AI+HPC
利用 NVIDIA Modulus 實踐 AI 於物理模擬

Jay Chen | Data Scientist @ NVIDIA

Dr. CK Lee | Lead of Taiwan NVAITC @ NVIDIA

AGENDA – DAY 2

09:30-10:20: Introduce Modulus

10:30-11:20: Physics Inspired Lab 0: Solve a Darcy flow problem using Fourier Neural Operators

11:30-12:00: Physics Driven Lab 1: Solve Partial Differential Equations using PINN approach

12:00-13:00: Launch Break

13:00-13:50: Physics Driven Lab 1: Solve Partial Differential Equations using PINN approach

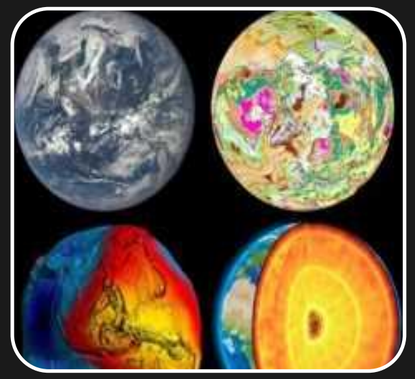
14:00-14:50: Physics Driven Lab 2: Solve transient problems and inverse problems

15:00-16:00: Physics Driven Challenge: Solve a fluid mechanics problem

COMPUTATIONAL DOMAINS

Computational Eng.

Solid & Fluid Mechanics, Electromagnetics, Thermal, Acoustics, Optics, Electrical, Multi-body Dynamics, Design Materials

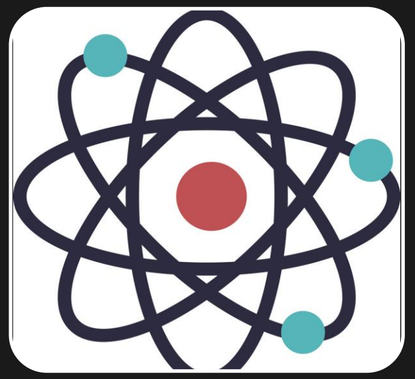


Earth Sciences

Climate Modeling, Weather Modeling, Ocean Modeling, Seismic Interpretation

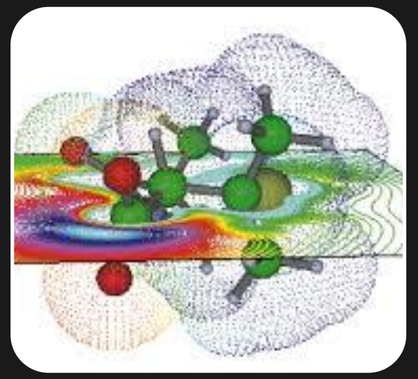
Life Sciences

Genomics, Proteomics



Computational Physics

Particle Science, Astrophysics

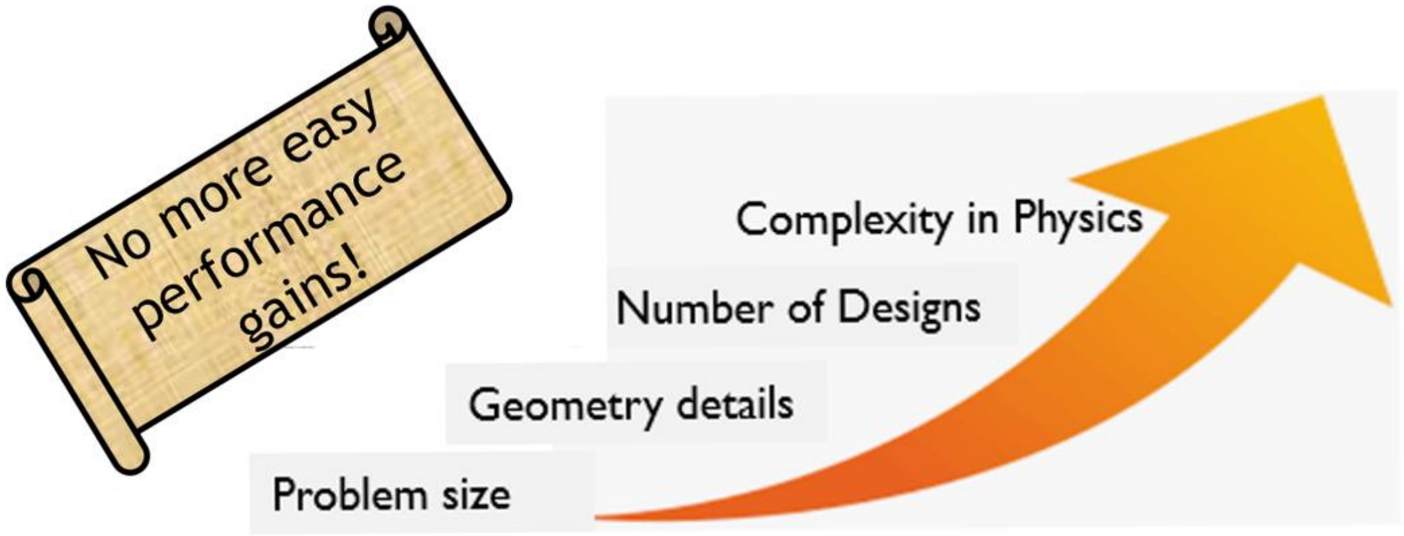


Computational Chemistry

Quantum Chemistry, Molecular Dynamics

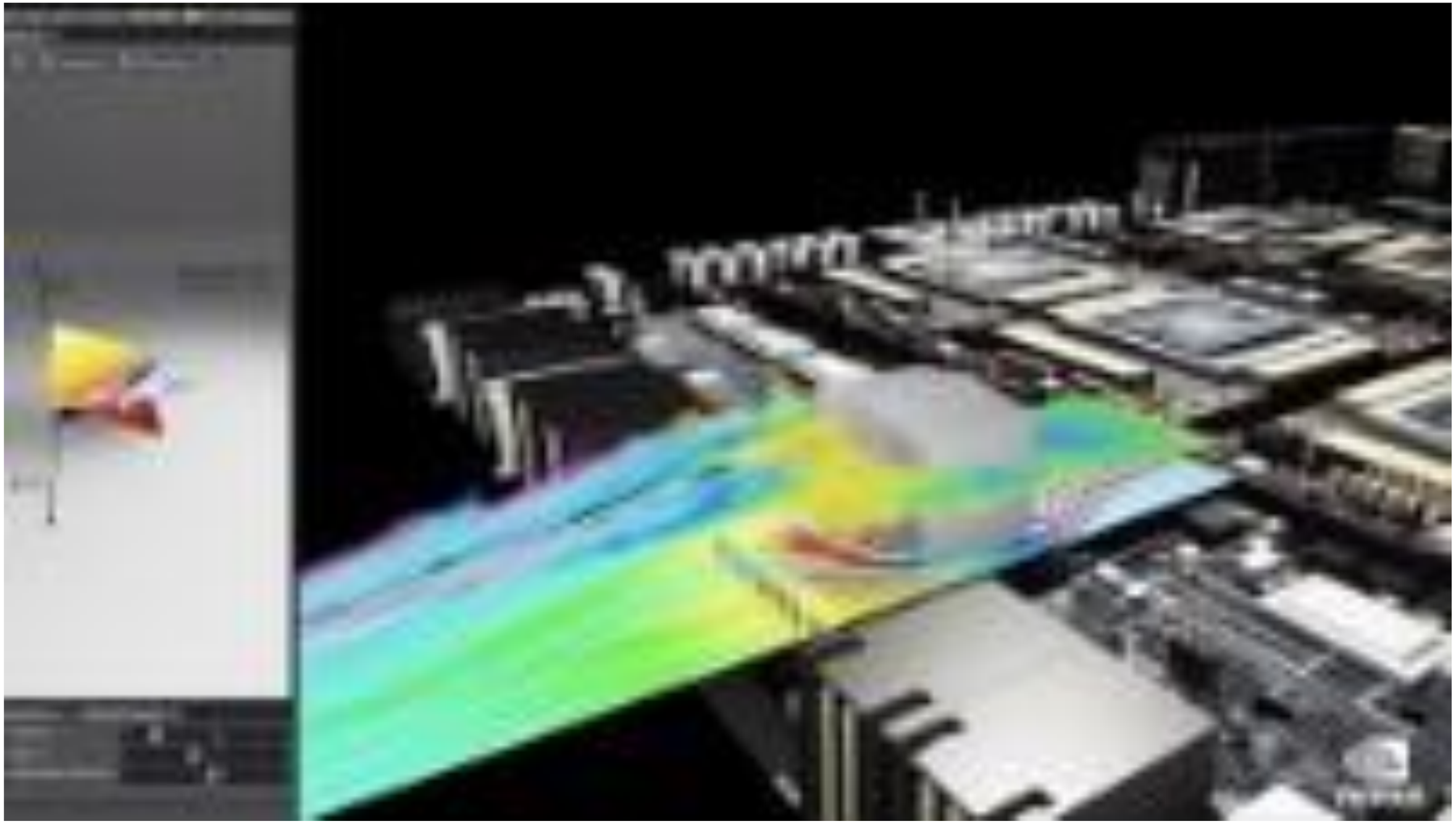
SATURATING PERFORMANCE IN TRADITIONAL HPC

Simulations are getting larger & more complex



Traditional simulation methods are:

- Computationally expensive
- Demand ever-increasing resolution
- Plagued by domain discretization techniques
- Not suitable for data-assimilation or inverse problems

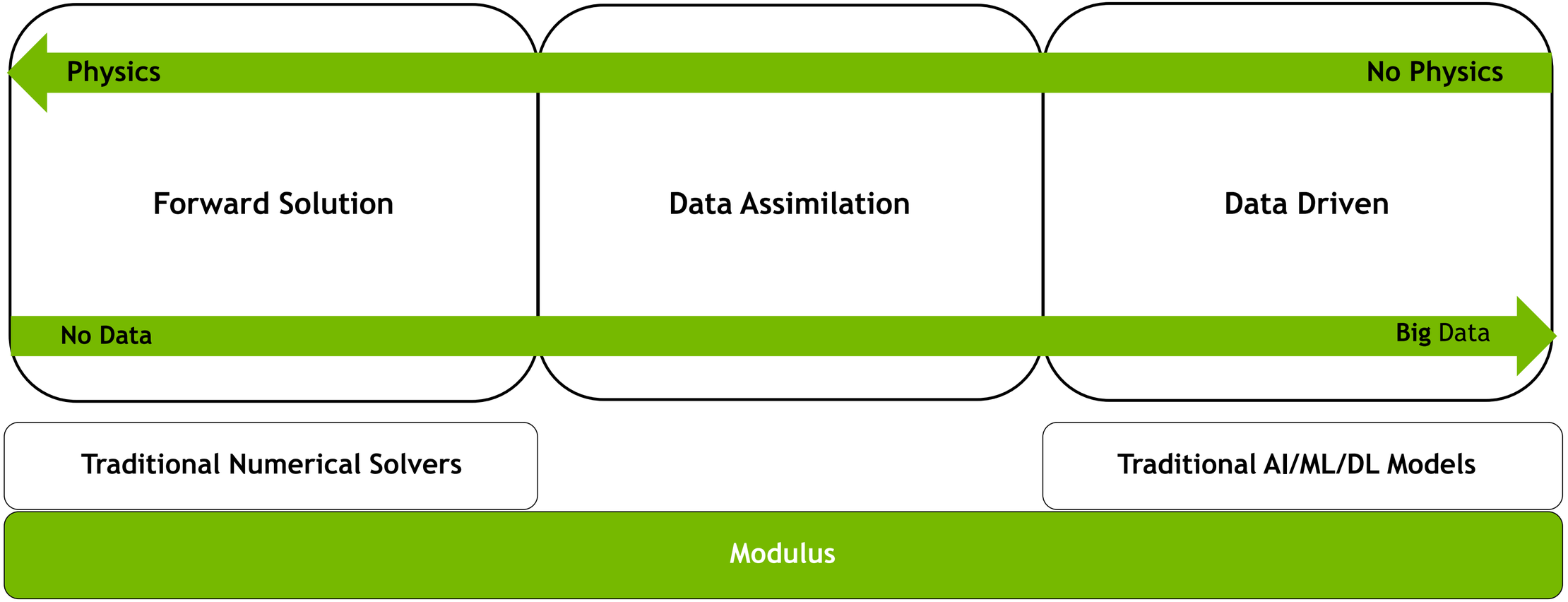


https://www.nvidia.com/en-us/on-demand/session/gtcfall21-d31037/

https://www.nvidia.com/en-us/on-demand/session/gtcfall21-d31038/

AI IN COMPUTATIONAL SCIENCES

Primary Driver: Data vs. Physics



AI/ML FOR PHYSICAL SYSTEMS

What is different between Modulus and Traditional AI Models?

Think of Blood flow in an aneurysm

- Use the underlying knowledge of the system
  - Confine the training space
  - Reduce the size of data (experimental data is expensive)
- CV based AI approaches are not good enough - why?
  - These networks don’t work – don’t converge
    - Not appropriate architectures
    - generally biased toward low-frequency solutions, a phenomenon known as "spectral bias"

NVIDIA Modulus

Solving PDEs with neural networks

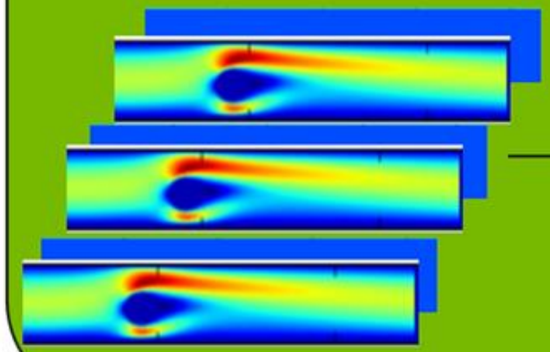
https://docs.nvidia.com/deeplearning/modulus/index.html

A **Data Driven Neural Network** requires training data

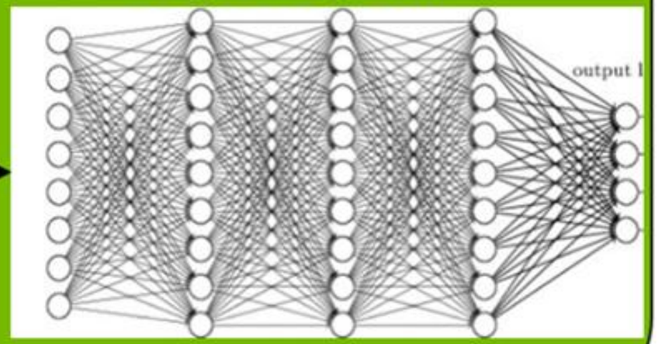
A **Physics Driven Neural Network** solver does **NOT** require training data

