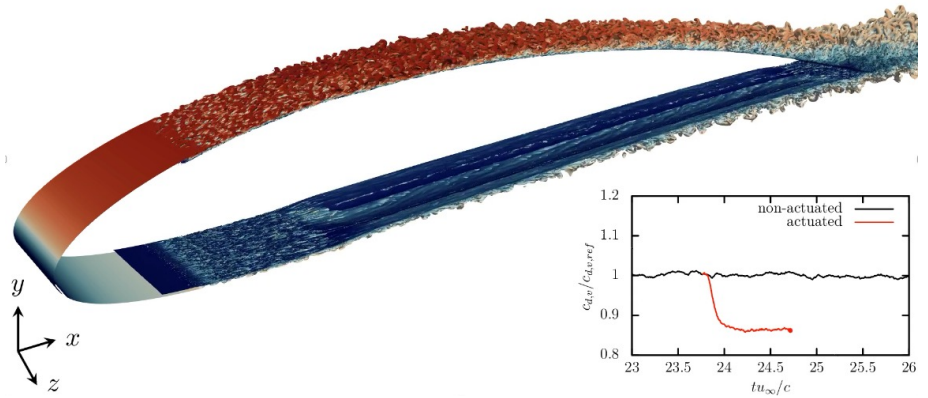
In total 88 configurations… 8.3TB of data



$$y^+|_{\text{wall}}(z^+, t^+) = A^+ \cos\left(\frac{2\pi}{\lambda^+} z^+ + \frac{2\pi}{T^+} t^+\right)$$

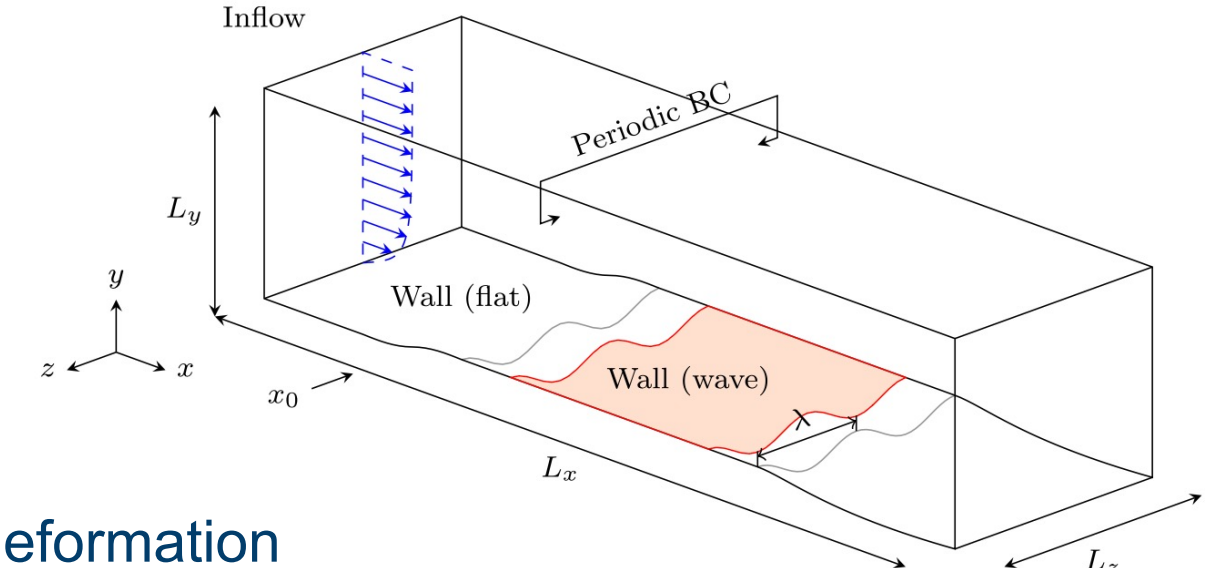
SIMULATION METHOD

M-AIA on structured meshes [7]



Simulating TBLs with a finite-volume method

- Solves the compressible Navier-Stokes equations:
  - AUSM method for the formulation of the inviscid fluxes
  - MUSCL scheme for flow quantity reconstruction at the cell surfaces
  - 5-stage Runge-Kutta for temporal integration
  - LES modeling via MILES approach (numerical dissipation models viscous dissipation)
  - Synthetic turbulence generation at inlet
- Uses a structured, body-fitted mesh



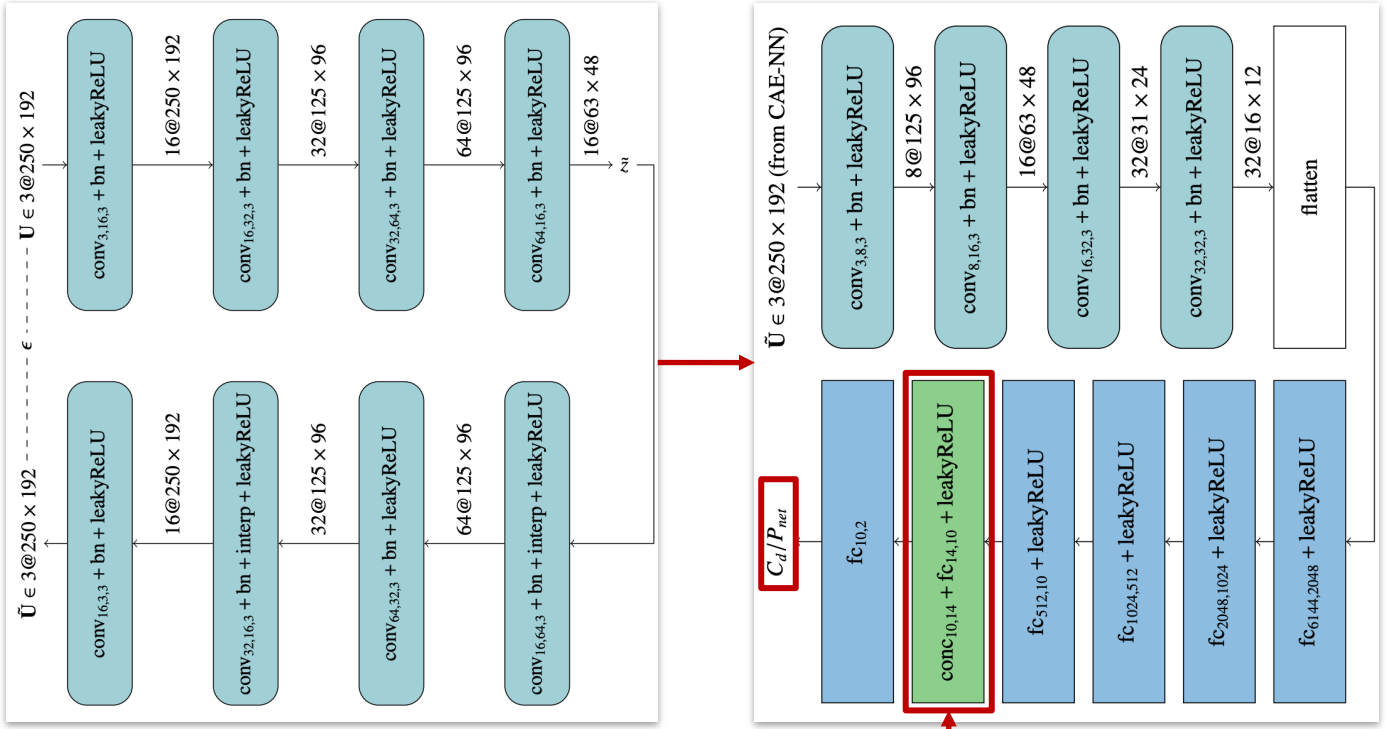
- Arbitray Lagrangian Eulerian (ALE) for mesh deformation
- Simulation parameters: $Re_\theta = 1{,}000, Ma = 0.1$

COMBINING MULTIPLE CNNS

Convolutional AutoEncoders (CAEs) [8,9]

Training

- First train CAE-NN section of the network
- Once the CAE-NN network provides satisfactory results, the rest of the network for the prediction of the QoIs is trained
- Train for C_D, P_{net} separately

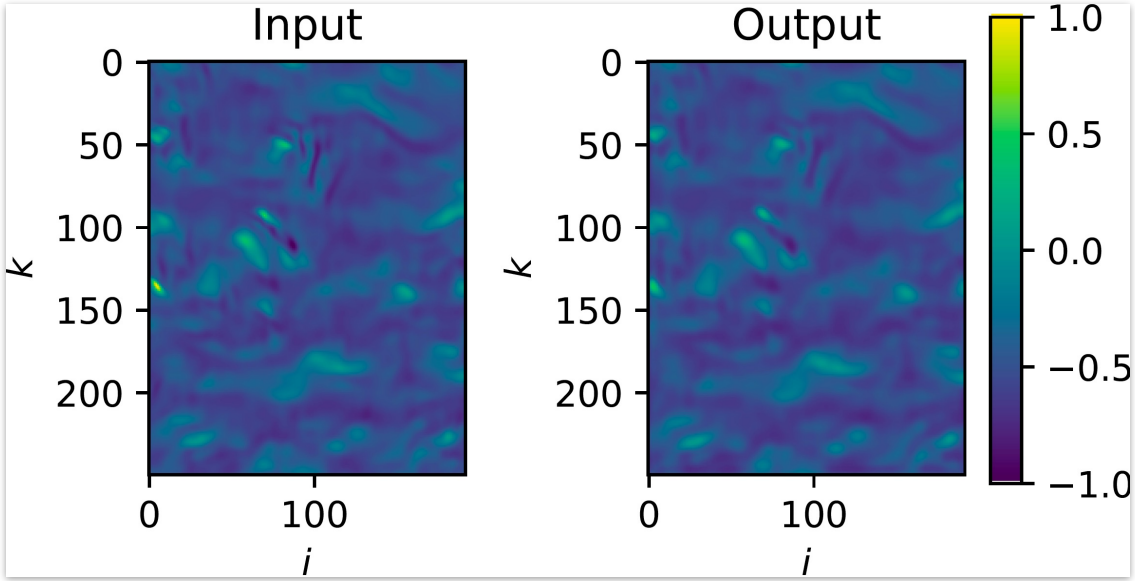


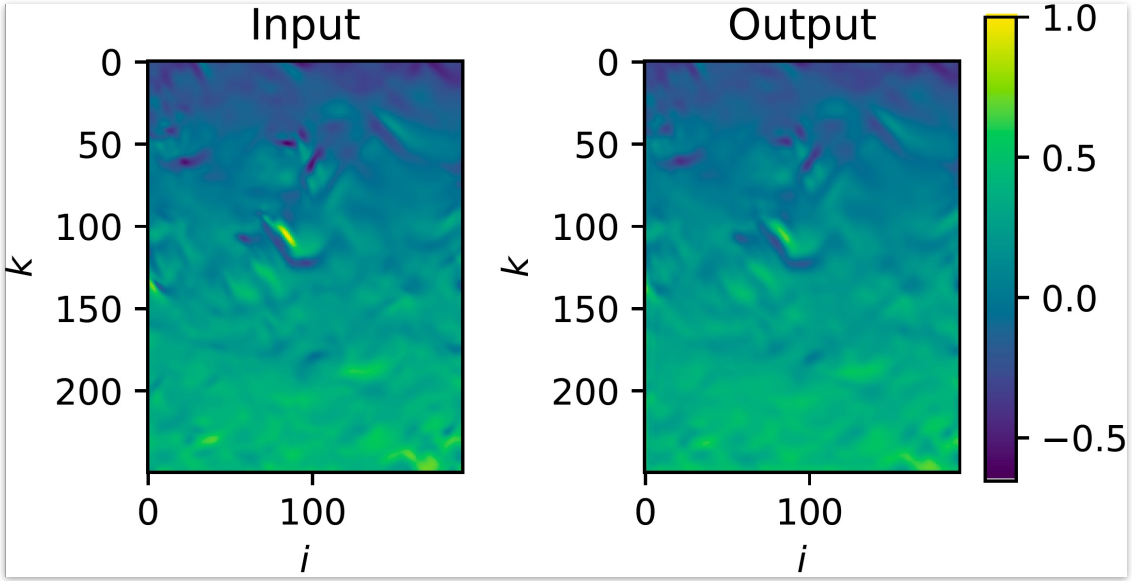
- During testing, the decoder part of the CAE-NN initially reconstructs the higher-resolution velocity fields from the latent space, which is fed to the rest of the corresponding network.

COMBINING MULTIPLE CNNS

Convolutional AutoEncoders (CAEs) [8,9]

Results: direct comparison of LES and the reconstrution from the CAE-NN





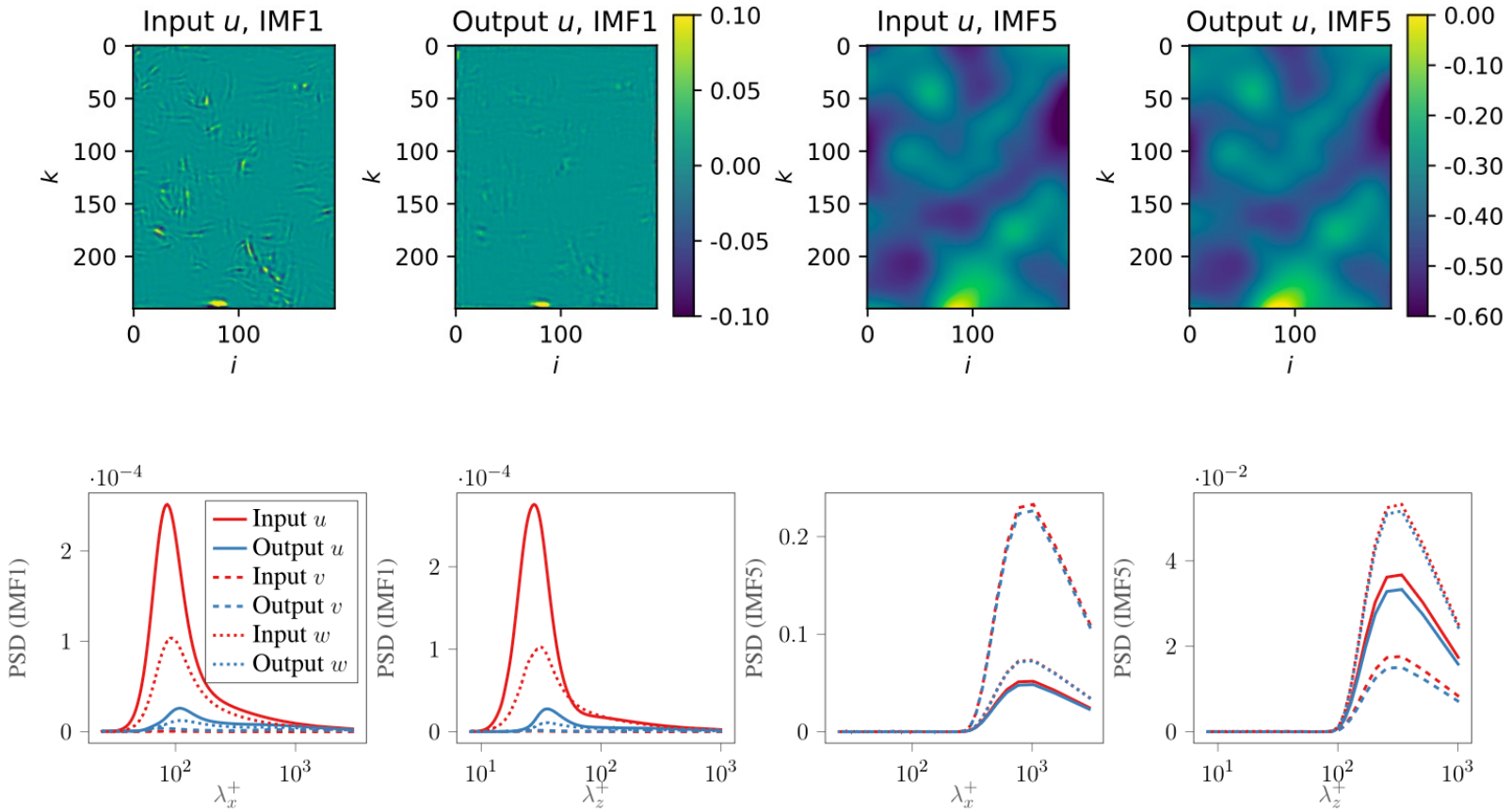
Cross-sections of the scaled streamwise (u^*) (left) and spanwise (w^*) (right) velocity components from the LES (Input) and reconstruction by CAE-NN (Output).

COMBINING MULTIPLE CNNS

Results [8-10]

A deeper look into the performance of the CAE

- Measure CAE's scale-specific accuracy via NA-MEMD analysis
- Intrinsic Mode Functions (IMFs) are modal representations categorized with respect to flow features that share a certain range of scales
- Large-scale flow features contained in IMF5 of the reconstructions are in satisfactory agreement
- Agreement is worse for the small-scale structures visible from IMF1



NA-MEMD = Noise-Assisted Multivariate Empirical Mode Decomposition

COMBINING MULTIPLE CNNS

Convolutional AutoEncoders (CAEs) [8,9]

Results: Performance of the CAE-PN

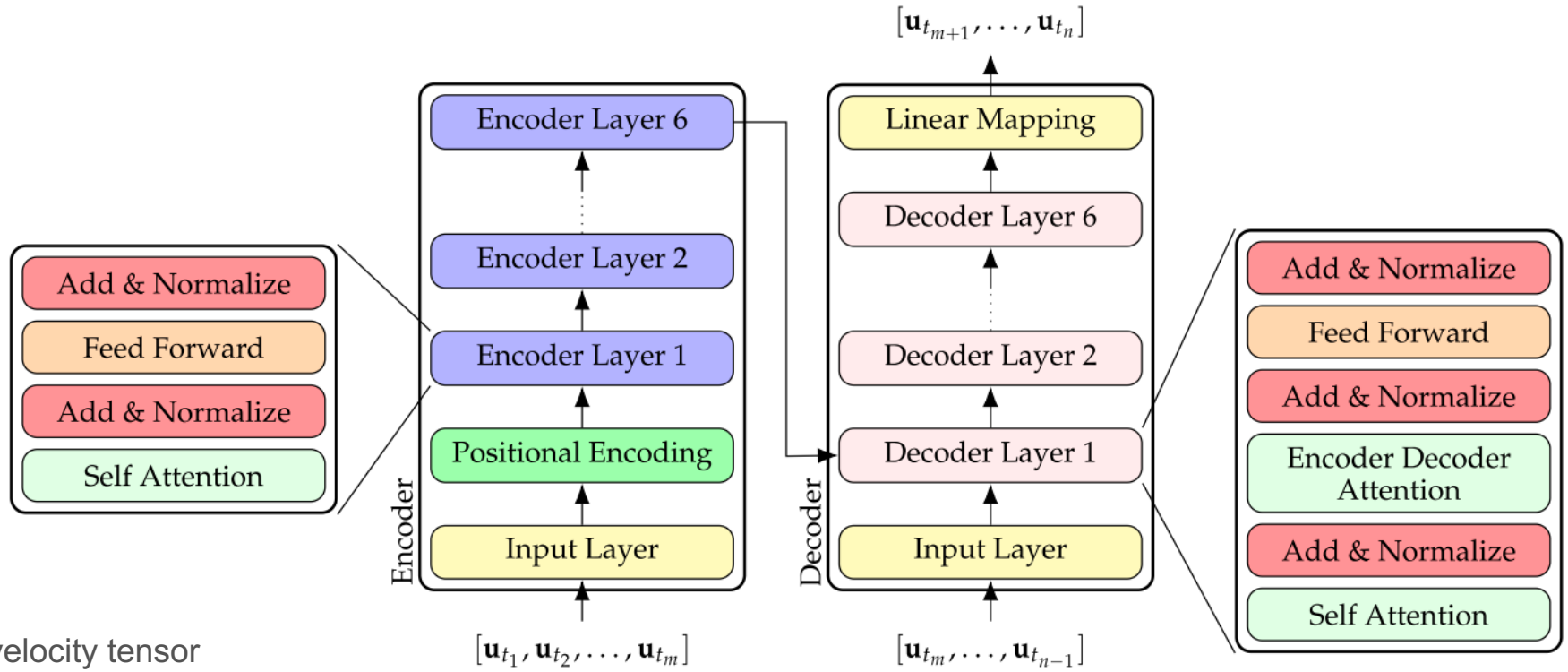
	Interpolation				Extrapolation			
	max	min	mean	std	max	min	mean	std
ΔC_d(tar)	0.48	0.15	0.35	0.11	0.51	0.09	0.28	0.14
ΔC_d(pred)	0.55	0.18	0.37	0.08	0.52	0.14	0.29	0.08
deviation	14.5	20.0	5.71	27.3	1.96	55.6	3.57	42.9
P_{net} (tar)	0.36	0.08	0.14	0.09	0.16	0.06	0.11	0.04
P_{net} (pred)	0.32	0.01	0.15	0.07	0.23	0.09	0.14	0.03
deviation	11.1	87.5	7.14	22.2	43.8	50.0	27.3	25.0

Satisfactory interpolation performance. Extrapolation performance not good for net power saving

TRANSFORMERS FOR TEMPORAL PREDICTION

Architecture [11,12]