Supervised Data Driven Approach

Simulation Dataset



Neural Network



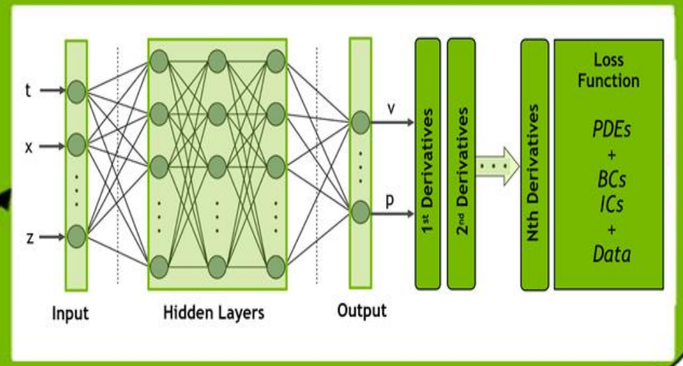
N layers

Unsupervised Physics Driven Approach

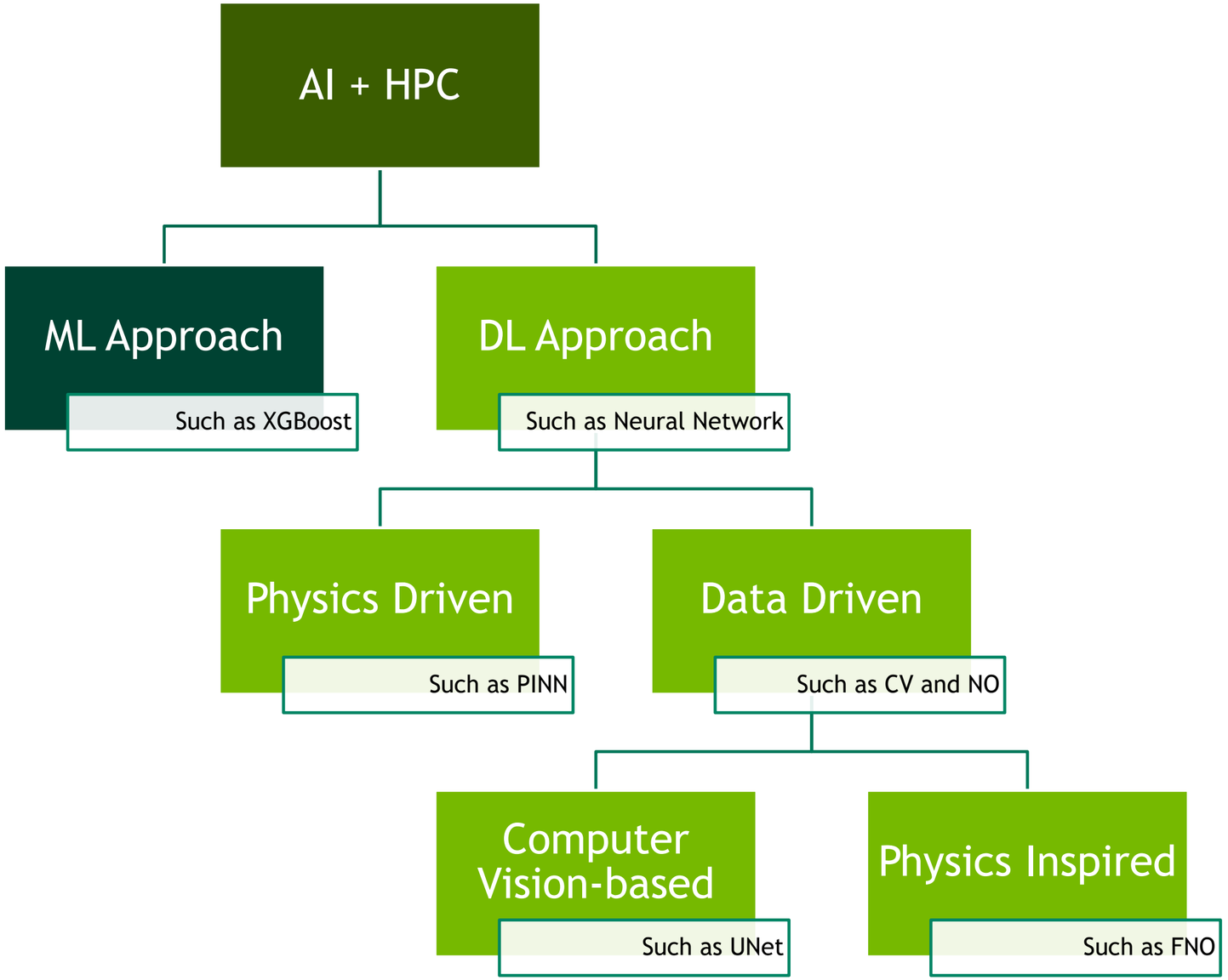
PDE and Boundary Conditions

$$0 = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$
$$0 = u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$$
$$0 = u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \nu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2})$$

Neural Network



m*N layers (for mth order PDE)

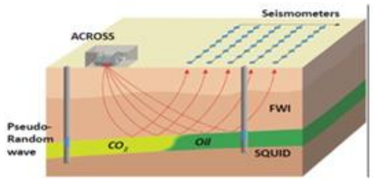

AI + HPC
ML Approach
Such as XGBoost
DL Approach
Such as Neural Network
Physics Driven
Such as PINN
Data Driven
Such as CV and NO
Computer Vision-based
Such as UNet
Physics Inspired
Such as FNO

AI ENABLING NEXT GENERATION SIMULATION

Single Simulation

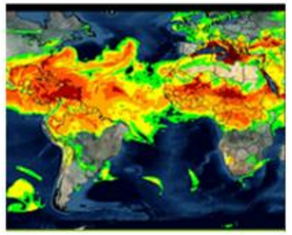
Multiple Simulations

Inverse & Data Assimilation Problems



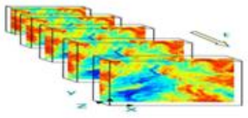
Oil & Gas

Medical Imaging

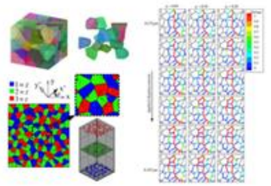


Climate

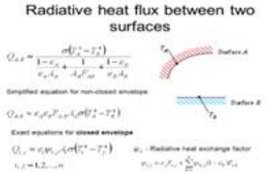
Improved Physics & Predictions



Turbulence



Micro-mechanical Material Model



Radiation

Real Time Simulations

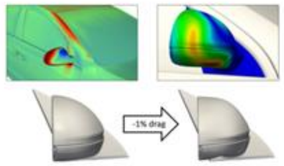
Robotics

Autonomous Ride & Handling

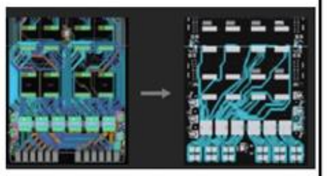
Digital Twin

Digital Design & Manufacturing

Heat Sink



Aerodynamics

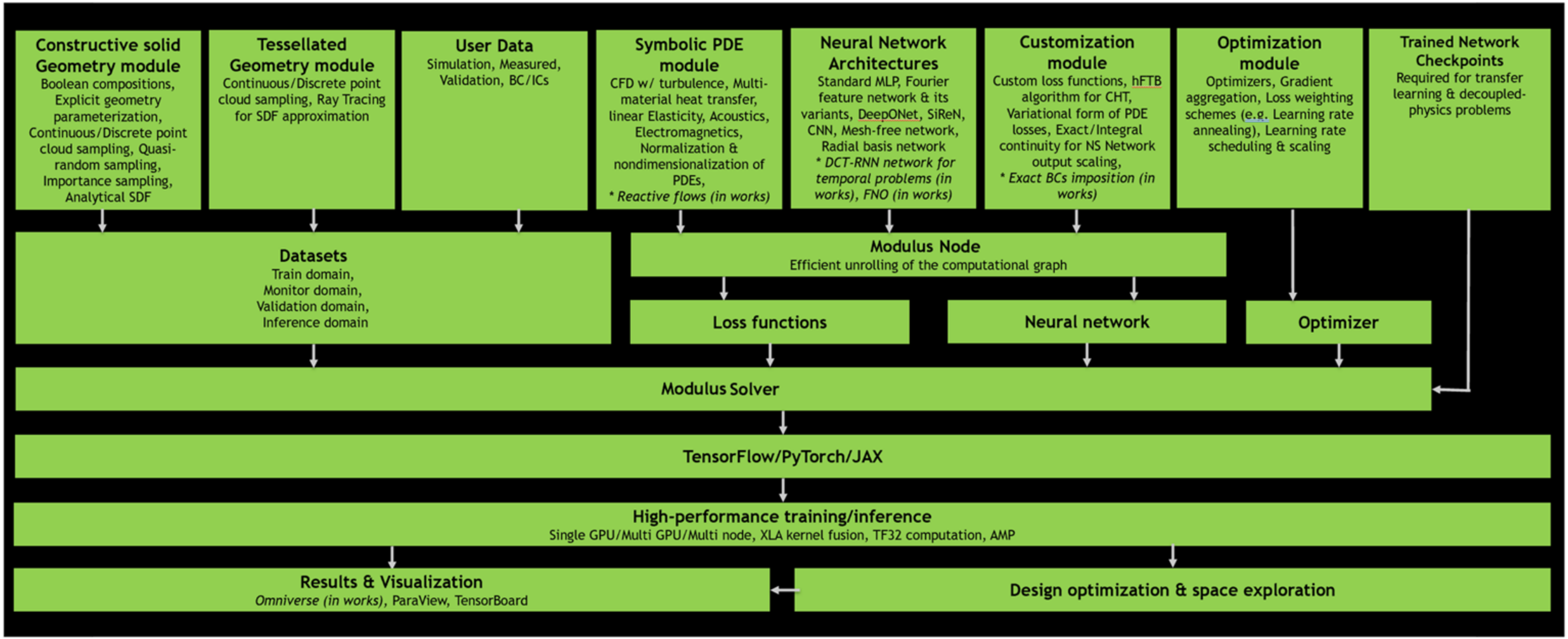


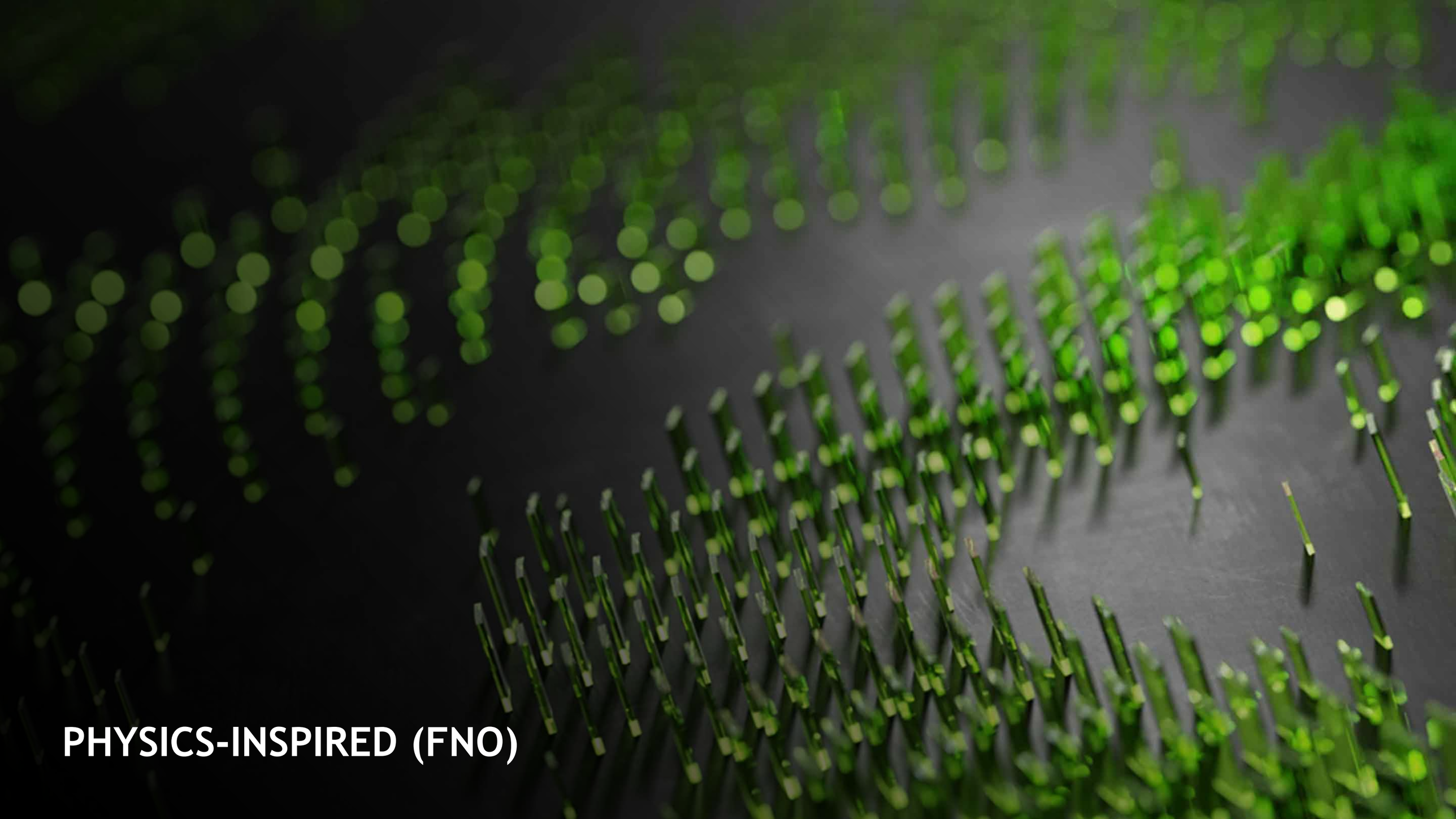
Vias on a PCB

Physics & Data - No Traditional Solver

Physics - Traditional Solver (Speed is a limitation)

Modulus code flow




PHYSICS-INSPIRED (FNO)

MILLION-X CLIMATE SCIENCE

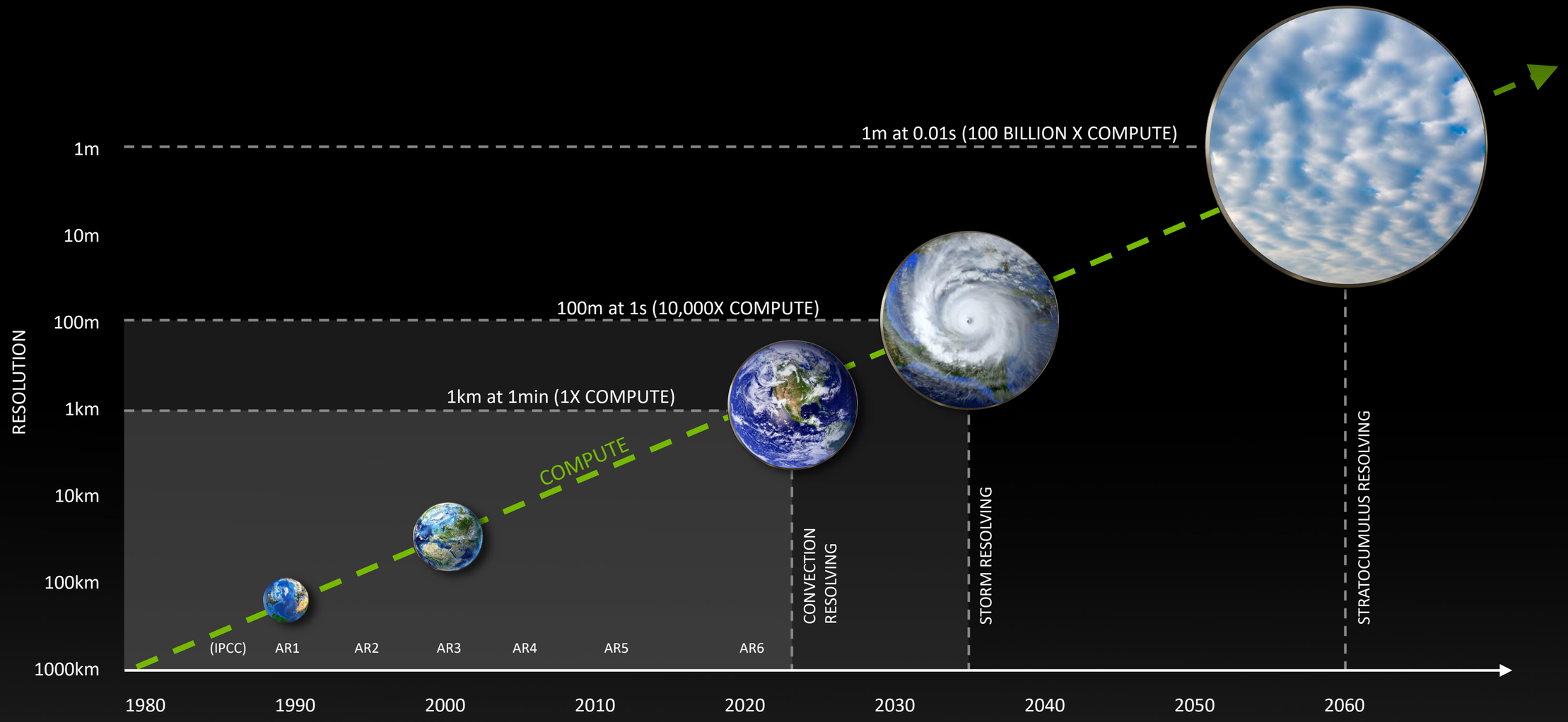
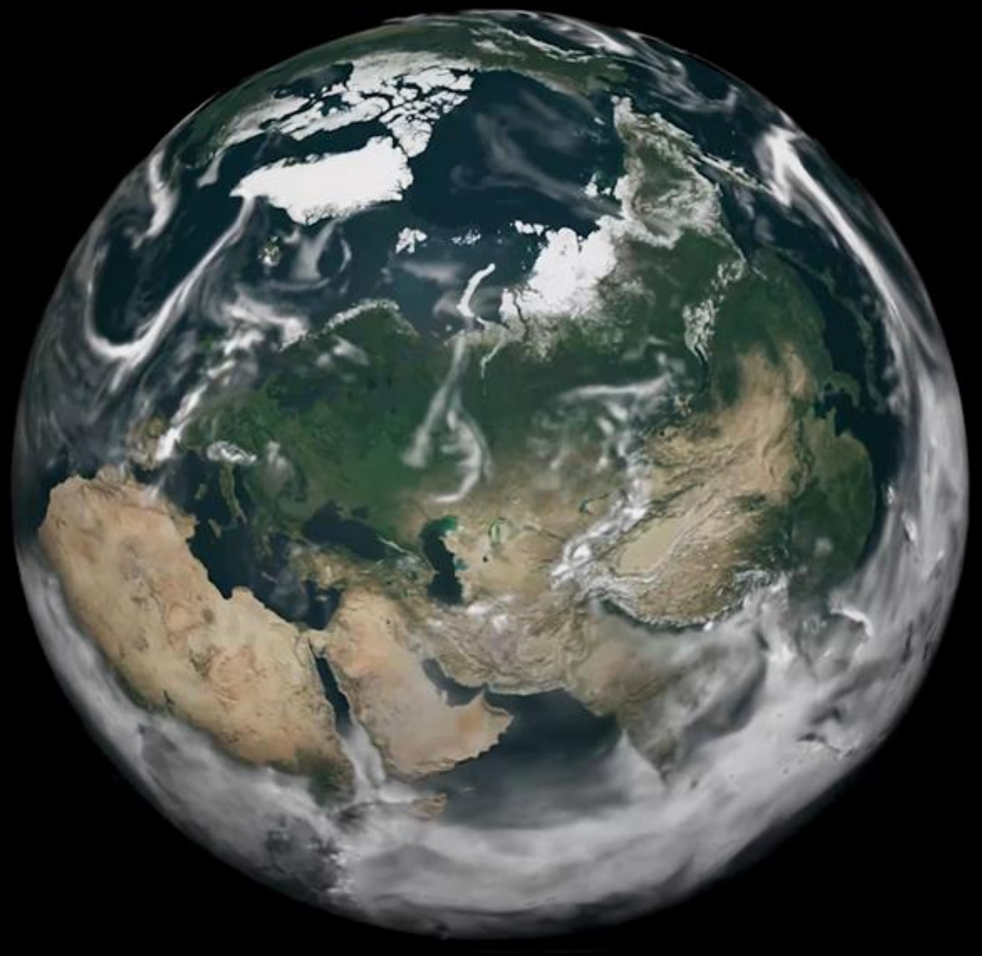


Figure adapted from: Schneider, T., Teixeira, J., Bretherton, C. *et al.* Climate goals and computing the future of clouds. *Nature Clim Change* **7,** 3–5 (2017). https://doi.org/10.1038/nclimate3190



ACCELERATING EXTREME WEATHER PREDICTION WITH FourCastNet IN NVIDIA MODULUS

Use Case

- Climate change is making storms both stronger and less predictable, leading to more fires, floods, heatwaves, mudslides, and droughts.
- Predicting global weather patterns and extreme weather events, like atmospheric rivers, is important to quantify any catastrophic event with confidence.

Challenges

- To develop the best strategies for mitigation and adaptation, we need climate models that can predict the climate in different regions of the globe over decades.

Solution

- NVIDIA has created a physics-ML model that emulates the dynamics of global weather patterns and predicts extreme weather events, like atmospheric rivers, with unprecedented speed and accuracy.

NVIDIA Solution Stack

- Hardware: NVIDIA A100
- Software: NVIDIA Omniverse, NVIDIA Modulus

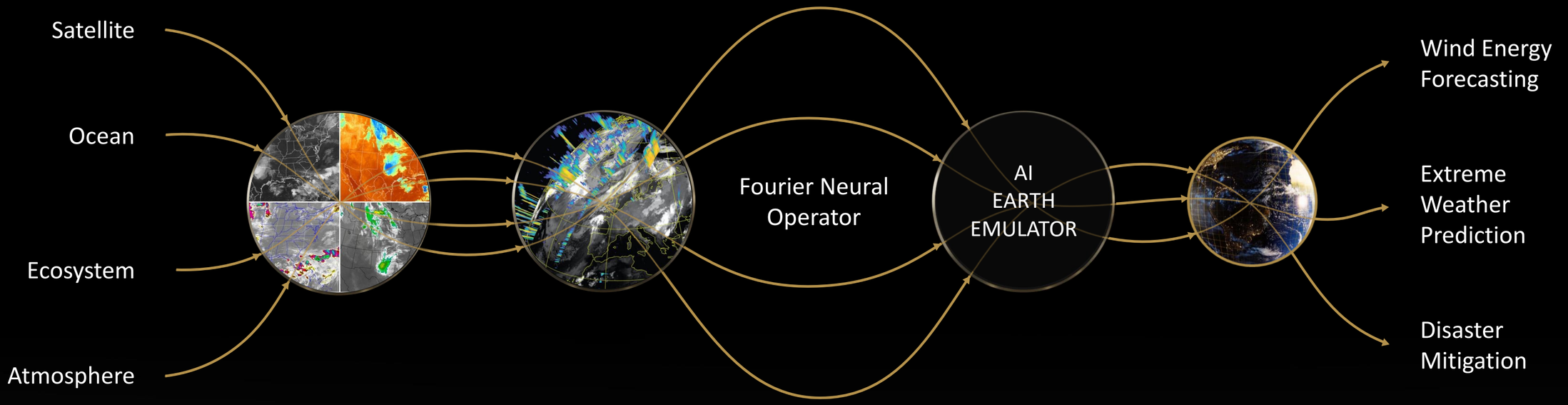
Outcome

- Powered by the Fourier Neural Operator, this GPU-accelerated AI-enabled digital twin, called FourCastNet, is trained on 10 TB of Earth system data.
- Using this data, together with NVIDIA Modulus and Omniverse, we are able to forecast the precise path of catastrophic atmospheric rivers a full week in advance.

Demo

NVIDIA

EARTH DIGITAL TWIN IN OMNIVERSE



ERA5 ECMWF
Atmospheric Winds & Geopotential
10 TB | 30km | 5 Atmos Layers

100,000X Speed-Up
2 Seconds for 7-Day Forecast
Training: 16 Hours on 64 A100 GPUs

Digital Twin
Interactivity & Verification

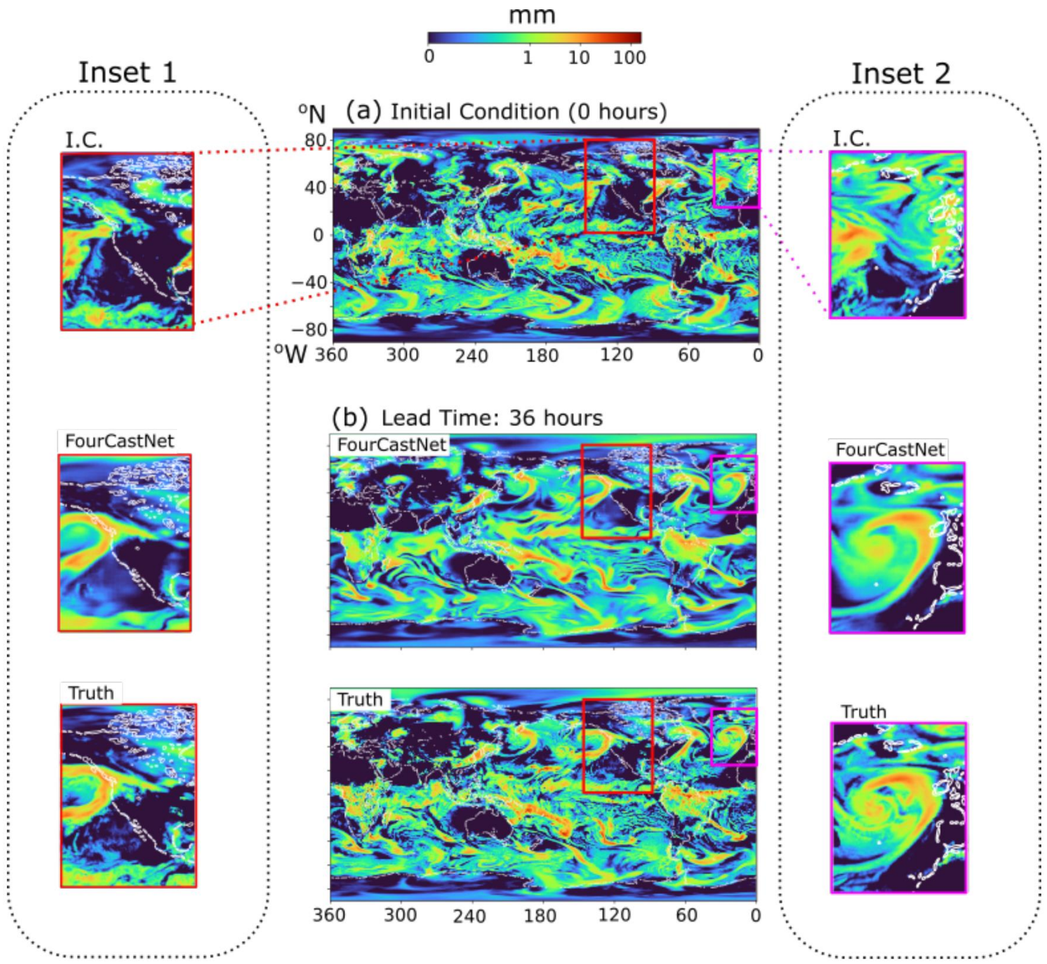
RAPIDS — Modulus — Omniverse

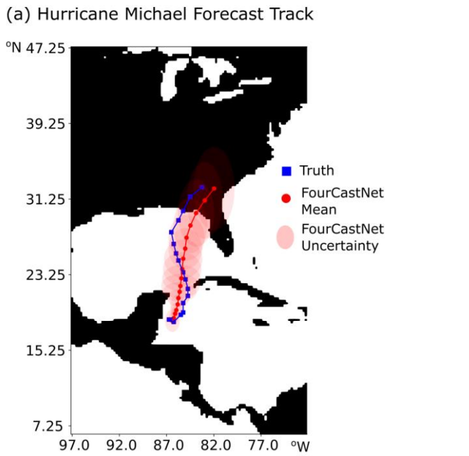
DATA PROCESSING **AI-SIMULATION** **VISUALIZATION**

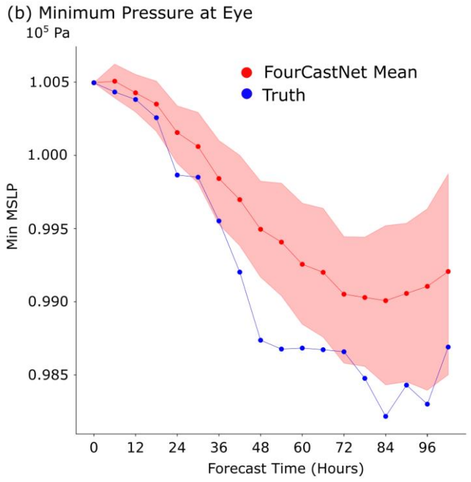
FOURCASTNET: A GLOBAL DATA-DRIVEN HIGH-RESOLUTION WEATHER MODEL USING ADAPTIVE FOURIER NEURAL OPERATORS

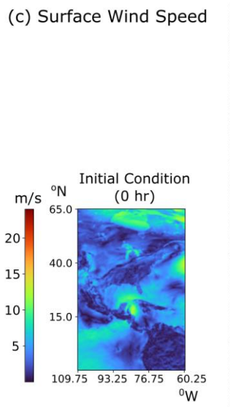
https://arxiv.org/abs/2202.11214

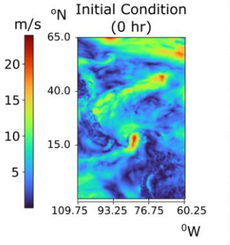
NVIDIA, Lawrence Berkley National Lab, University of Michigan, Rice University, Caltech, Purdue University