Idea: Perform temporal predictions to replace expensive LES time steps



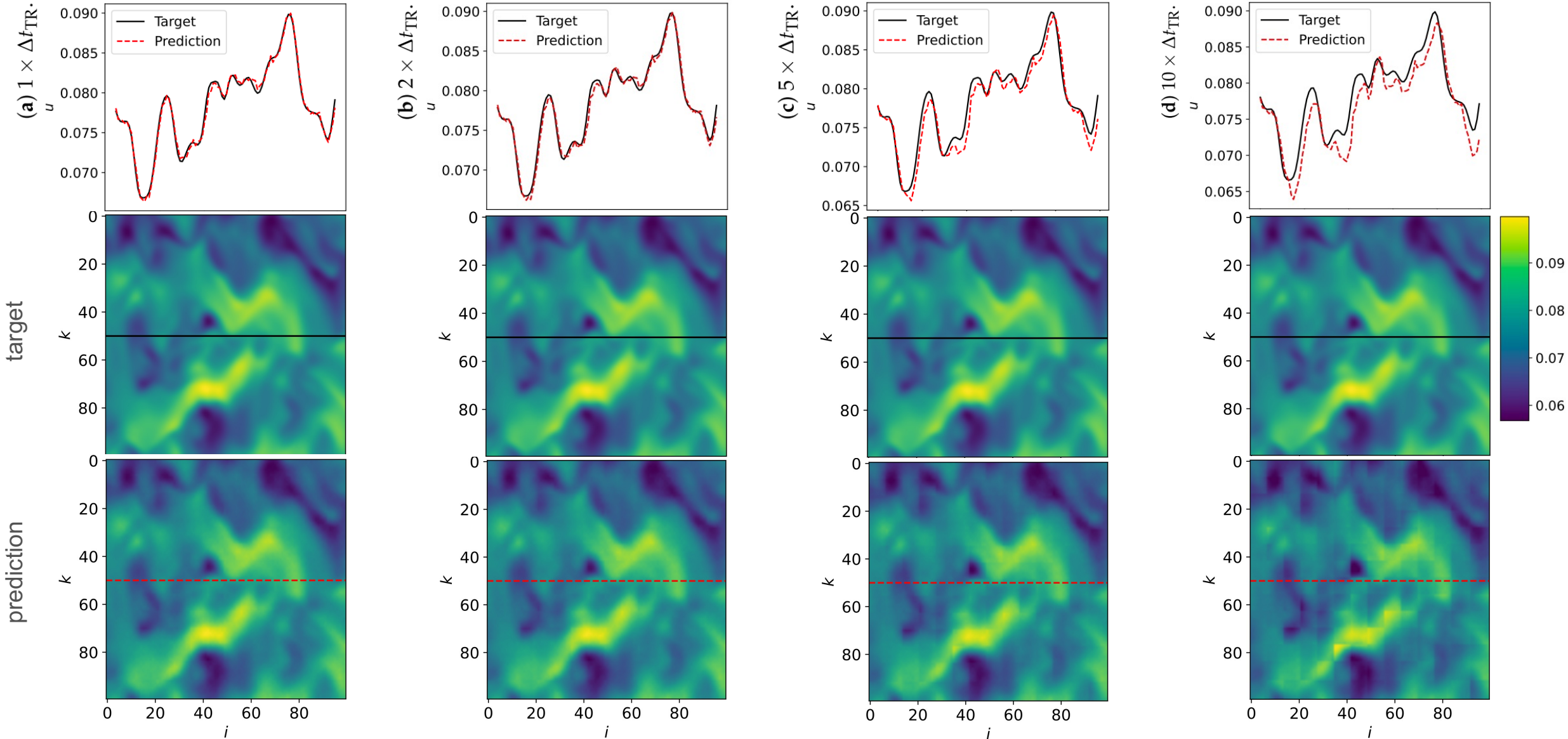
dimension of the velocity tensor

empirically chosen

$$L_{total} = \frac{\alpha}{d_u} \sum_{i=1}^{d_u} \left(\mathbf{u}_{t_n} - \mathbf{u}^*_{t_n}\right)^2 + \frac{\beta}{d_u} \sum_{i=1}^{d_u} \left[\left(\frac{\partial \mathbf{u}_{t_n}}{\partial x} - \frac{\partial \mathbf{u}^*_{t_n}}{\partial x} \right)^2 + \left(\frac{\partial \mathbf{u}_{t_n}}{\partial y} - \frac{\partial \mathbf{u}^*_{t_n}}{\partial y} \right)^2 + \left(\frac{\partial \mathbf{u}_{t_n}}{\partial z} - \frac{\partial \mathbf{u}^*_{t_n}}{\partial z} \right)^2 \right], \quad \alpha = 1.0, \beta = 0.06$$

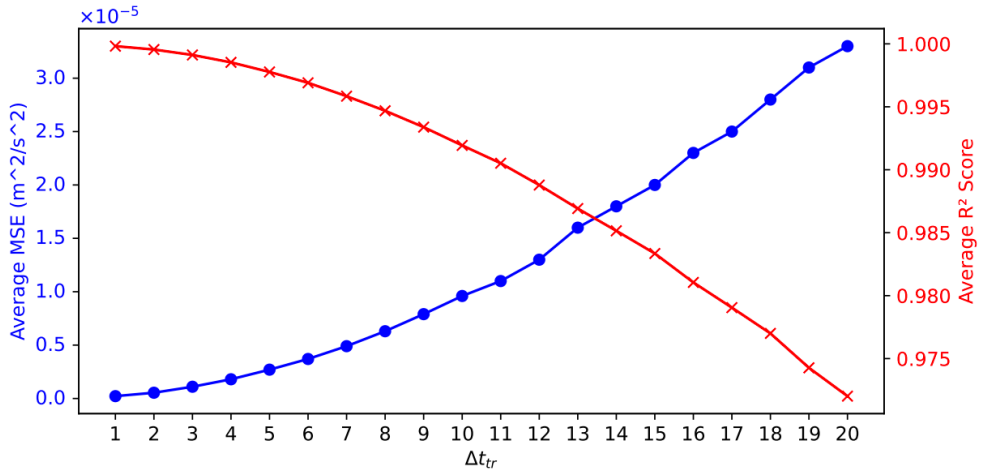
TRANSFORMERS FOR TEMPORAL PREDICTION

Results: Prediction performance (velocity components) [11,12]



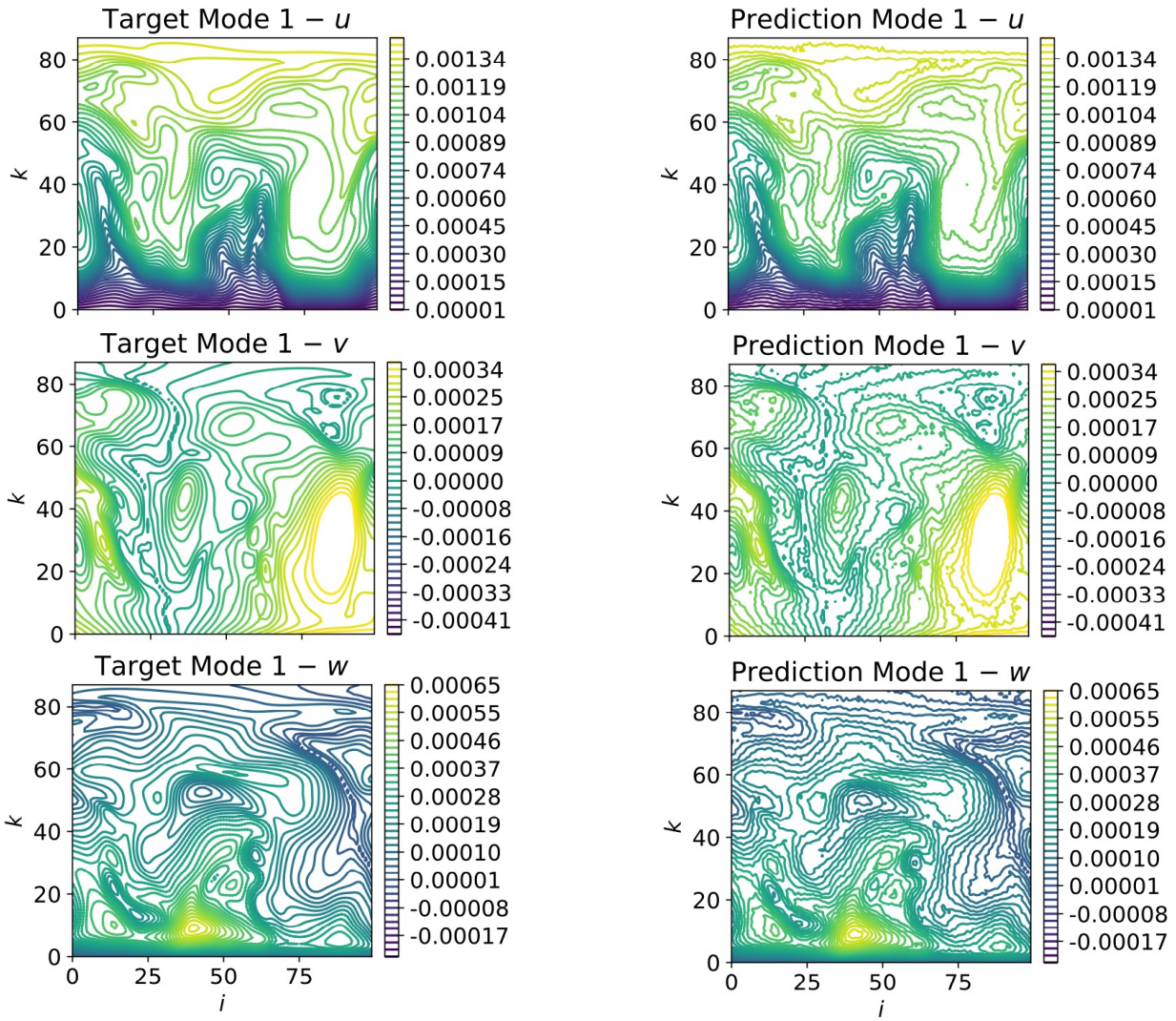
TRANSFORMERS FOR TEMPORAL PREDICTION

Results: Prediction (MSE, DMD) and computational performance [11,12]



Model accuracy with increasing no. of time steps

	LES	Transformer	Factor
wall-time (node sec.)	36.48	0.685	**53.3** (speed-up)
memory (GB)	466.69	0.422	**1105.9** (memory savings)



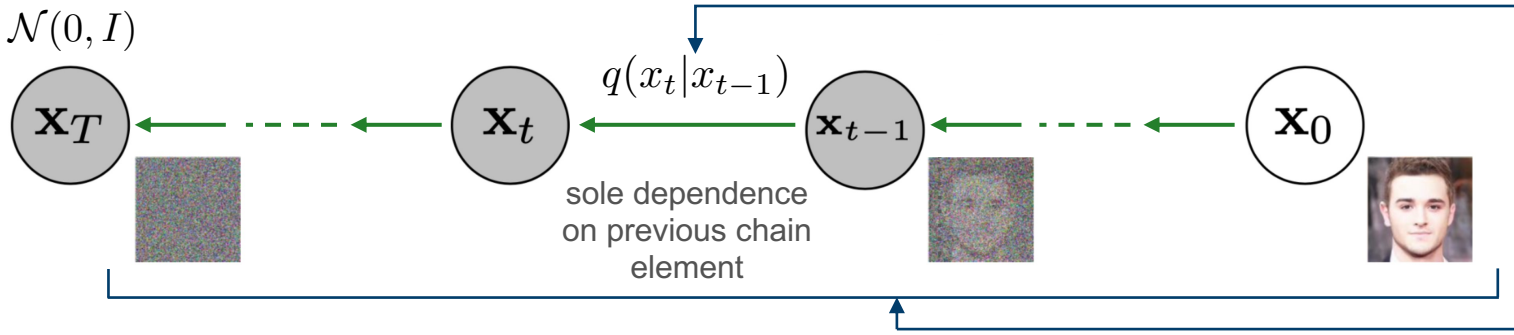
DMD analysis: first mode comparison for $\Delta t_r = 2$.