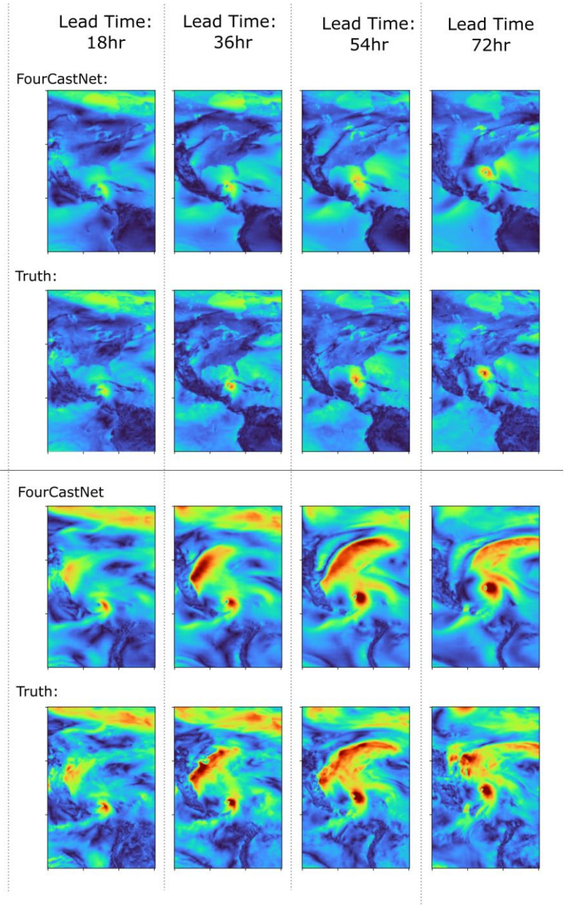








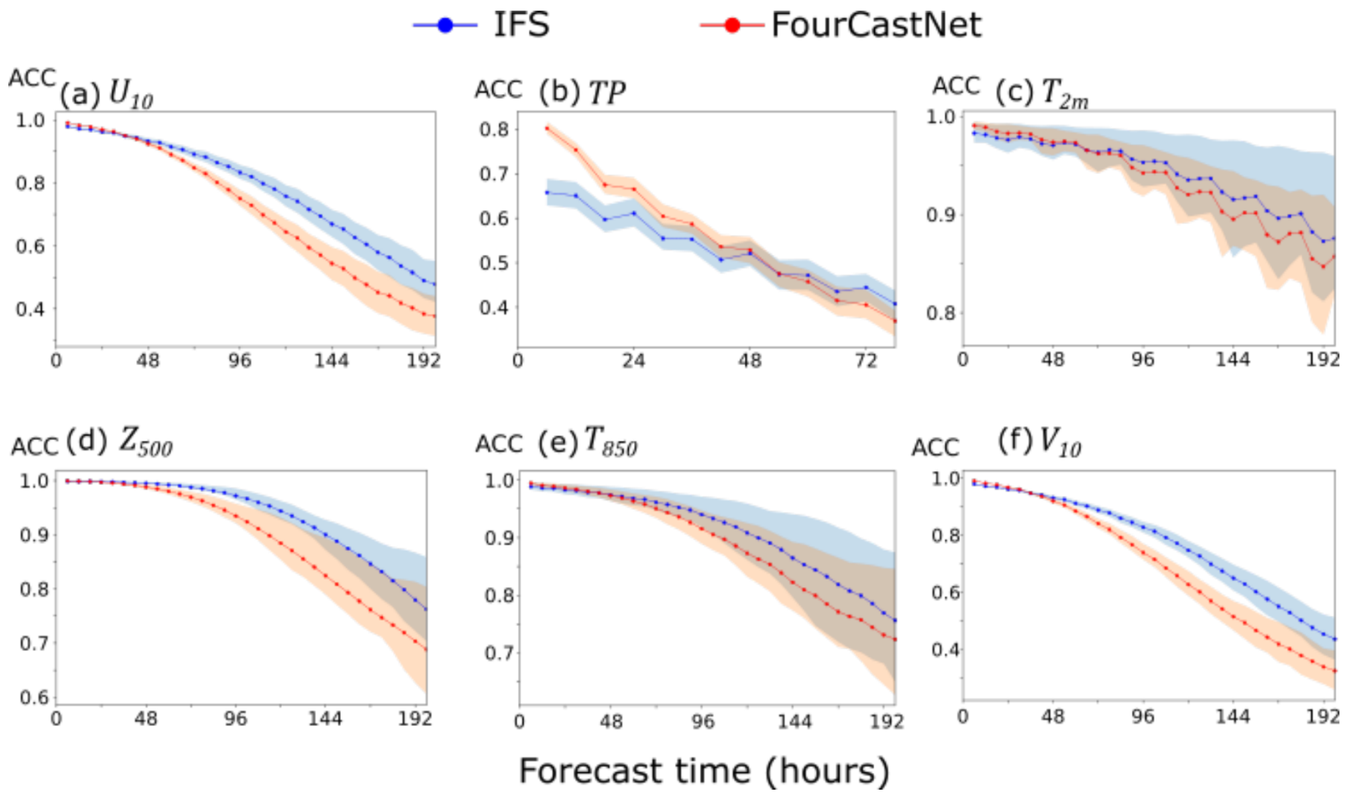




FOURCASTNET: A GLOBAL DATA-DRIVEN HIGH-RESOLUTION WEATHER MODEL USING ADAPTIVE FOURIER NEURAL OPERATORS

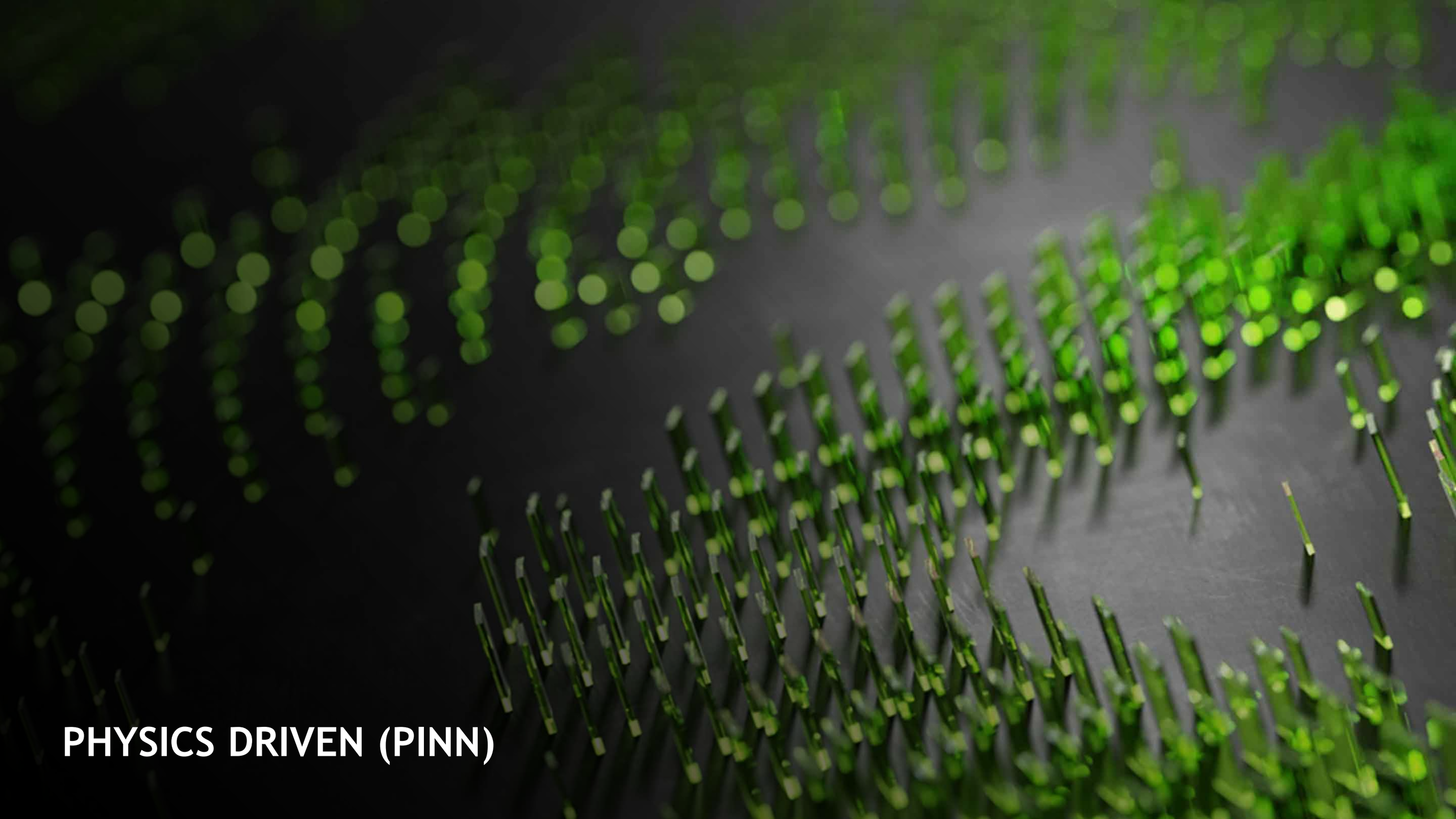
https://arxiv.org/abs/2202.11214

NVIDIA, Lawrence Berkley National Lab, University of Michigan, Rice University, Caltech, Purdue University



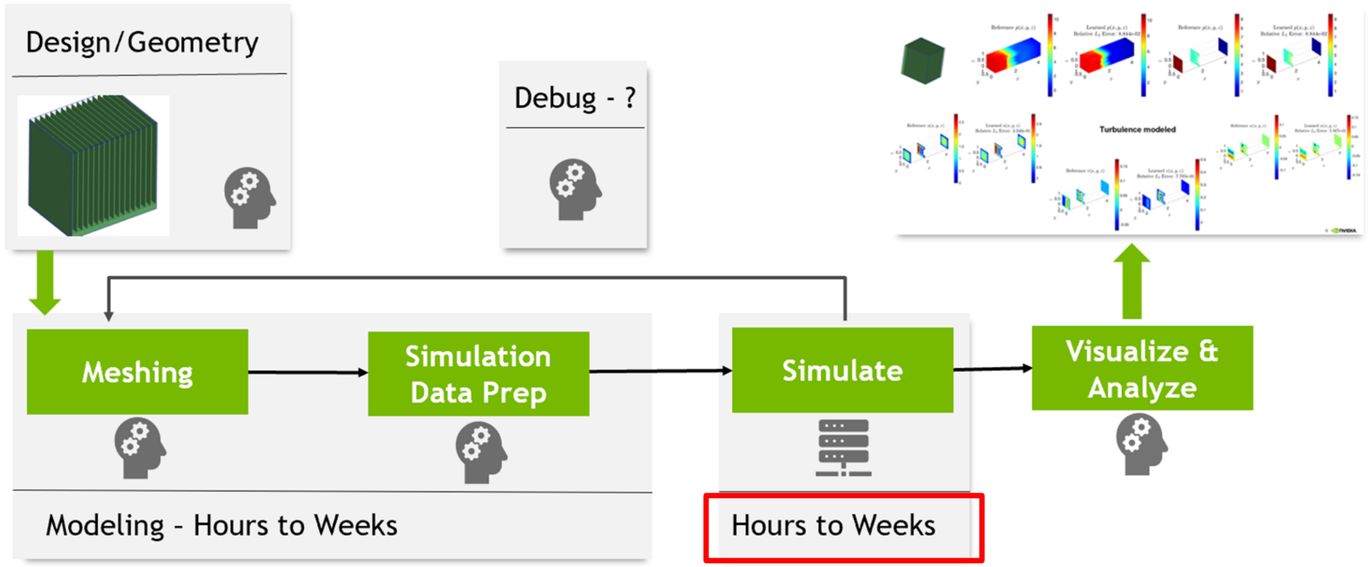
Latency and Energy consumption for a 24-hour 100-member ensemble forecast				
	IFS	FCN - 30km (actual)	FCN - 18km (extrapolated)	IFS / FCN(18km) Ratio
Nodes required	3060	1	2	**1530**
Latency (Node-seconds)	984000	7	22	**44727**
Energy Consumed (kJ)	271000	7	22	**12318**

NVIDIA

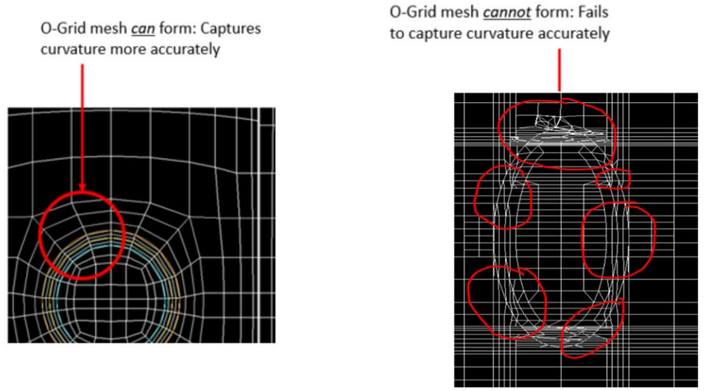


PHYSICS DRIVEN (PINN)

ISSUES IN TRADITIONAL SIMULATIONS



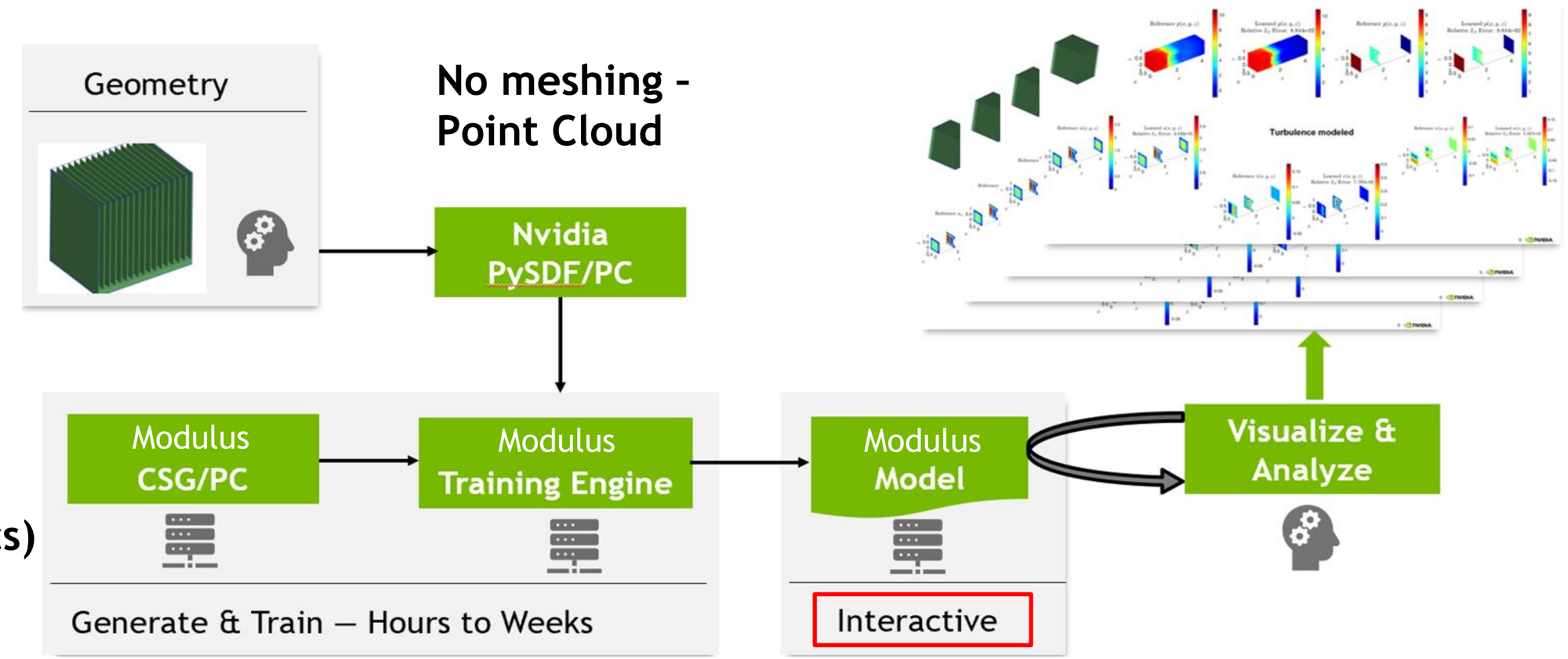
- **Meshing Accuracy**



- **Limited Scope & Applicability**
- **Computational Speed Bottleneck**

MODULUS - PHYSICS INFORMED NEURAL NETWORK

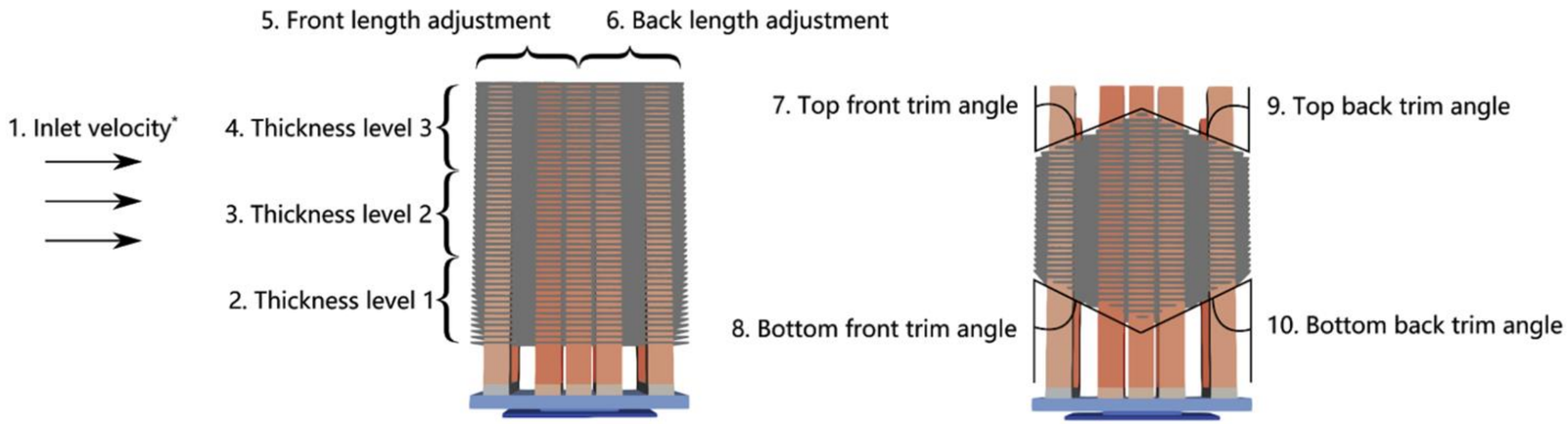
Parameterization (Geometry/Physics)



PARAMETERIZED A100 NVSWITCH HEAT SINK

Optimization with 10 Design Parameters

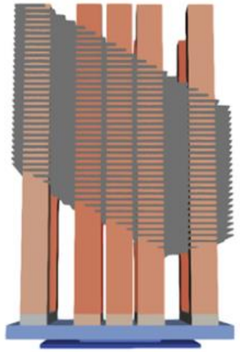
Limerock Design Variables



*Inlet velocity is not a design variable. It will be used for robust design optimization of Limerock in future.