CONVOLUTIONAL DEFILTERING MODELS (CDMS)

Super-Resolution modeling

The general idea behind CDMs [13]

- Makes use of the idea of diffusion models
- Model a Markov chain as forward diffusion process with continuous Gaussian noising



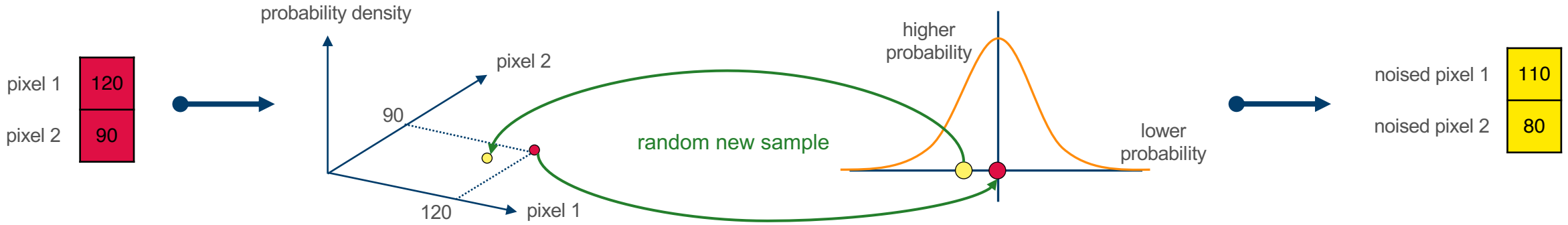
variance schedule
controls step size

$$\{\beta_t \in (0,1)\}_{t=1}^{T}$$

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

- Change pixel values slightly in the area of the Gaussian distribution function

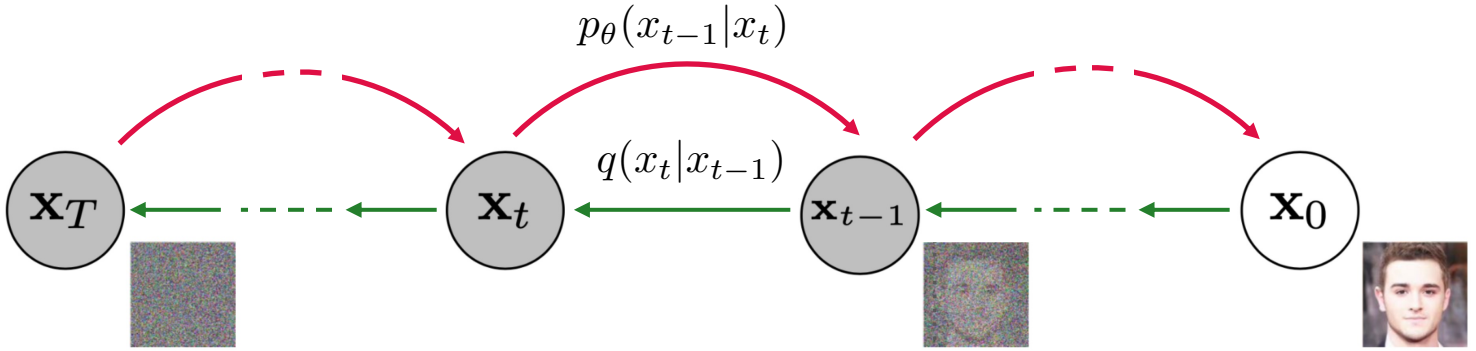


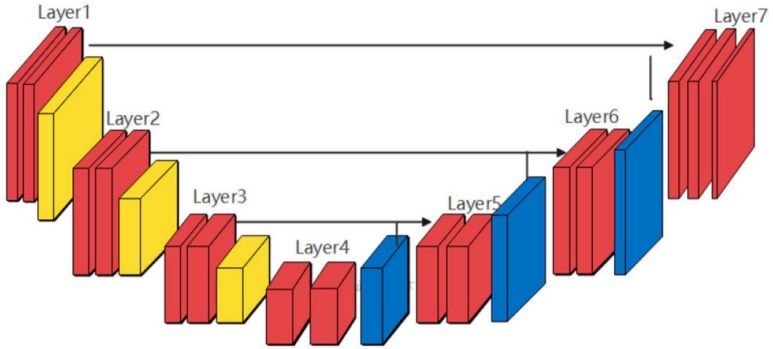
CONVOLUTIONAL DEFILTERING MODELS (CDMS)

Super-Resolution modeling

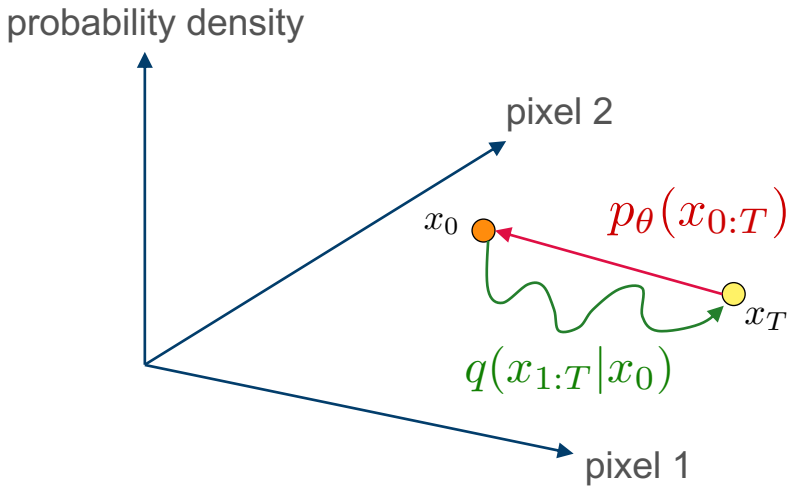
The general idea behind CDMs [13]

- Now train a U-Net to model the reverse Markov process.





$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$



- θ are the parameters of the neural network
- We would like to minimize the negative log-likelihood:

$$-\log p_\theta(x_0)$$

JÜLICH Forschungszentrum

CONVOLUTIONAL DEFILTERING MODELS (CDMS)

Super-Resolution modeling

The general idea behind CDMs [13]

Towards a loss function of the U-Net:

is untractable (too many paths)

$$-\log p_\theta(x_0) = -\log \int p_\theta(x_{0:T}) dx_{1:T}$$

Jensen's inequality →

$$\leq -\mathbb{E}_{q(x_{1:T}|x_0)} \left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \right]$$

Evidence Lower Bound (ELBO)

$$= -\mathbb{E}_{q(x_{1:T}|x_0)} \left[D_{KL}\left(q(t_T|x_0)||p(x_T)\right) + \sum_{t>1} D_{KL}\left(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1|x_t})\right) - \log p_\theta(x_0|x_1) \right]$$

$$= \dots$$

$p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta, \sigma_\theta)$

(Gaussian with fixed σ_t)

independent of θ

small

true posterior
(but we are missing the true x_0)

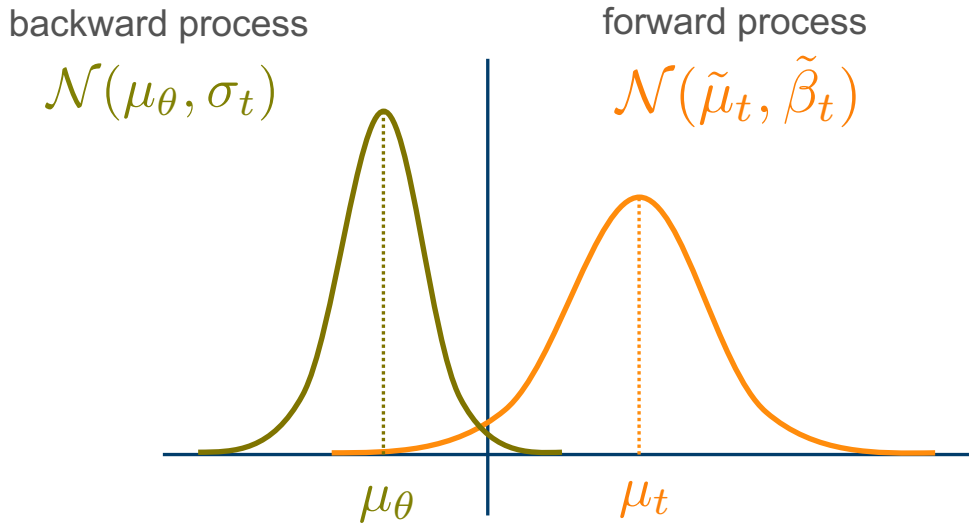
(Gaussian) $q(x_{t-1}|x_t, x_0) = \mathcal{N}(\tilde{\mu}_t, \tilde{\beta}_t)$

CONVOLUTIONAL DEFILTERING MODELS (CDMS)

Super-Resolution modeling

The general idea behind CDMs [13]

Towards a loss function of the U-Net:



$$\dots = \mathbb{E}_{q(x_{1:T}|x_0)}\left[\sum_{t>1}\frac{1}{2\sigma_t^2}||\tilde{\mu}_t - \mu_\theta(x_t,t)||^2\right]$$

$$= \mathbb{E}_{q(x_{1:T}|x_0)}\left[\sum_{t>1}\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\alpha_t)}||\epsilon - \epsilon_\theta(x_t,t)||^2\right]$$

noise difference

reparametrization

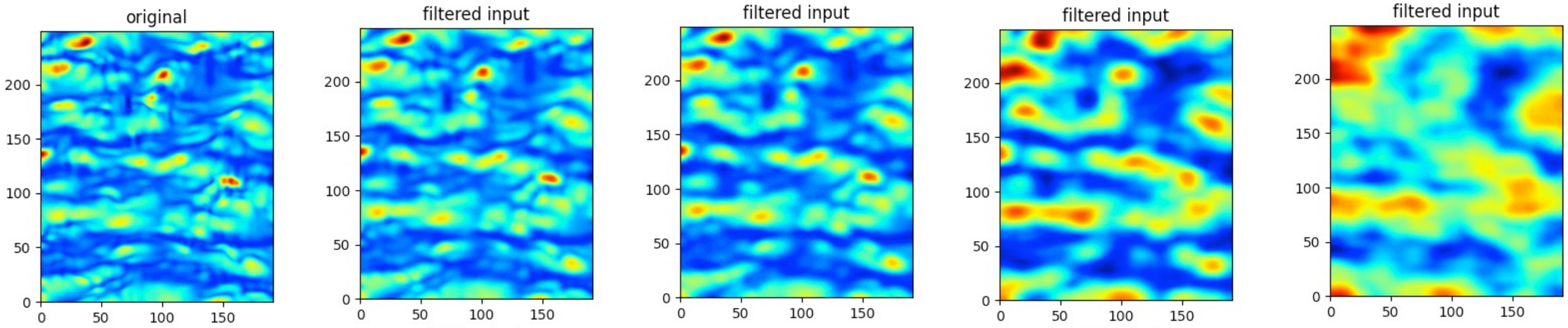
$$\tilde{\mu}_t = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

$$\mu_\theta(x_t,t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t,t)\right)$$