Limerock Optimal Design

1. Inlet velocity: 5.7 m/s
2. Thickness level 1: 0.0031
3. Thickness level 2: 0.0044
4. Thickness level 3: 0.0030
5. Front length adjustment: -0.0124
6. Back length adjustment: 0.0025
7. Top front trim angle: 0.0223 rad
8. Bottom front trim angle: 0.5197 rad
9. Top back trim angle: 0.5147 rad
10. Bottom back trim angle: 0.2217 rad

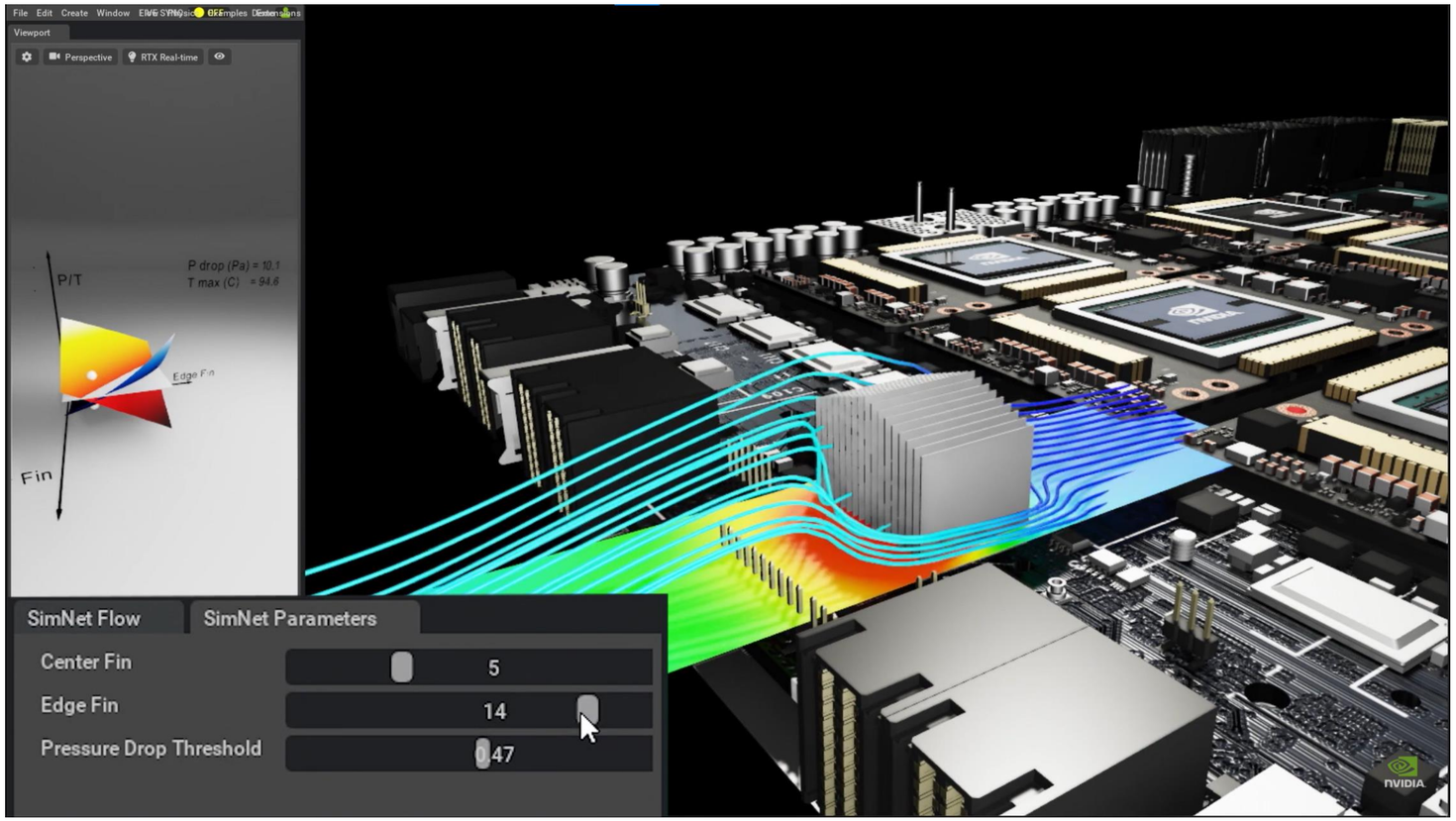


Max allowed pressure drop: 2.59
Number of random design evaluations: 4M

Peak temperature: 38.25 degC
Pressure drop: 2.5896

A100 NVSWITCH HEAT SINK

Multi-Physics Application: Fluids + Heat Transfer



https://www.youtube.com/watch?v=Oq2Mpi5pF1w&ab_channel=NVIDIADeveloper

Turbulent Flow (Re=19,000)		
	Temperature	**Pressure Drop**
SimNet - Fourier Network	**43.1°C**	**4.05**
OpenFOAM (method 1)	**41.6 °C**	**3.56**
OpenFOAM (method 2)	**41.6 °C**	**4.58**

Computational Times (10 parameters, 3 values per parameter)	
Modulus	**1000 V100 GPU hrs.**
~=18000x speedup	
Traditional Solver (OpenFOAM) 59,049 separate runs (26 wall hours on 12 CPU cores)	**18.4 Million CPU hrs.**

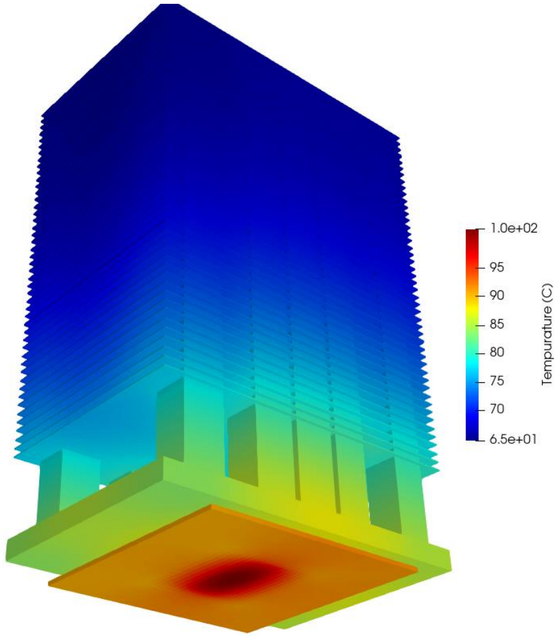
IMPROVED PREDICTIONS WITH MODULUS

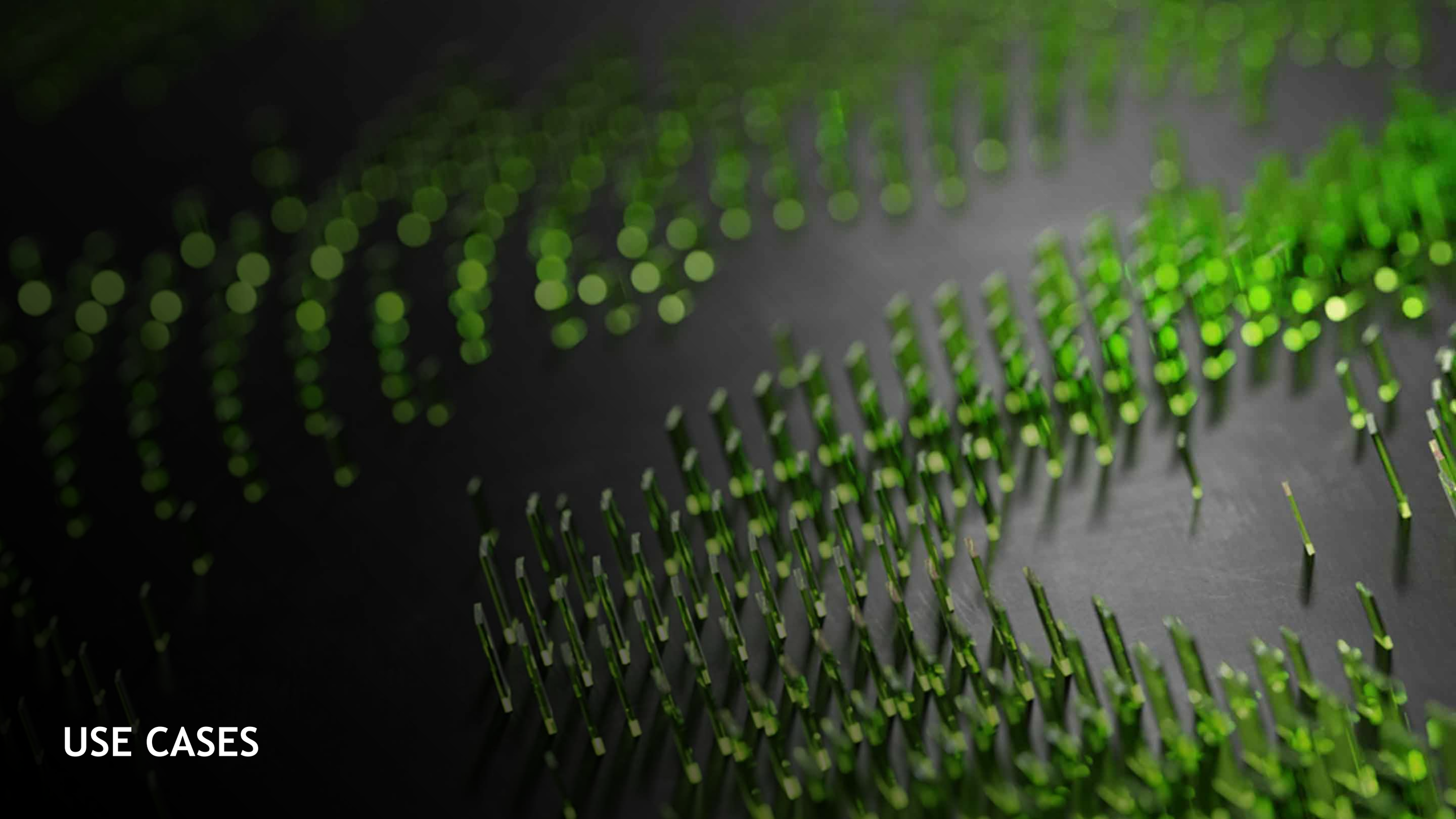
https://docs.nvidia.com/deeplearning/modulus/user_guide/advanced/industrial_heat_sink.html#results-and-post-processing
not only faster but also more accurate

Table 14 Commercial solver mesh refinement results for NVSwitch pressure drop and peak temperature.

Number of elements	Pressure drop (Pa)			Peak temperature ($^{\circ}C$)		
	Commercial solver	Modulus	% diff	Commercial solver	Modulus	% diff
22.4 M	81.27	150.25	84.88	97.40	97.35	0.05
24.7 M	111.76	150.25	34.44	95.50	97.35	1.94
26.9 M	122.90	150.25	22.25	95.10	97.35	2.36
30.0 M	132.80	150.25	13.14	•	•	•
32.0 M	137.50	150.25	9.27	•	•	•