CONVOLUTIONAL DEFILTERING MODELS (CDMS)

Super-Resolution modeling: application to turbulent flows [9]

Instead of to pictures, apply to physical fields (gradually increase filter width)

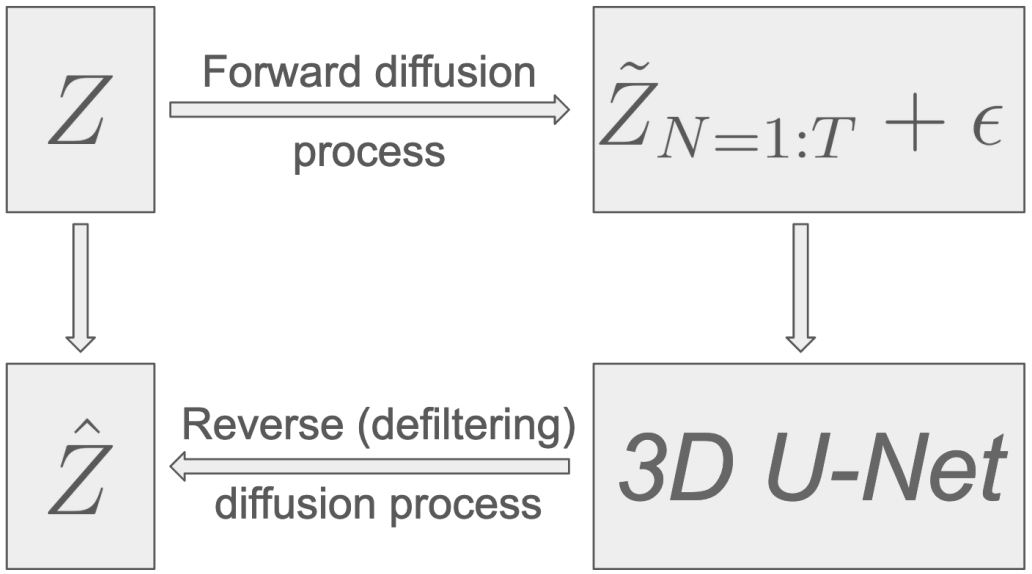


Streamwise velocity component of a turbulent flow field

CONVOLUTIONAL DEFILTERING MODELS (CDMS)

Super-Resolution modeling: application to turbulent flows [9]

Use 3D U-Nets to perform the defiltering

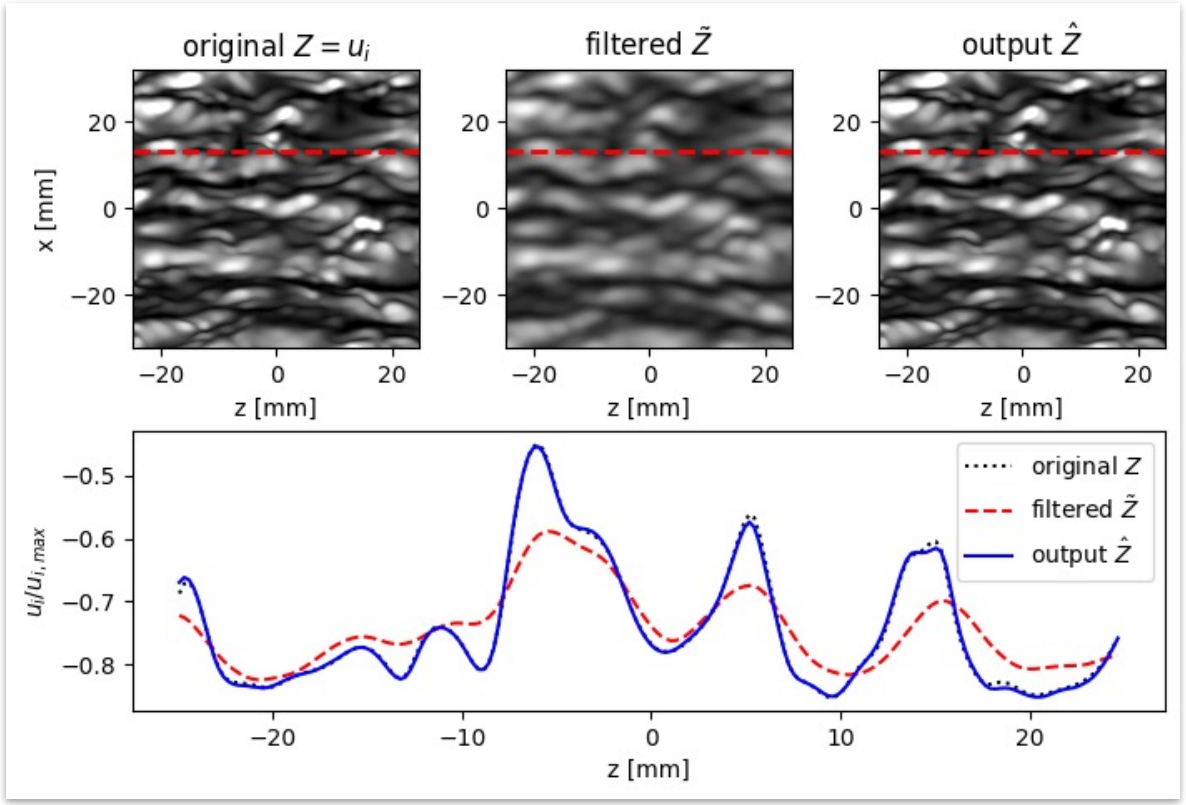


- Input Z is filtered (Gaussian filter) to $\tilde{Z}$
- Add small noise ϵ to avoid overfitting to filter width
- Provide $\tilde{Z}$ to the 3D U-Net
- Loss-function as $\hat{Z} \overset{!}{=} Z$
  - Pixel-to-pixel
  - Gradient-to-gradient
  - Physics-based, etc.
- The output is the “defiltered” $\tilde{Z}$
- Repeat by increasing filter width N

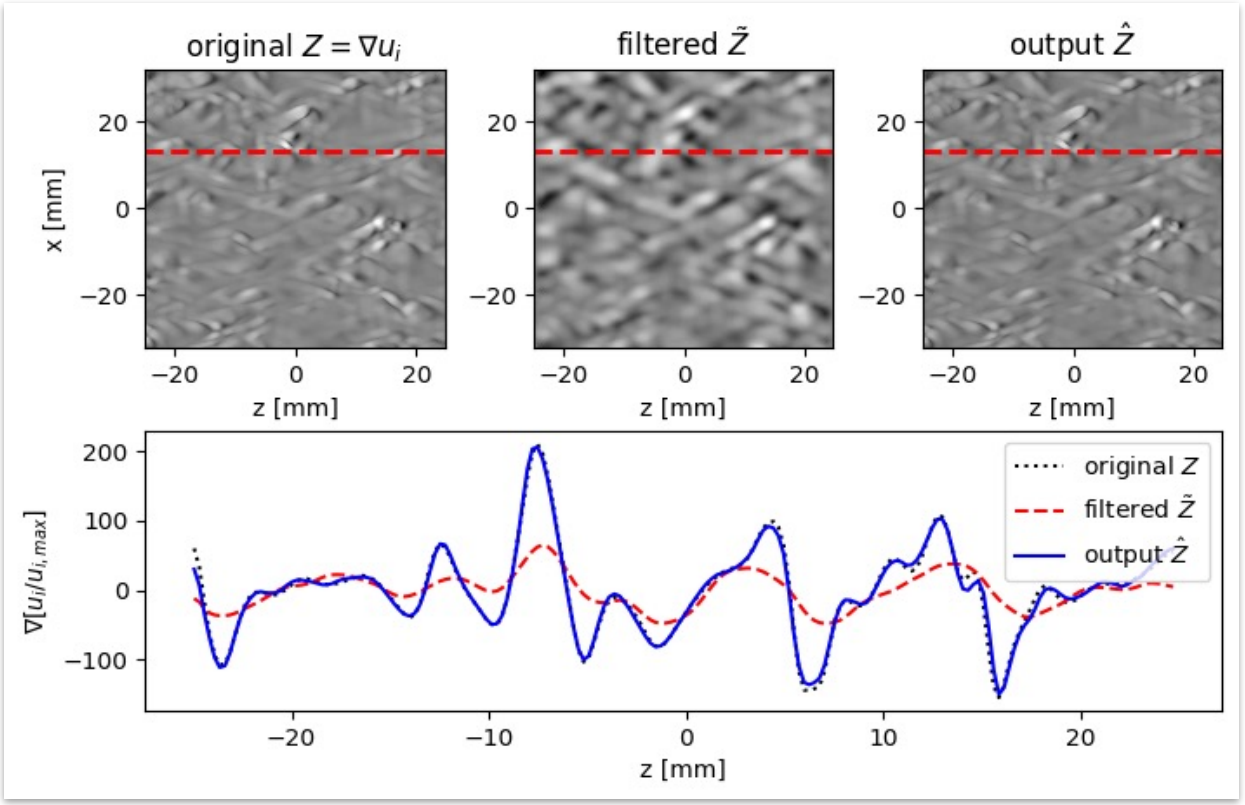
CONVOLUTIONAL DEFILTERING MODELS (CDMS)

Super-Resolution modeling: application to turbulent flows [9]