≈1.4x

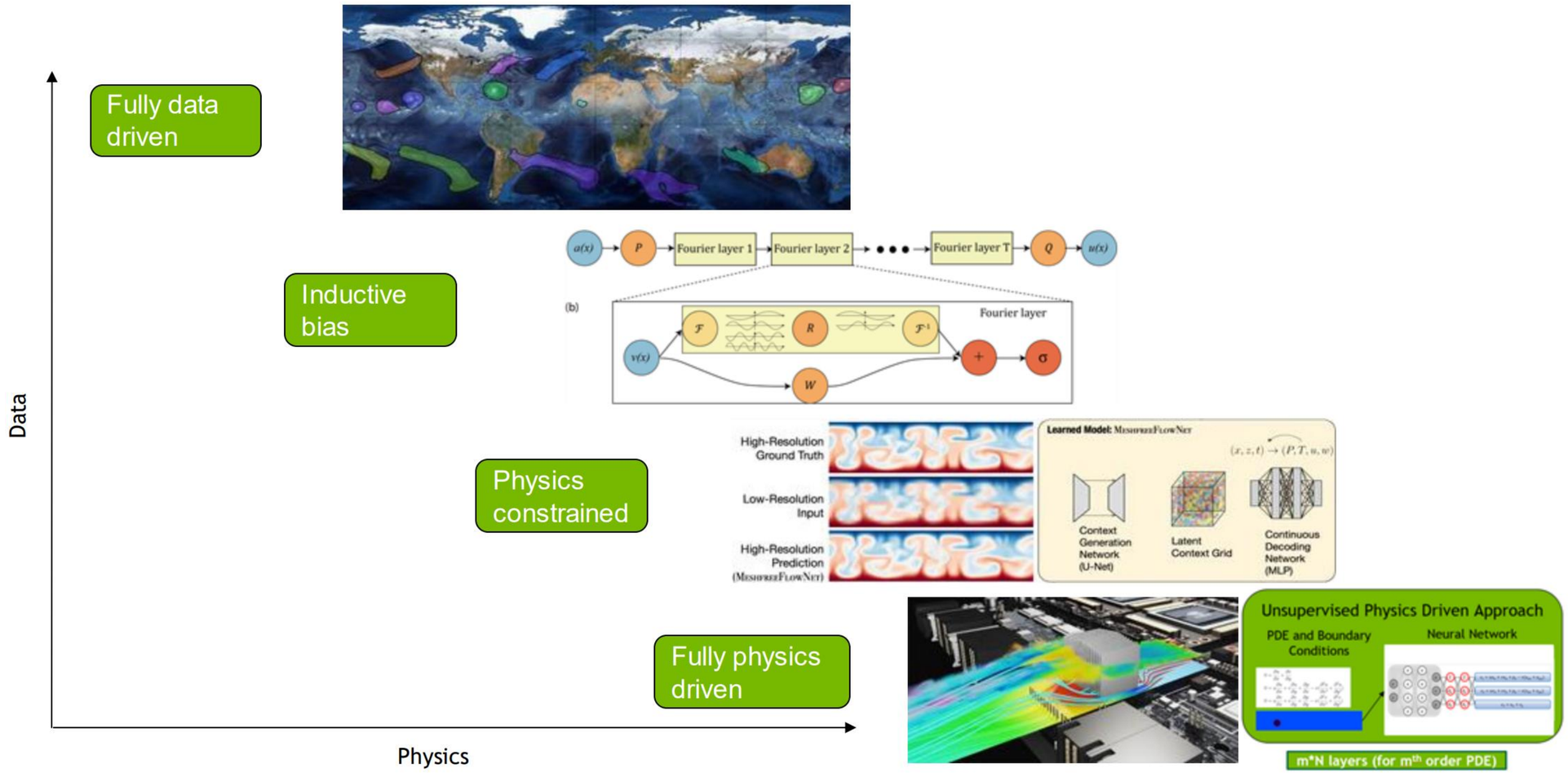




USE CASES

NVIDIA MODULUS TEAM @ GTC 22

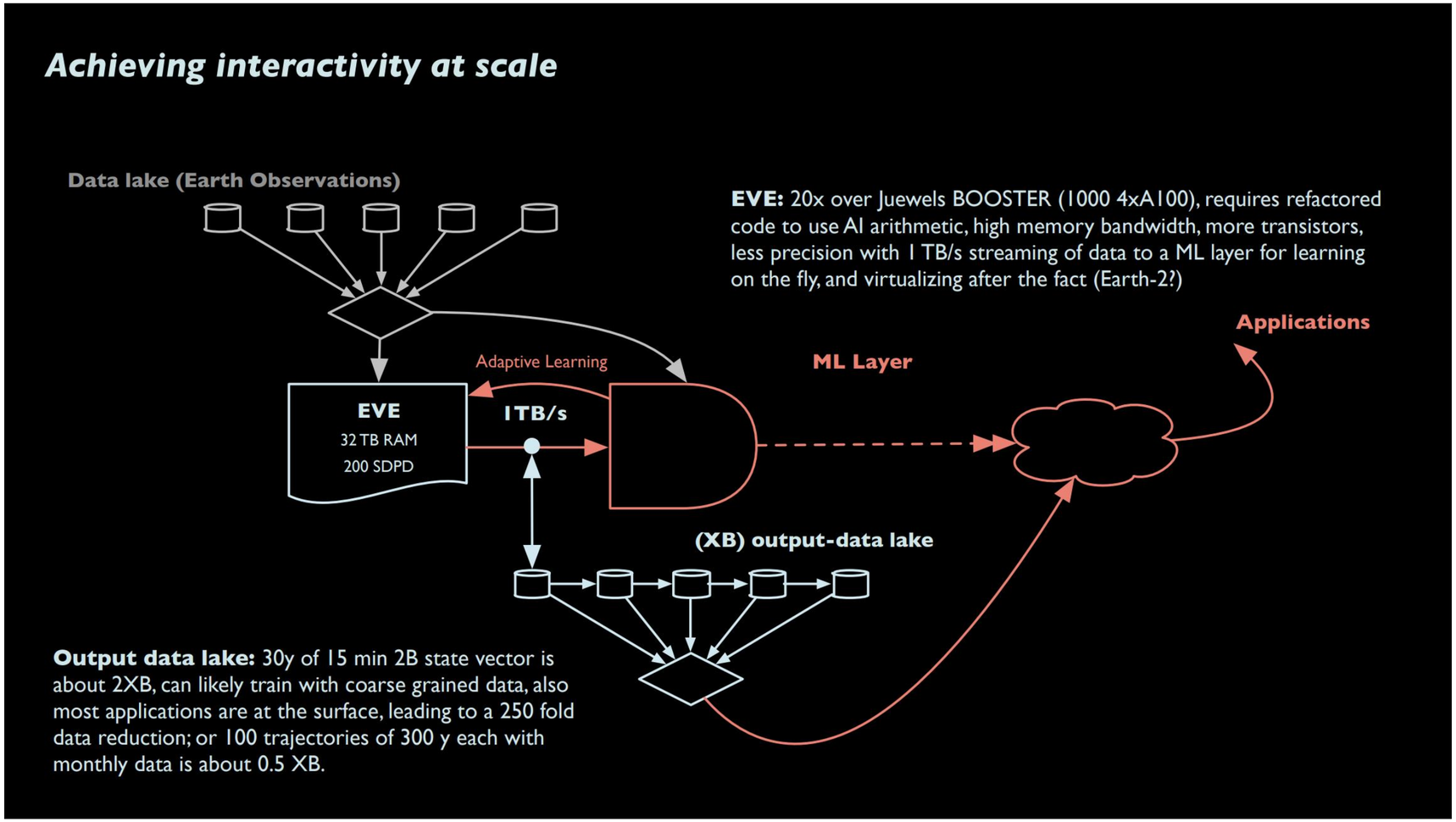
https://www.nvidia.com/en-us/on-demand/session/gtcspring22-s41823/



MAX PLANCK INSTITUTE FOR METEOROLOGY @ GTC 22

https://www.nvidia.com/en-us/on-demand/session/gtcspring22-s41950/

An Earth Virtualization Engine (EVE) - by Prof. Bjorn Stevens



DIGITAL TWINS FOR POWER PLANT BOILERS — BATTELLE, NETL

GTC 22 Spring Webinar

Use Case

- Using digital twins to accelerate the design and development cycle of a power plant boiler and enable effective carbon capture and storage

Challenges

- During the power plant development process, many techniques are used to design robust carbon management
  - This requires complex simulations of fluid flow mechanics, heat transfer, and chemical reactions.
- Conventional modeling techniques like computational fluid dynamics (CFD) are far too slow for large-scale design and optimization.

Solution

- Using NVIDIA's Physics Informed Neural Network (PINNs) and Modulus, National Energy Technology Lab developed a digital twin of a boiler capable of modeling turbulent reacting flows faster than CFD methods.

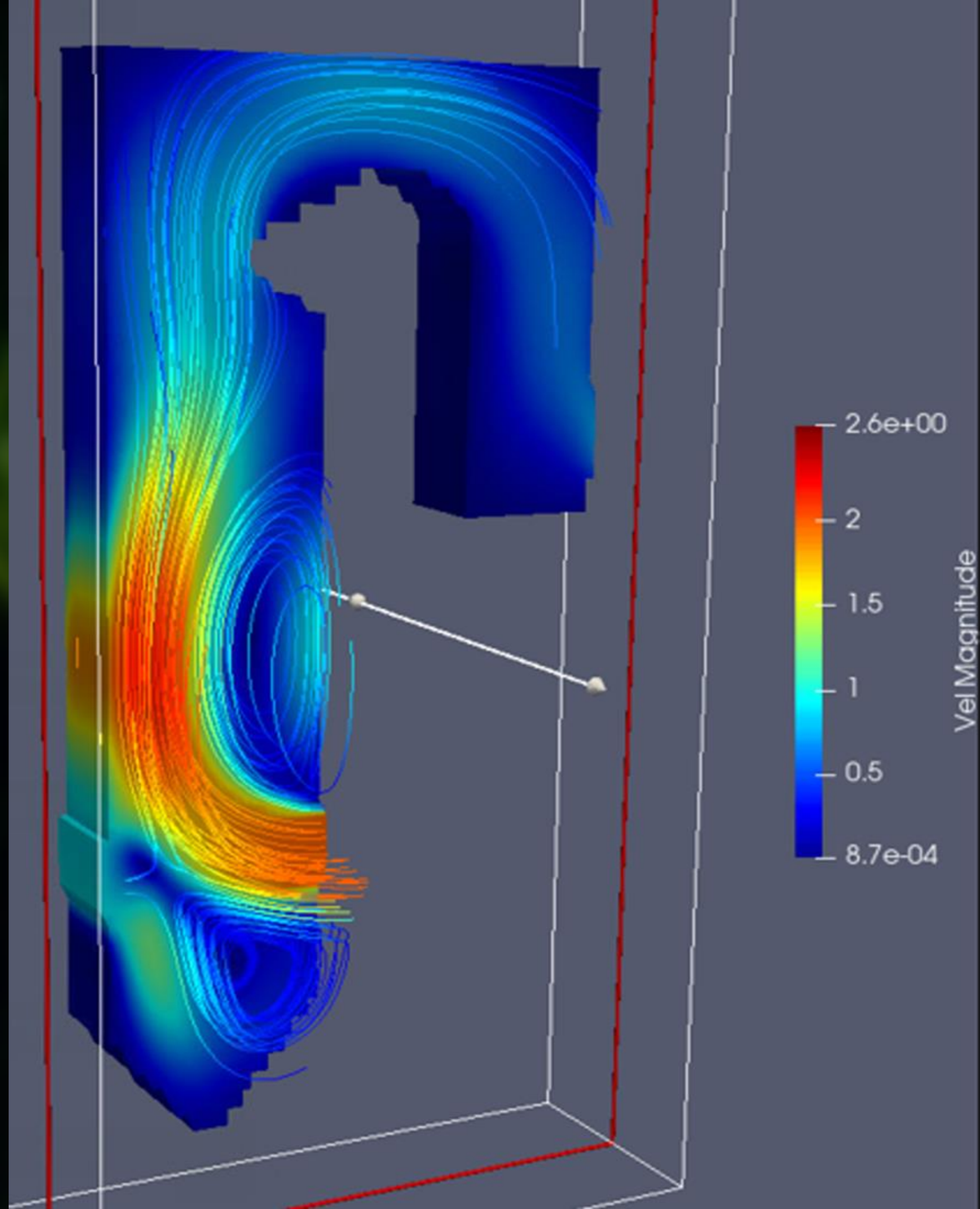
NVIDIA Solution Stack

- Hardware: NVIDIA V100, A100 GPUs
- Software: CUDA 10.2, NVIDIA Modulus

Outcome

- NETL achieved predictions in orders of magnitude less time than traditional CFD methods while maintaining very high accuracy.

NVIDIA BATTELLE NETL NATIONAL ENERGY TECHNOLOGY LABORATORY



HRSG FLUID ACCELERATED CORROSION SIMULATION — SIEMENS ENERGY

Use Case

- Detecting and predicting point of corrosion in heat recovery steam generators (HRSGs)

Challenges

- Using standard simulation to detect corrosion, it took SE at least couple of weeks, and the overall process took 14-16 weeks for every HRSG unit.

Solution

- Using NVIDIA Modulus Physics-Informed Neural Network, SE simulates the corrosive effects of heat, water and other conditions on metal over time to fine-tune maintenance needs.
- SE can replicate and deploy HRSG plant digital twins worldwide with NVIDIA Omniverse.

NVIDIA Solution Stack

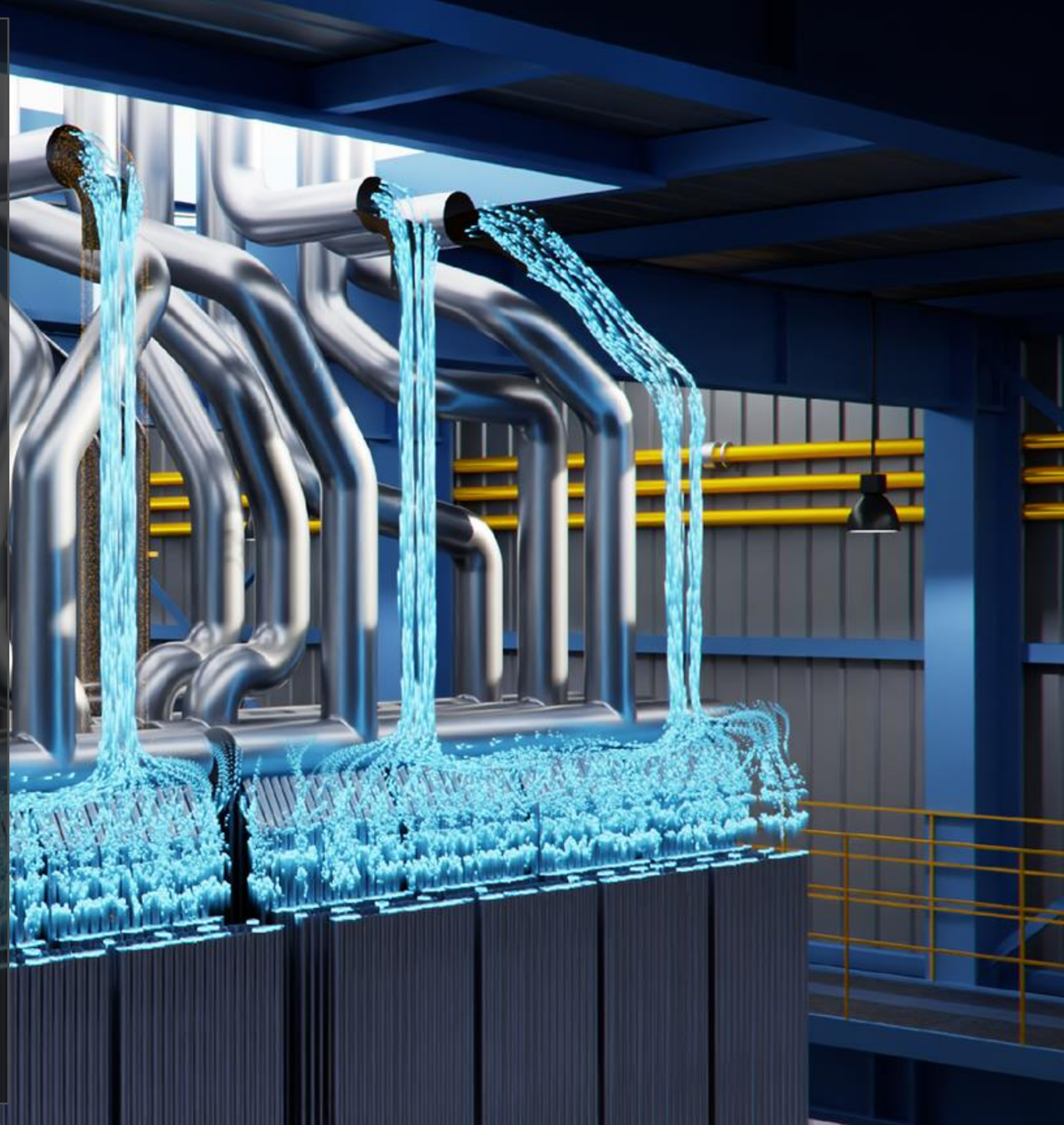
- Hardware: NVIDIA V100 & A100 Tensor Core GPUs
- Software: NVIDIA Modulus, NVIDIA Omniverse

Outcome

- 10,000X speed-up and inference in seconds can reduce downtime by 70%, saving the industry $1.7 billion annually

Link to Demo

NVIDIA SIEMENS energy



WIND TURBINE WAKE OPTIMIZATION — SIEMENS GAMESA

Use Case

- Developing optimal engineering wake models to optimize wind farm layouts
- Simulating the effect that a turbine might have on another when placed in close proximity

Challenges

- Generating high-fidelity simulation data from Reynolds-averaged Navier-Stokes (RANS) or Large Eddy Simulations (LES) can take over a month to run, even on a 100-CPU cluster.

Solution

- NVIDIA Omniverse and Modulus enable accurate, high-fidelity simulations of the wake of the turbines, using low-resolution simulations as inputs and applying super resolution using AI.

NVIDIA Solution Stack

- Hardware: NVIDIA A100, A40, RTX 8000 GPUs
- Software: NVIDIA Omniverse, NVIDIA Modulus

Outcome

- Approximately 4,000X speedup for high-fidelity simulation
- Optimizing wind farm layouts in real-time increases overall production while reducing loads and operating costs.

Link to Demo

NVIDIA SIEMENS Gamesa RENEWABLE ENERGY

AGENDA – DAY 2

09:30-10:20: Introduce Modulus

10:30-11:20: Physics Inspired Lab 0: Solve a Darcy flow problem using Fourier Neural Operators

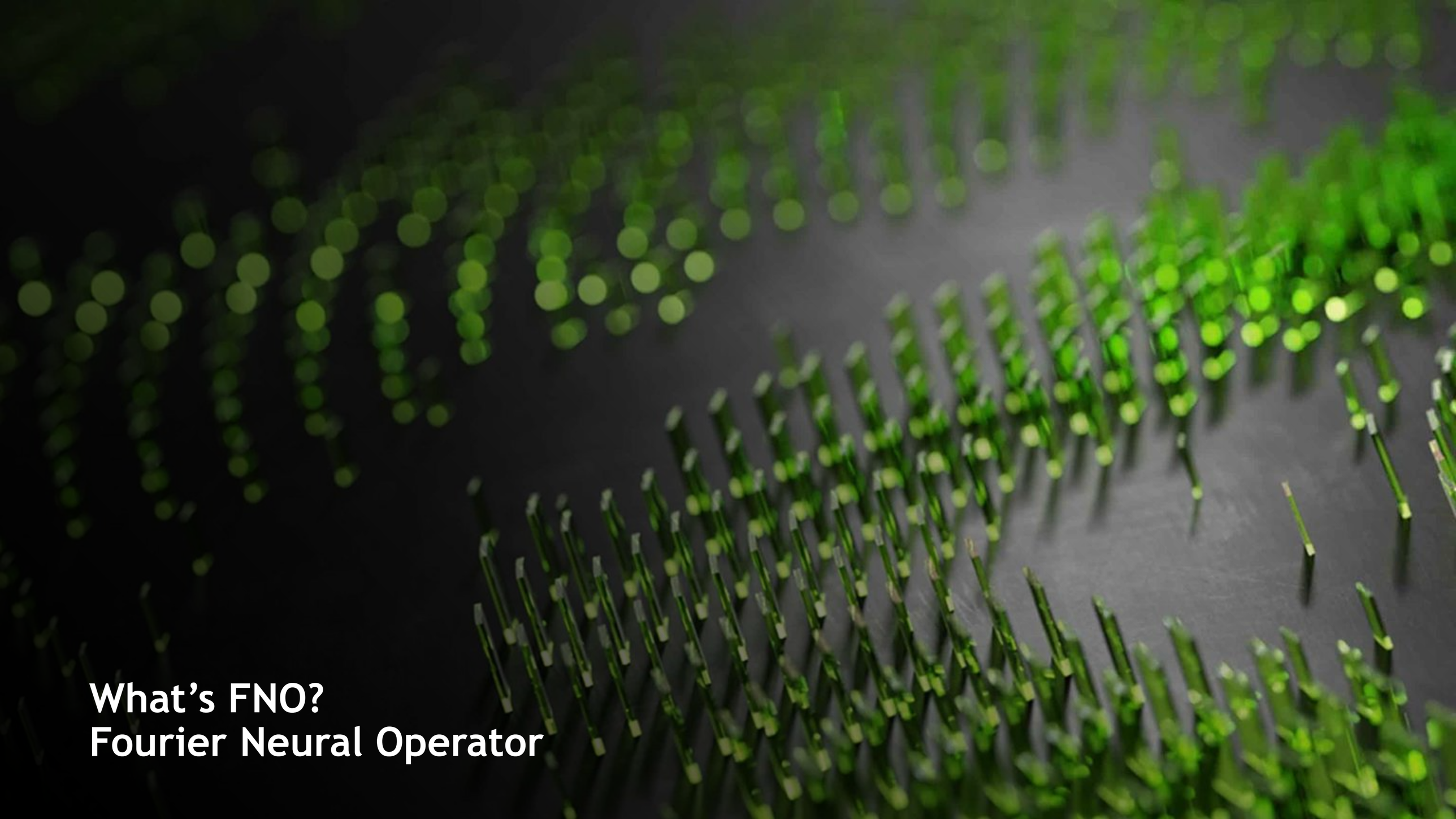
11:30-12:00: Physics Driven Lab 1: Solve Partial Differential Equations using PINN approach

12:00-13:00: Launch Break

13:00-13:50: Physics Driven Lab 1: Solve Partial Differential Equations using PINN approach

14:00-14:50: Physics Driven Lab 2: Solve transient problems and inverse problems

15:00-16:00: Physics Driven Challenge: Solve a fluid mechanics problem


What’s FNO?
Fourier Neural Operator

A NOTE: MATH BEHIND NEURAL OPERATOR

Anima Anandkumar - Neural operator: A new paradigm for learning PDEs
Neural Operator: Learning Maps Between Function Spaces
Nickola K., Zongyi L. et. al., Caltech.
https://zongyi-li.github.io/neural-operator/
Green Function

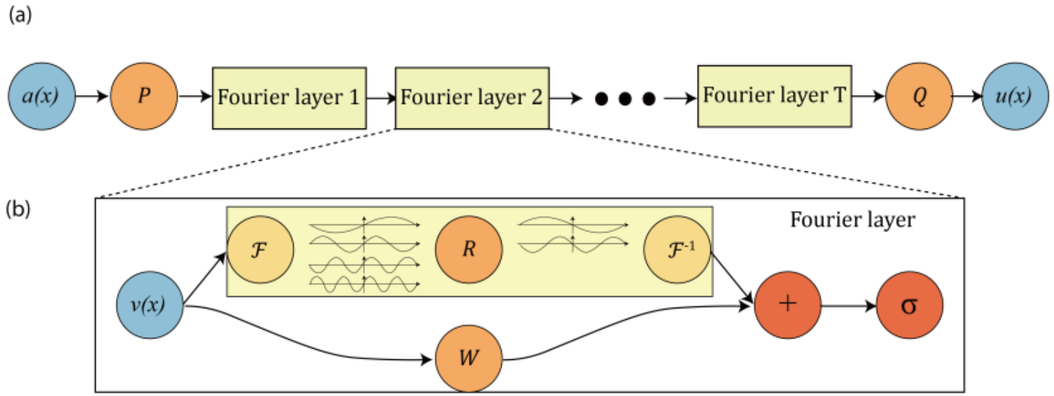
$$
\begin{aligned}
Goal: \quad & Lu = f \\
Def: \quad & L^\dagger G = \delta(x-\xi) \\
& \langle Lu, G \rangle = \langle f, G \rangle \\
& \langle u, L^\dagger G \rangle = \langle f, G \rangle \\
& \langle u, \delta(x-\xi) \rangle = \langle f, G \rangle \\
& u(\xi) = \langle f, G \rangle
\end{aligned}
$$

- Note
  - $\langle , \rangle$: function space inner product, project one function to another, which is the integral of multiplying two functions.
  - L: a linear operator.
  - $L^\dagger$: a "adjoint" operator of L, if $L = L^\dagger$ then they should has the same boundary conditions, named "self-adjoint operator".
  - δ: a `dirac function` use to `silter` function `u` at value ξ.
  - u: target solution. and ξ is just a dummy variable, actually we could simply change it here $u(\xi) \rightarrow u(x)$, we get the final solution with proper form.

FOURIER NEURAL OPERATOR (FNO)

https://arxiv.org/abs/2010.08895, ICLR 2021.
FNO: FFT + NO, Zongyi L. et. al., Caltech+NVIDIA

- Contribution
  - Learned an entire family of PDEs instead of solving only one instance such as FEM, FDM, PINN.
  - Resolution-invariant, can do Zero-Shot super-resolution.
  - Outperformed all existing DL methods with 30%+ lower error rate.
  - Sped up more than 440x comparing to Spectral Method.



- Method

Definition 1 (Iterative updates) *Define the update to the representation* $v_t \mapsto v_{t+1}$ *by*

$$v_{t+1}(x) := \sigma\Big(W v_t(x) + \big(\mathcal{K}(a;\phi)v_t\big)(x)\Big), \qquad \forall x \in D \tag{2}$$

Definition 2 (Kernel integral operator $\mathcal{K}$**)** *Define the kernel integral operator mapping in (2) by*

$$\big(\mathcal{K}(a;\phi)v_t\big)(x) := \int_D \kappa\big(x, y, a(x), a(y); \phi\big) v_t(y) \mathrm{d}y, \qquad \forall x \in D \tag{3}$$

Definition 3 (Fourier integral operator $\mathcal{K}$**)** *Define the Fourier integral operator*

$$\big(\mathcal{K}(\phi)v_t\big)(x) = \mathcal{F}^{-1}\Big(R_\phi \cdot (\mathcal{F}v_t)\Big)(x) \qquad \forall x \in D \tag{4}$$

Lab 0: Solving a Darcy flow problem using Fourier Neural Operators

DARCY FLOW EXAMPLE

DATA: https://github.com/zongyi-li/fourier_neural_operator

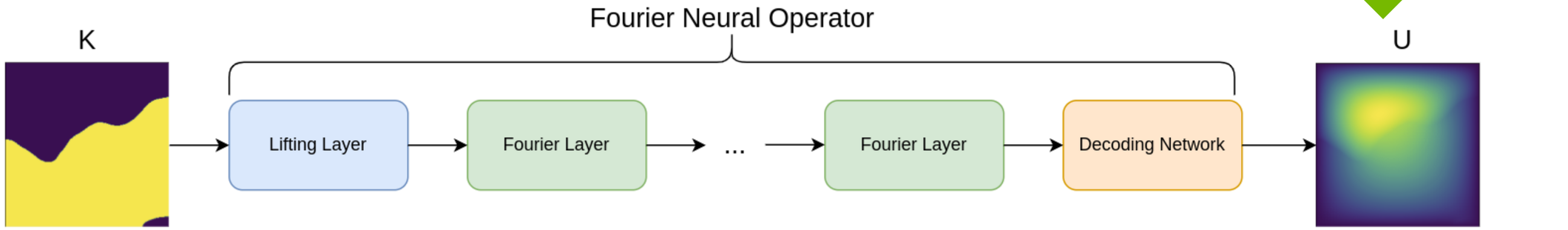
$$-\nabla \cdot (k(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x}), \quad \mathbf{x} \in D,$$

permeability field | flow pressure | forcing function

+ Conditions

MATLAB Solver

Generate **U, K** data pairs



This problem develops a surrogate model that learns the mapping between a permeability field and the pressure field, K→U, for a distribution of permeability fields K~p(K). This is a key distinction of this problem from other examples, you are *not* learning just a single solution but rather a distribution.

DARCY FLOW EXAMPLE

```python
# make list of nodes to unroll graph on
model = instantiate_arch(
    input_keys=input_keys,
    output_keys=output_keys,
    cfg=cfg.arch.fno,
)
nodes = model.make_nodes(name="FNO", jit=cfg.jit)
```

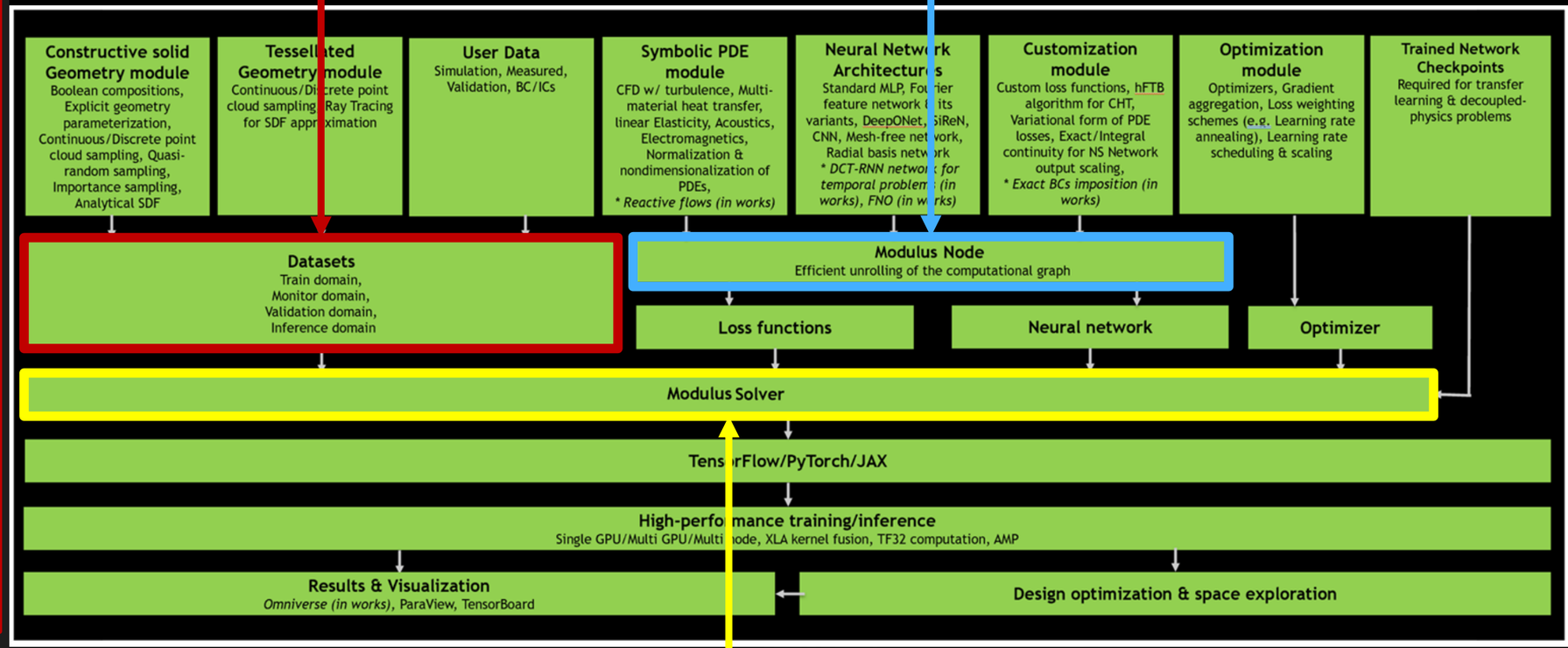
```python
# make domain
domain = Domain()

# add constraints to domain
supervised = SupervisedGridConstraint(
    nodes=nodes,
    invar=invar_train,
    outvar=outvar_train,
    batch_size=cfg.batch_size.grid,
    cell_volumes=None,
    lambda_weighting=None,
)
domain.add_constraint(supervised, "supervised")

# add validator
val = GridValidator(
    invar_test,
    outvar_test,
    nodes,
    batch_size=cfg.batch_size.validation,
    plotter=GridValidatorPlotter(n_examples=5),
)
domain.add_validator(val, "test")
```

```python
# make solver
slv = Solver(cfg, domain)

# start solver
slv.solve()
```



AGENDA – DAY 2

09:30-10:20: Introduce Modulus

10:30-11:20: Physics Inspired Lab 0: Solve a Darcy flow problem using Fourier Neural Operators

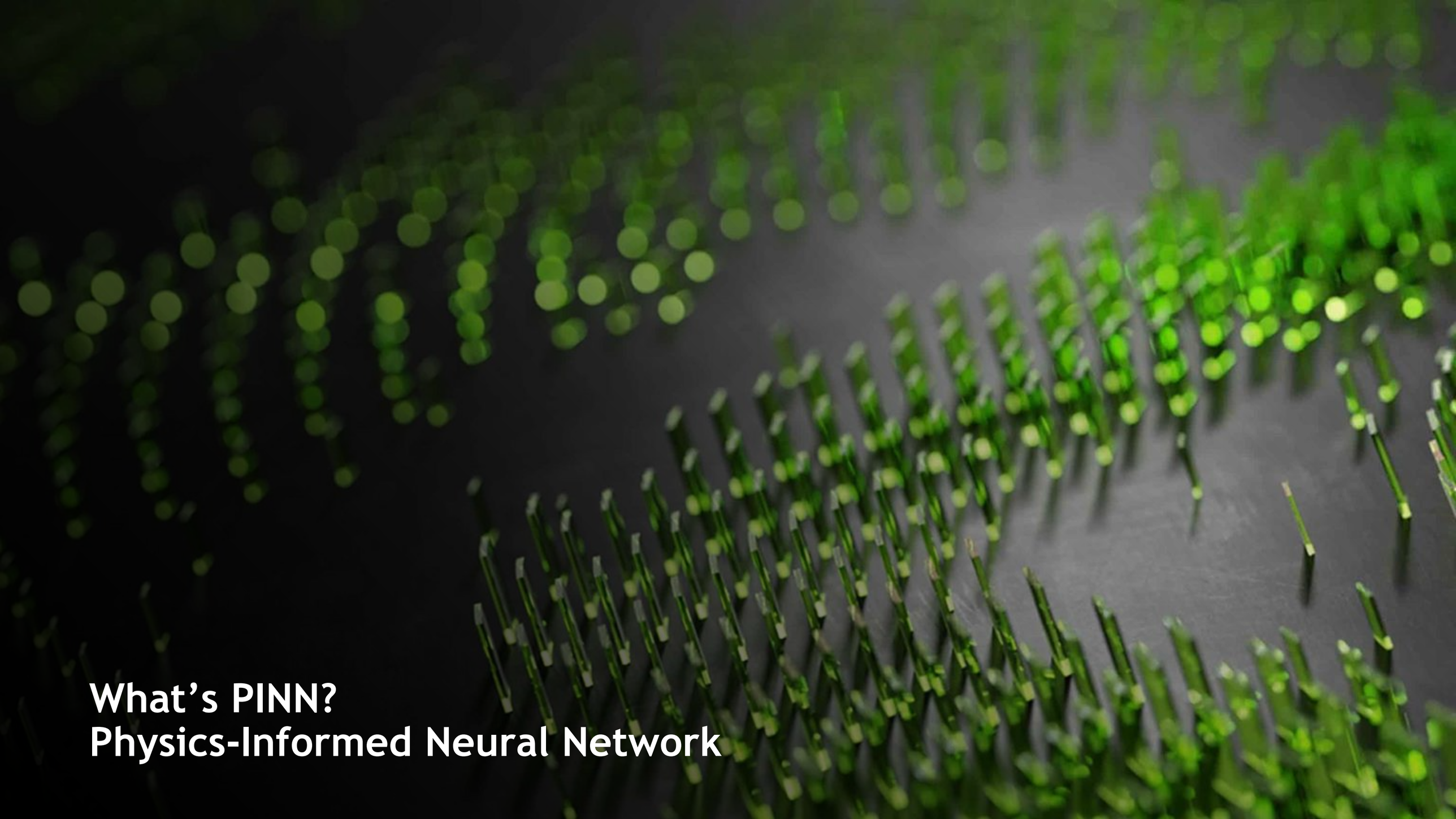
11:30-12:00: Physics Driven Lab 1: Solve Partial Differential Equations using PINN approach