Results of the reconstruction



Tested filter width *N=5:* 0.5% error on average

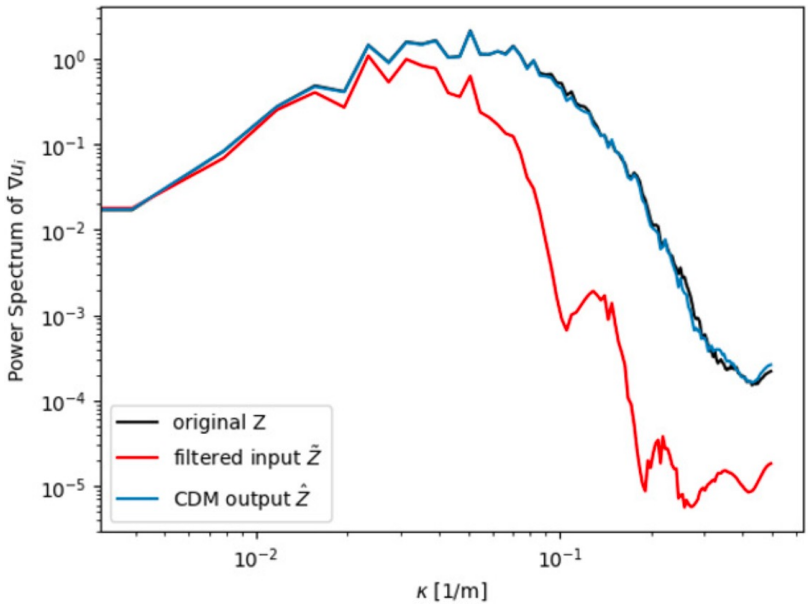


Gradient: 3.5% error on average

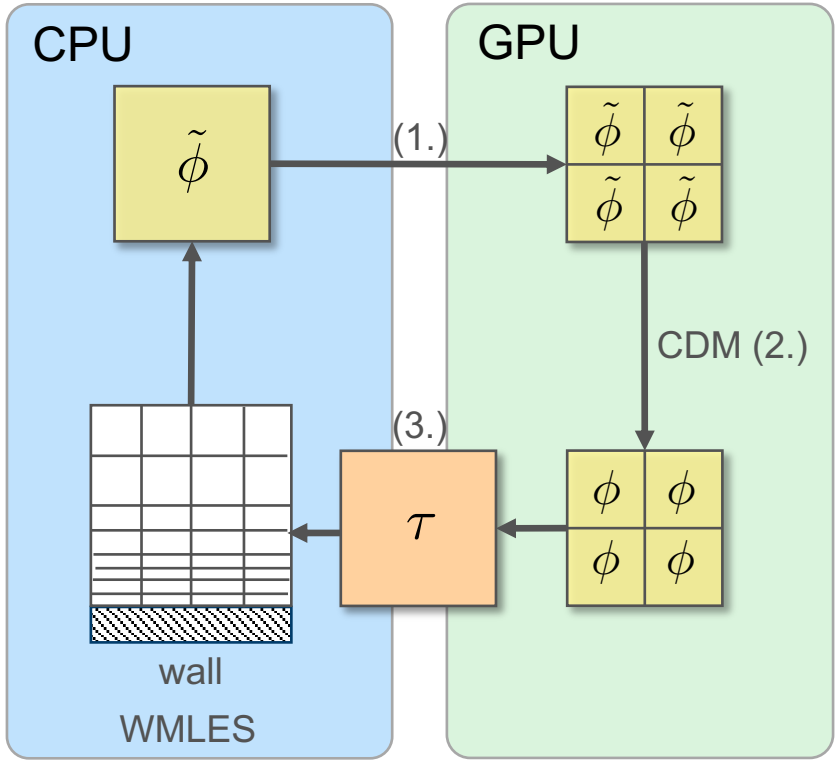
CONVOLUTIONAL DEFILTERING MODELS (CDMS)

Super-Resolution modeling: application to turbulent flows [9]

Results and future integration into simulations



PSD of the instantaneous velocity gradients



Using CDMs in WMLES

1. Zero-degree interpolation of the LES fields close to wall
   - Allocation of a DNS grid
   - Lower memory foot print (only at layers close to the wall and for selected quantities)
2. Provide interpolated fields to CDM
3. Output from CDM back to LES, e.g., the wall-shear stresses

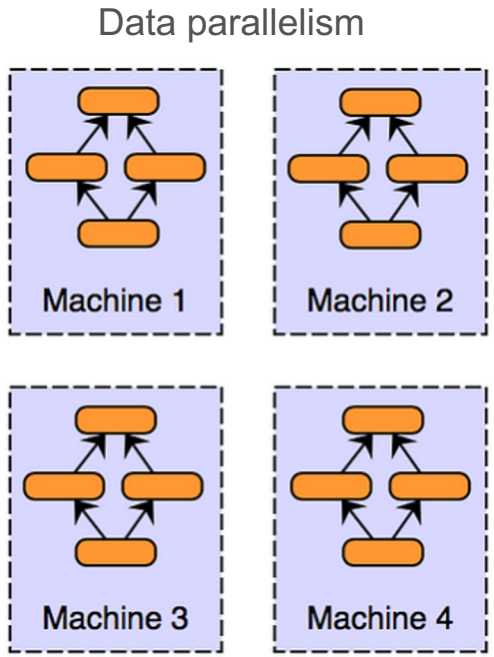
RUNNING AI EFFICIENTLY AT SCALE ON HPC SYSTEMS

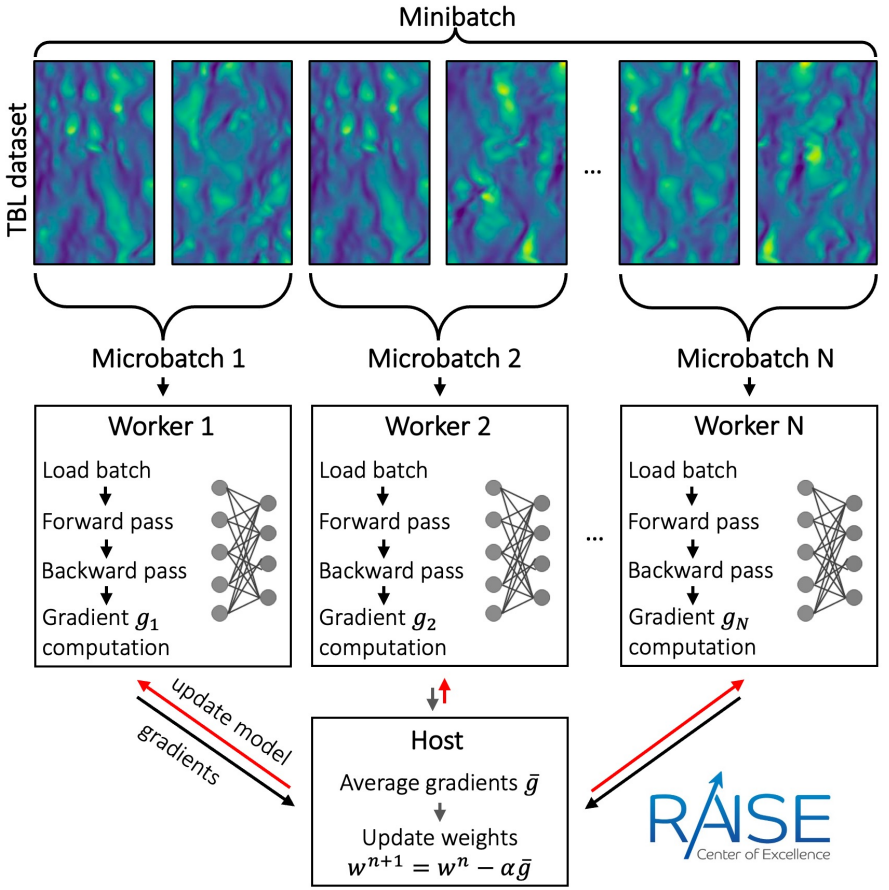
EFFICIENT TRAINING IN PARALLEL

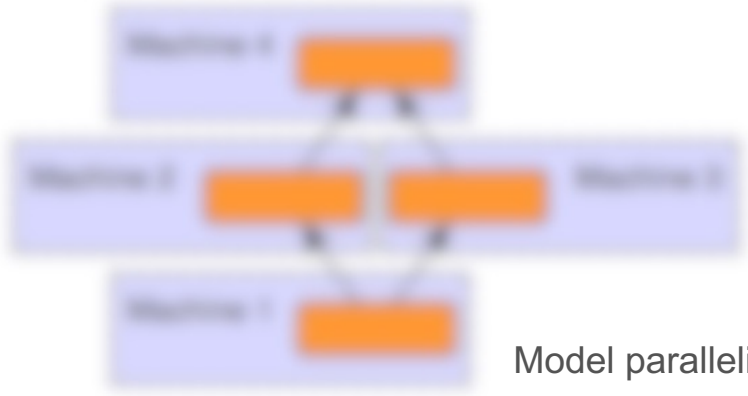
Bringing training to GPUs [9]

Most common case: Data parallelism

- **Data Distributed Parallel** with, e.g., PyTorch-DDP, Horovod, DeepSpeed
- A mini batch is split over the number of workers (GPUs) = micro batches
- **Each worker**
  - has a copy of the architecture
  - works only on a micro batch
- **A supervisor node**
  - collects all gradients
  - updates the weights
  - sends model updates to the workers







Model parallelism

EFFICIENT TRAINING IN PARALLEL

Bringing training to GPUs

To make the most efficient use of GPUs, the full memory should be used

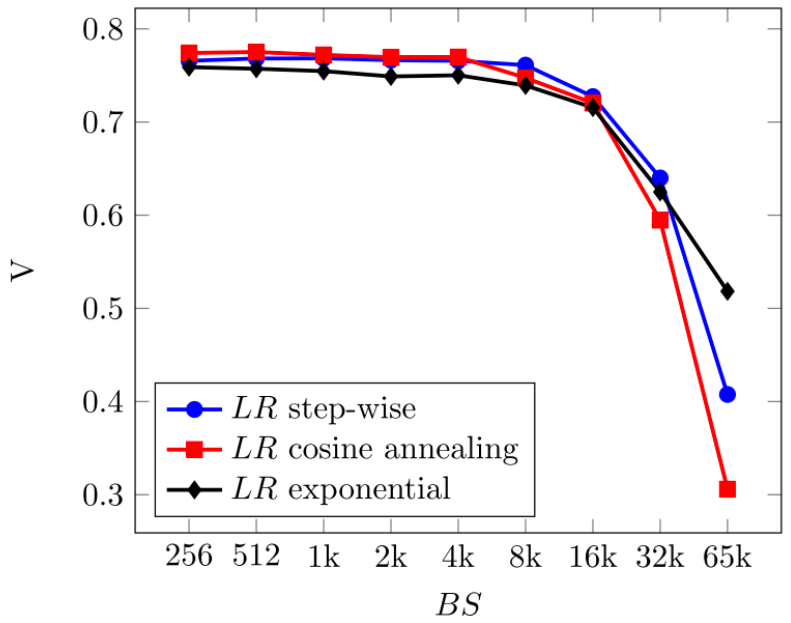
- Reduces limited computation per GPU
- Reduces communication overhead

Large batch size problem

- The batch size is a function of the number of GPUs
- updates of the model parameter happen too sparsely

Despite of great scaling, the accuracy of the model is not improving

Adapting the learning rate might help



$$B = f(no.\ GPUs)$$

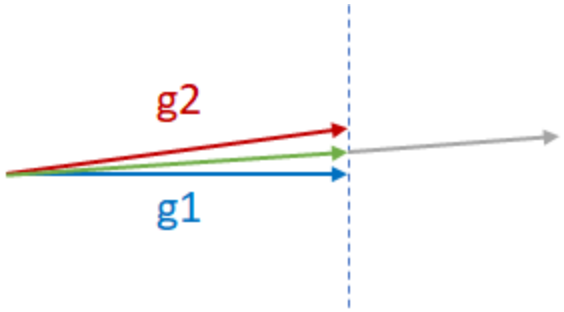
EFFICIENT TRAINING IN PARALLEL

Improving the training of Neural Networks

Model accuracy improvement / faster convergence

- Use advanced optimizers such as Adam / use adpative learning rate methods
- Make sure not to exceed a limiting batch size (by using, e.g., too many GPUs)
- Use the Adaptive Summation Algorithm (AdaSum)

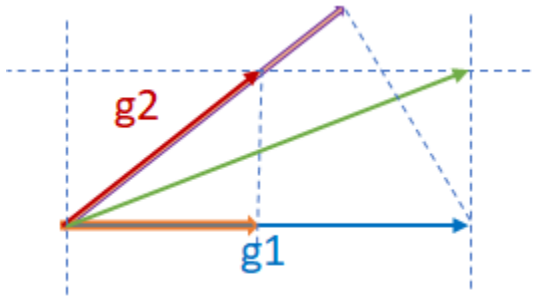
parallel / almost parallel



div.: $\vec{g}_r = \vec{g}_1 + \vec{g}_2$

OK: $\vec{g}_r = \frac{1}{2}(\vec{g}_1 + \vec{g}_2)$

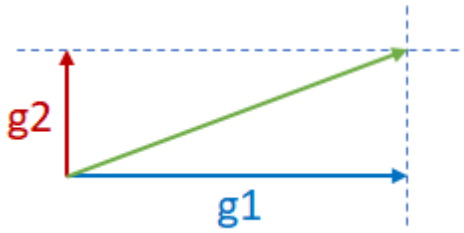
neither parallel / orthogonal



div.: $\vec{g}_r = \vec{g}_1 + \vec{g}_2$

OK: $\vec{g}_r = \left(1 - \frac{\vec{g}_1 \cdot \vec{g}_2}{2|\vec{g}_1|^2}\right)\vec{g}_1 + \left(1 - \frac{\vec{g}_1 \cdot \vec{g}_2}{2|\vec{g}_2|^2}\right)\vec{g}_2$

orthogonal



OK: $\vec{g}_r = \vec{g}_1 + \vec{g}_2$

JÜLICH Forschungszentrum

EFFICIENT TRAINING IN PARALLEL

Improving the training of Neural Networks

Training speed-up with Automatic Mixed Precision (AMP)

```
FP16_NN_FW                          // Compute network forward pass in FP16
L = FP32_L(X,W)                     // Compute loss in FP32
L = alpha * L                       // Scale by a large factor to ensure
                                    // representability in FP16
FP16 res = FP_16_NN_BW              // Compute backward pass in FP16
if!(res == NAN || res == INF)       // Check for NAN / INF
{
  grad = 1/z grad                   // Unscale gradients
  updateOpt                         // Update optimizer
}
```

EFFICIENT TRAINING IN PARALLEL

Improving the training of Neural Networks

Training speed-up with the cuDNN autotuner

- Many compute kernels are tuned for NVIDIA GPUs
- Using the cuDNN library runs benchmarks on the computation and automatically selects the most efficient kernels

Gradient accumulation

- Reduces the `AllReduce` communications
- First sums all gradients in a bucket and fuses all gradients in a single parallel operation
- Overlaps computation and communication (asynchronous communication)

EFFICIENT TRAINING IN PARALLEL

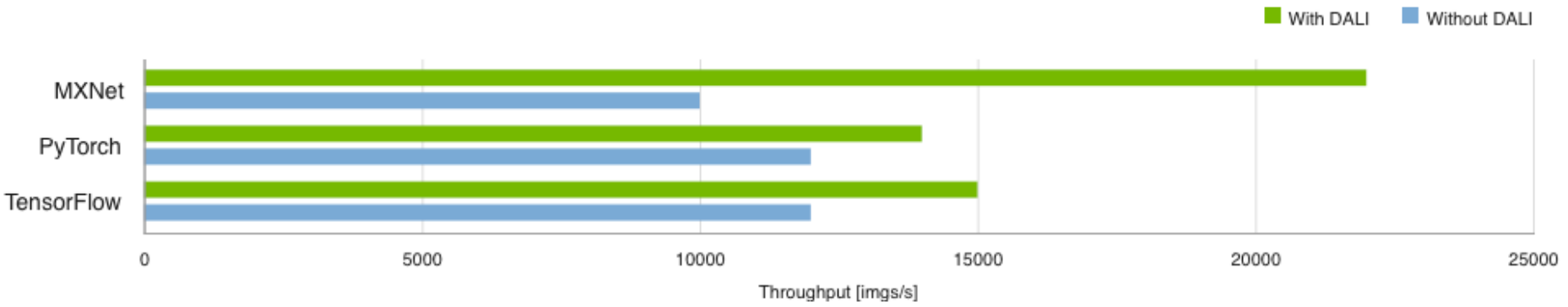
Improving the training of Neural Networks

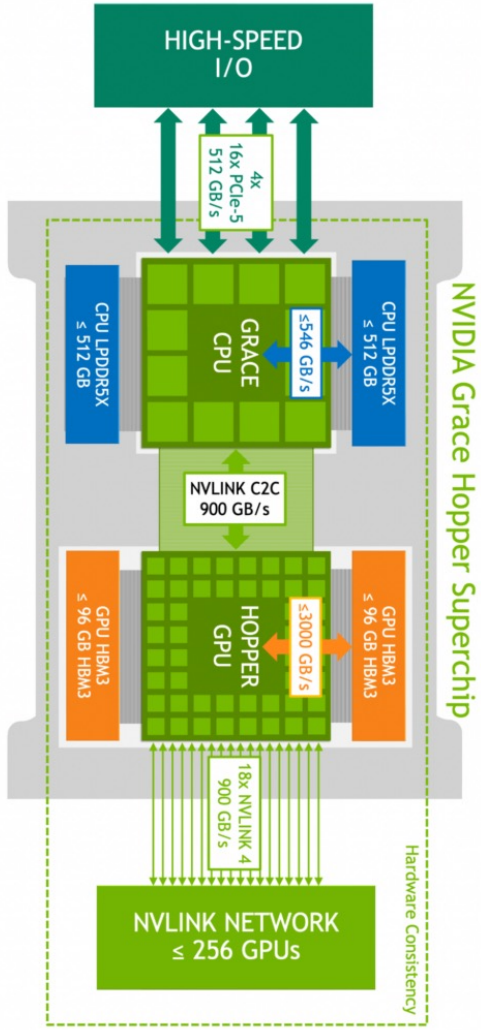
Training speed-up with CUDA-aware MPI

- Makes use of the direct connection between GPU and network link

Training speed-up with improved I/O

- Play around with multi-process data loaders such as NVIDIA DALI





JÜLICH Forschungszentrum

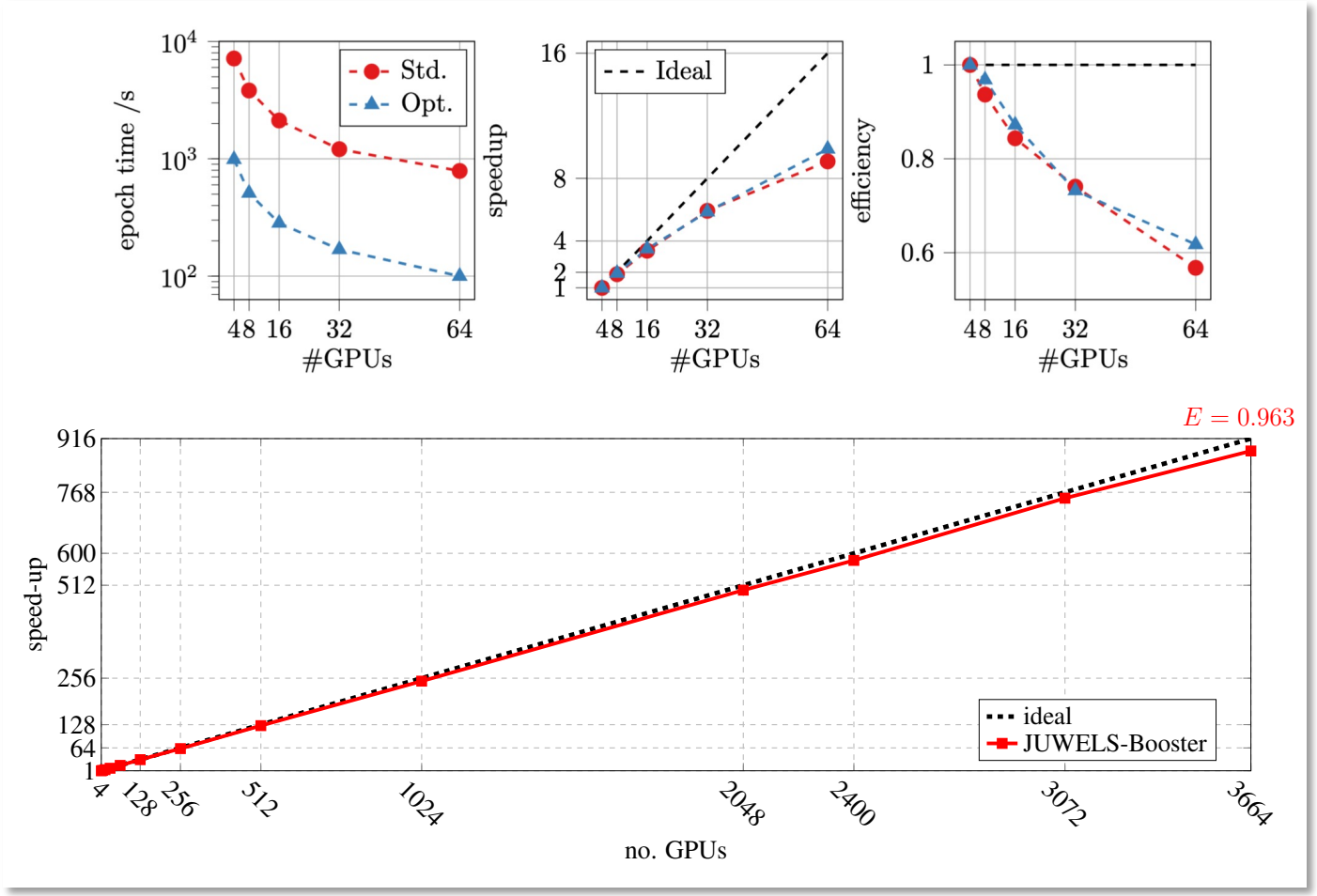
EFFICIENT TRAINING IN PARALLEL

Improving the training of Neural Networks [9]

Improvement of up to 90% using

- AMP
- cuDNN
- Gradient accumulation
- CUDA-aware MPI

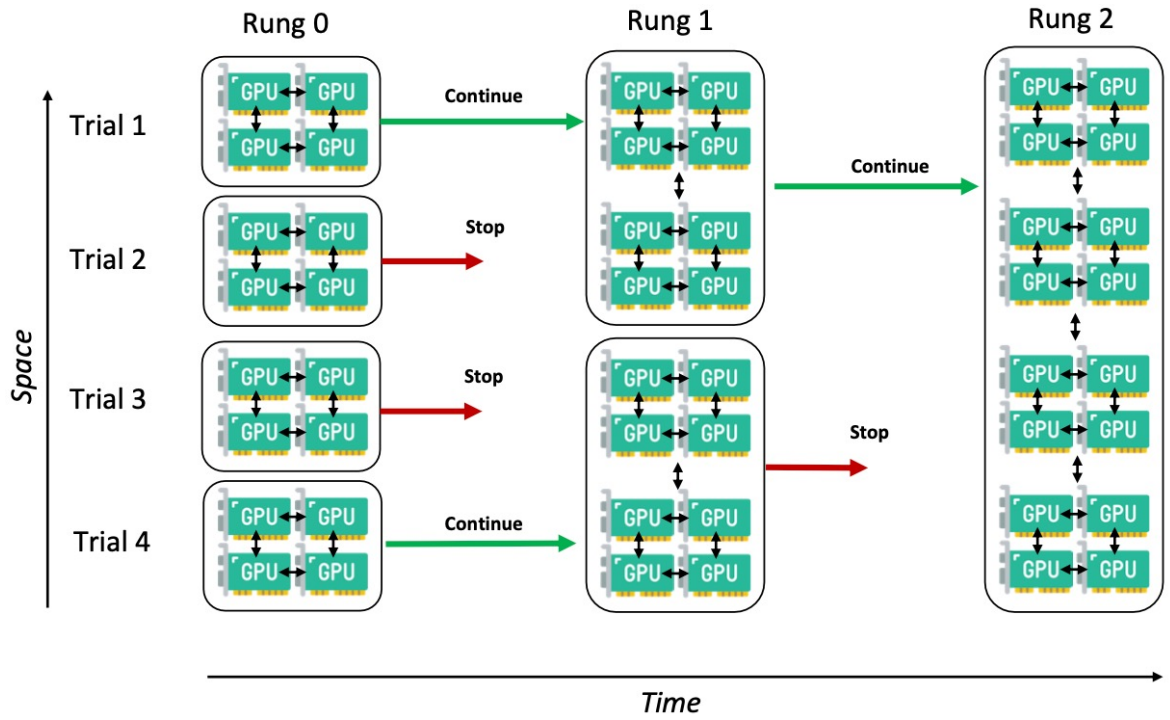
Scaling of a DL-library on the (almost) whole JUWELS Booster with up to 96% efficiency

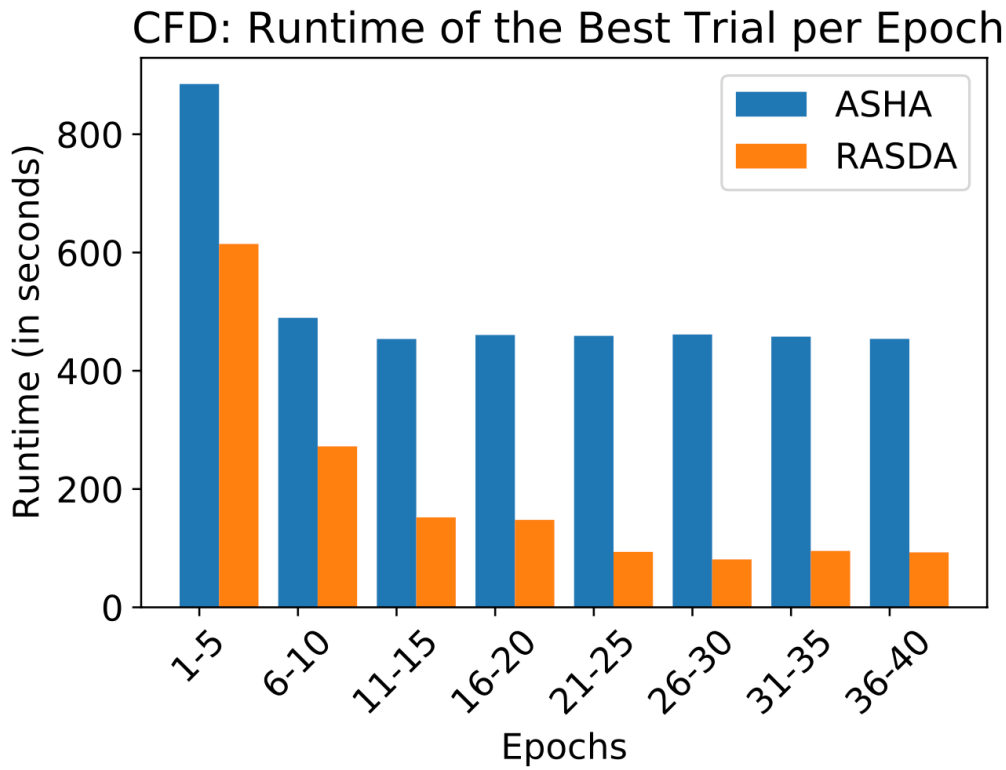


EFFICIENT TRAINING IN PARALLEL

Improving the training of Neural Networks: Hyperparameter Optimization *

Resource Adaptive Successive Doubling Algorithm (RASDA) [14]





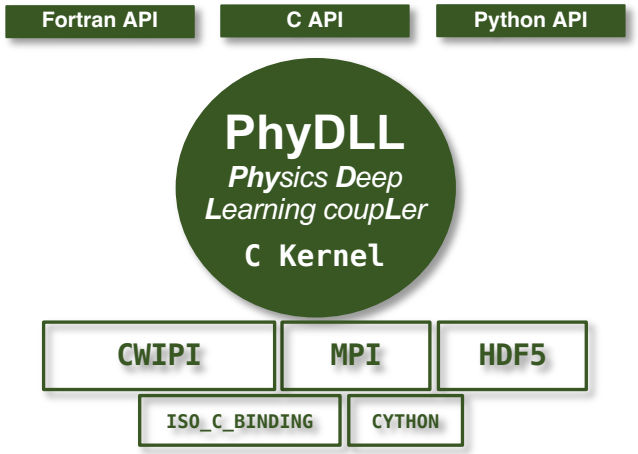
* More details about HPO: Visit Eric Wulff's talk this afternoon

SUMMARY, CONCLUSIONS, ONGOING AND NEXT STEPS

SUMMARY, CONCLUSIONS, ONGOING AND NEXT STEPS

- There exist various methods for physics-aware AI model training
  - Note: the methods shown are just an excerpt, there is a whole zoo of models
- The choice of model and its parameters is highly dependent on the considered case
  - Note: some methods work better than others (e.g., PINNs vs. DNNs, PC-AEs vs. AEs)
  - Hyperparameter tuning is certainly required for model improvement
  - Training costs need to be considered
- Next and ongoing steps:
  - Full coupling of simulations and training / inference *
  - Hyperparameter optimization for model tuning
  - Integration with the itwinai ** and AI4HPC *** libraries

AI4HPC



* PhyDLL: https://phydll.readthedocs.io/en/latest
** itwinai https://itwinai.readthedocs.io
*** AIA4HPC https://ai4hpc.readthedocs.io

REFERENCES

[1] Suarez, E., Eicker, N., & Lippert, T. (2019). Modular Supercomputing Architecture: From Idea to Production. In Contemporary High Performance Computing (pp. 223–255). CRC Press. https://doi.org/10.1201/9781351036863-9

[2] Lintermann, A., Meinke, M., & Schröder, W. (2020). Zonal Flow Solver (ZFS): a highly efficient multi-physics simulation framework. International Journal of Computational Fluid Dynamics, 34(7–8), 458–485. https://doi.org/10.1080/10618562.2020.1742328

[3] Lintermann, A., Schlimpert, S., Grimmen, J. H., Günther, C., Meinke, M., & Schröder, W. (2014). Massively parallel grid generation on HPC systems. Computer Methods in Applied Mechanics and Engineering, 277, 131–153. https://doi.org/10.1016/j.cma.2014.04.009

[4] Lintermann, A. (2016). Efficient Parallel Geometry Distribution for the Simulation of Complex Flows. In Proceedings of ECCOMAS Congress 2016, pp. 1277–1293. https://doi.org/10.7712/100016.1885.5067

[5] Lintermann, A., & Schröder, W. (2020). Lattice–Boltzmann simulations for complex geometries on high-performance computers. CEAS Aeronautical Journal, 11(3), 745–766. https://doi.org/10.1007/s13272-020-00450-1

[6] Wegmann, T., Niemöller, A., Meinke, M., & Schröder, W. (2025). Parallel Eulerian-Lagrangian coupling method on hierarchical meshes. Journal of Computational Physics, 521(Part 1), 113509. https://doi.org/10.1016/j.jcp.2024.113509

[7] Albers, M., Meysonnat, P. S., Fernex, D., Semaan, R., Noack, B. R., & Schröder, W. (2020). Drag Reduction and Energy Saving by Spanwise Traveling Transversal Surface Waves for Flat Plate Flow. Flow, Turbulence and Combustion, 105(1), 125–157. https://doi.org/10.1007/s10494-020-00110-8

[8] Sarma, R., Albers, M., Inanc, E., Aach, M., Schröder, W., & **Lintermann, A.** (2022). Parallel and Scalable Deep Learning to Reconstruct Actuated Turbulent Boundary Layer Flows. Part I: Investigation of Autoencoder-Based Trainings. *ParCFD2022 33rd International Conference on Parallel Computational Fluid Dynamics*. https://juser.fz-juelich.de/record/1007692

[9] Sarma, R., Inanc, E., Aach, M., & **Lintermann, A.** (2024). Parallel and scalable AI in HPC systems for CFD applications and beyond. *Frontiers in High Performance Computing*, *2*. https://doi.org/10.3389/fhpcp.2024.1444337

[10] E. Mäteling, E., & Schröder, W. (2022). Analysis of spatiotemporal inner-outer large-scale interactions in turbulent channel flow by multivariate empirical mode decomposition. *Physical Review Fluids*, 7(3), 034603. https://doi.org/10.1103/PhysRevFluids.7.034603

[11] Sarma, R., Hübenthal, F., Inanc, E., & **Lintermann, A.** (2024). Prediction of Turbulent Boundary Layer Flow Dynamics with Transformers. *Mathematics*, *12*(19), 2998. https://doi.org/10.3390/math12192998

[12] Sarma, R., Hübenthal, F., Orland, F., Terboven, C., & **Lintermann, A.** (2025). Predicting Turbulent Boundary Layer Flows Using Transformers Coupled to the Multi-Physics Simulation Tool m-AIA. In *Proceedings of the 35th Parallel CFD International Conference 2024 (ParCFD 2024), Schriften des Forschungszentrums Jülich IAS Series 69* (pp. 76–79). Forschungszentrum Jülich GmbH Zentralbibliothek, Verlag. https://doi.org/10.34734/FZJ-2025-02459

[13] Ho, J., Jain, A., & Abbeel, P. (2020). Denoising Diffusion Probabilistic Models. *ArXiv*. https://doi.org/10.48550/arXiv.2006.11239

[14] Sarma, R., Inanc, E., Aach, M., & Lintermann, A. (2024). Parallel and scalable AI in HPC systems for CFD applications and beyond. Frontiers in High Performance Computing, 2. https://doi.org/10.3389/fhpcp.2024.1444337