12:00-13:00: Launch Break

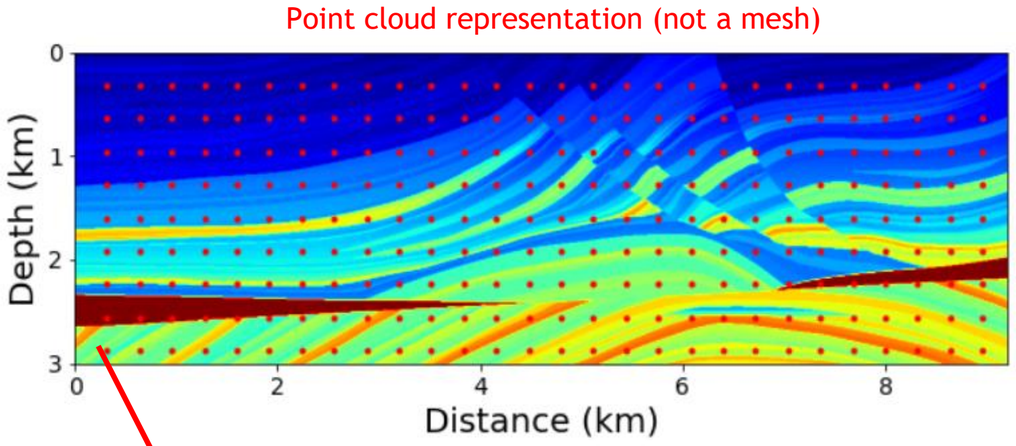
13:00-13:50: Physics Driven Lab 1: Solve Partial Differential Equations using PINN approach

14:00-14:50: Physics Driven Lab 2: Solve transient problems and inverse problems

15:00-16:00: Physics Driven Challenge: Solve a fluid mechanics problem


What’s PINN?
Physics-Informed Neural Network

AN EXAMPLE: SEISMIC PROBLEMS



PHYSICS

Neural Network

Input

Output

PDE

$\frac{\partial^2 u}{\partial t^2} - c^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z^2}\right)$

IC

$u(x,z,t) - u_D$

BC

$\frac{\partial u(x,z,t)}{\partial n} - u_N$

$L_{PDE} + L_{IC} + L_{BC}$

Loss

NEURAL NETWORK SOLVER THEORY

- Goal: Train a neural network to satisfy the boundary conditions and differential equations by constructing an appropriate loss function
  - Consider an example problem:

$$\mathbf{P}: \begin{cases} \dfrac{\delta^2 u}{\delta x^2}(x) = f(x) \\ u(0) = u(1) = 0 \end{cases}$$

- We construct a neural network $\boldsymbol{u_{net}(x)}$ which has a single value input $x \in \mathbb{R}$ and single value output $u_{net}(x) \in \mathbb{R}$.
- We assume the neural network is **infinitely differentiable** $u_{net} \in C^{\infty}$ - Use activation functions that are infinitely differentiable

NEURAL NETWORK SOLVER THEORY

$$\mathbf{P}: \begin{cases} \frac{\delta^2 u}{\delta x^2}(x) = f(x) \\ u(0) = u(1) = 0 \end{cases}$$

- Construct the loss function. We can compute the second order derivatives $\left(\frac{\delta^2 u_{net}}{\delta x^2}(x)\right)$ using **Automatic differentiation**

$$L_{BC} = u_{net}(0)^2 + u_{net}(1)^2$$

- Where x_i are a batch of points in the interior $\boldsymbol{x_i} \in (\mathbf{0}, \mathbf{1})$. Total loss becomes $L = L_{BC} + L_{Residual}$
- Minimize the loss using optimizers like Adam

$$L_{Residual} = \int_0^1 \left(\frac{\delta^2 u_{net}}{\delta x^2}(x) - f(x)\right)^2 dx \approx \left(\int_0^1 dx\right) \frac{1}{N} \sum_{i=0}^{N} \left(\frac{\delta^2 u_{net}}{\delta x^2}(x_i) - f(x_i)\right)^2$$

NEURAL NETWORK SOLVER THEORY

A Toy Example for the partial derivatives

- why no need mesh for gradient approximation? explanied by an toy example:

$$\mathcal{L}u = f$$
$$u(x) \approx \mathcal{N}(x; w_1, w_2) = w_1 x + w_2$$

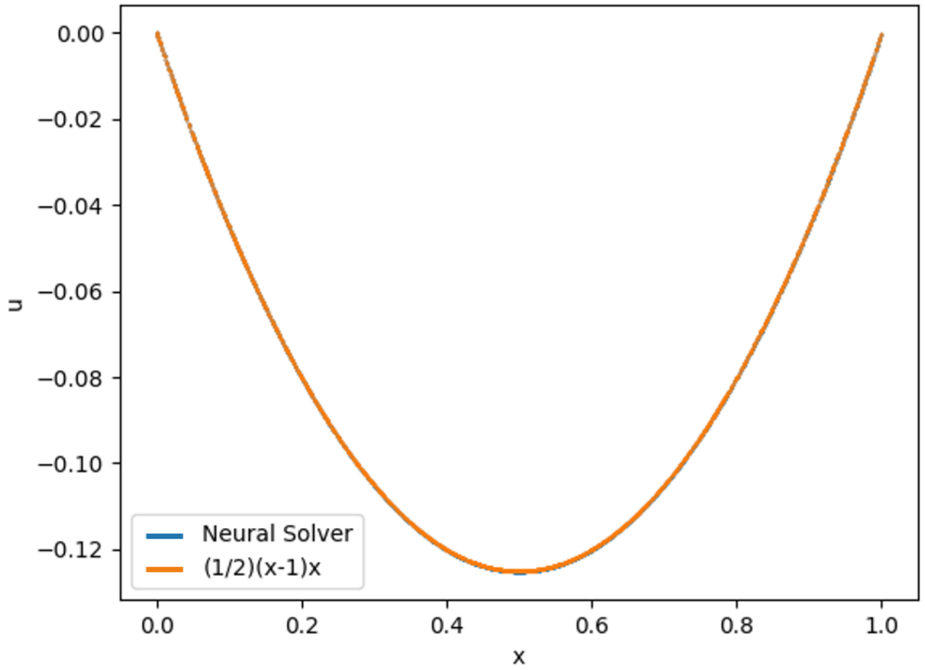
- then we could get derivate $\frac{\partial u(x)}{\partial x}$ by math definition directly without meshing.

$$\frac{\partial u}{\partial x} \approx \frac{\partial \mathcal{N}}{\partial x} = w_1$$

NEURAL NETWORK SOLVER THEORY

$$\mathbf{P}: \begin{cases} \frac{\delta^2 u}{\delta x^2}(x) = f(x) \\ u(0) = u(1) = 0 \end{cases}$$

- For $f(x) = 1$, the true solution is $\frac{1}{2}(x-1)x$. After sufficient training we have,



Comparison of the solution predicted by Neural Network with the analytical solution

SOLVING PARAMETERIZED PROBLEMS

- Consider the parameterized version of the same problem as before. Suppose we want to determine how the solution changes as we move the position on the boundary condition $u(l) = 0$
- Parameterize the position by variable $l \in [1, 2]$ and the problem now becomes:

$$\mathbf{P}: \begin{cases} \dfrac{\delta^2 u}{\delta x^2}(x) = f(x) \\ u(0) = u(l) = 0 \end{cases}$$

- This time, we construct a neural network $u_{net}(x, l)$ which has x and l as input and single value output $u_{net}(x, l) \in \mathbb{R}$.
- The losses become

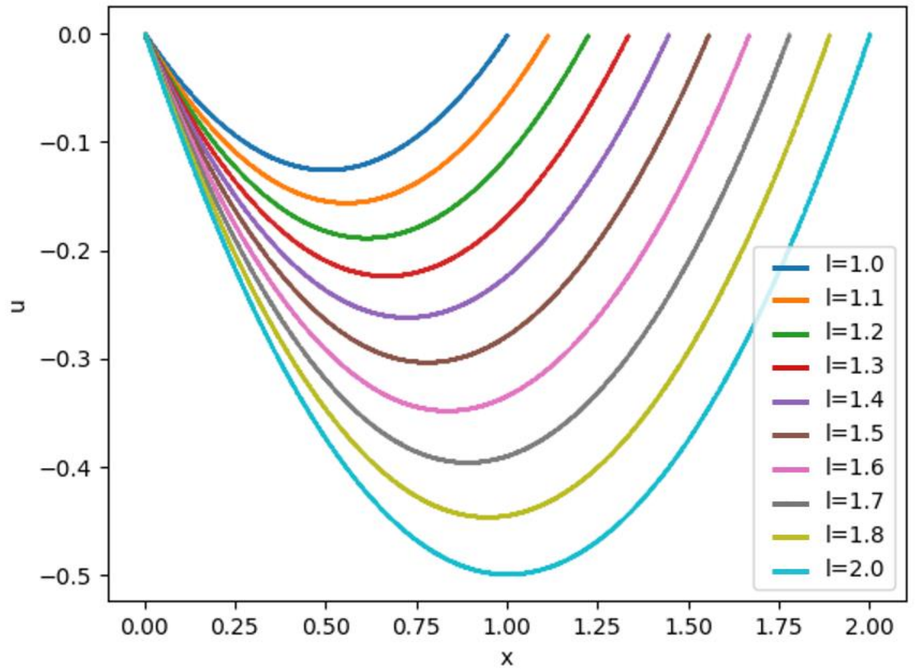
$$L_{Residual} = \int_1^2 \int_0^1 \left(\frac{\delta^2 u_{net}}{\delta x^2}(x) - f(x) \right)^2 dx dl \approx \left(\int_1^2 \int_0^1 dx\, dl \right) \frac{1}{N} \sum_{i=0}^{N} \left(\frac{\delta^2 u_{net}}{\delta x^2}(x_i, l_i) - f(x_i) \right)^2$$

$$L_{BC} = \int_1^2 \big(u_{net}(0, l)\big)^2 + \big(u_{net}(l, l)\big)^2 dl \approx \left(\int_1^2 dl \right) \frac{1}{N} \sum_{i=0}^{N} (u_{net}(0, l_i))^2 + (u_{net}(l_i, l_i))^2$$

SOLVING PARAMETERIZED PROBLEMS

$$\mathbf{P}: \begin{cases} \frac{\delta^2 u}{\delta x^2}(x) = f(x) \\ u(0) = u(l) = 0 \end{cases}$$

- For $f(x) = 1$, for different values of l we have different solutions



Solution to the parametric problem

SOLVING INVERSE PROBLEMS

$$\mathbf{P}: \begin{cases} \frac{\delta^2 u}{\delta x^2}(x) = f(x) \\ u(0) = u(1) = 0 \end{cases}$$

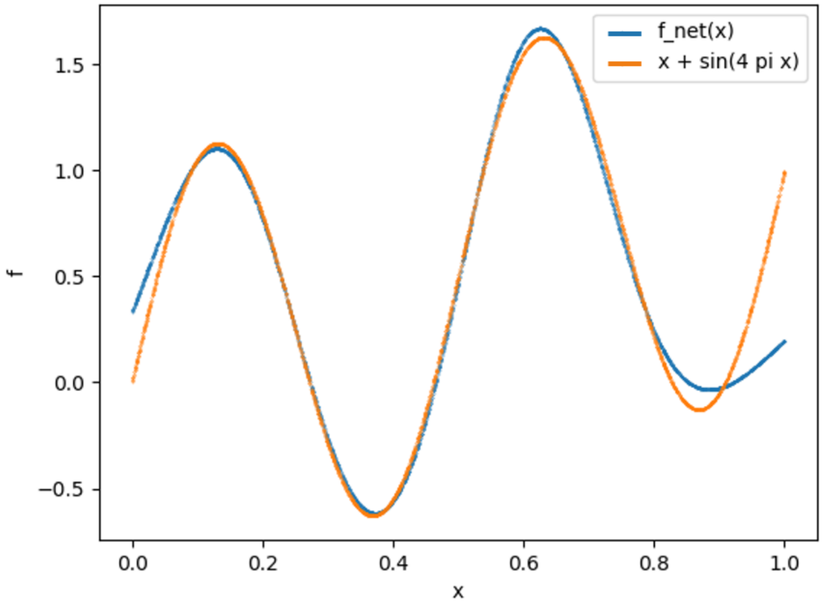
- For inverse problems, we start with a set of observations and then calculate the causal factors that produced them
- For example, suppose we are given the solution $\boldsymbol{u_{true}(x)}$ at 100 random points between 0 and 1 and we want to determine the $f(x)$ that is causing it
- Train two networks $\boldsymbol{u_{net}(x)}$ **and** $\boldsymbol{f_{net}(x)}$ to approximate $u(x)$ and $f(x)$

$$L_{Residual} \approx \left(\int_0^1 dx\right)\frac{1}{N}\sum_{i=0}^{N}\left(\frac{\delta^2 u_{net}}{\delta x^2}(x_i) - f_{net}(x_i)\right)^2$$

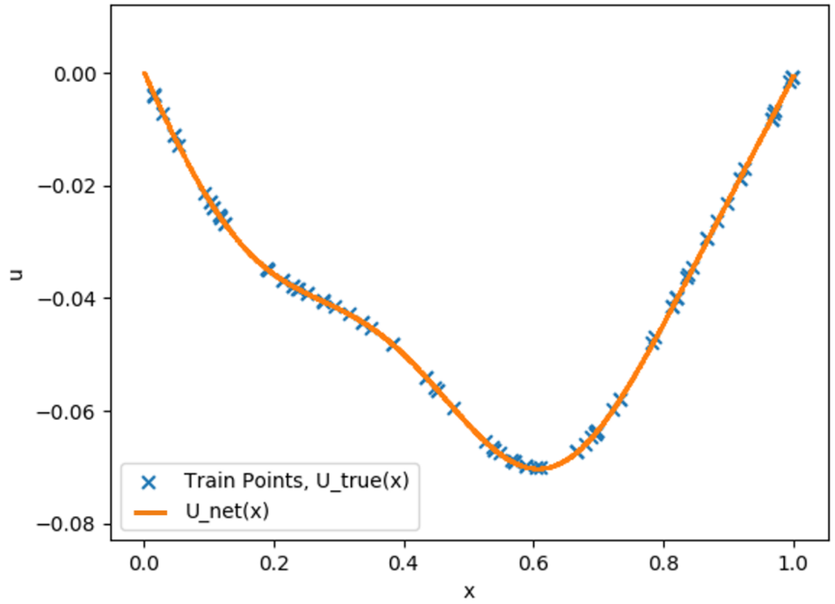
$$L_{Data} = \frac{1}{100}\sum_{i=0}^{100}\left(u_{net}(x_i) - u_{true}(x_i)\right)^2$$

SOLVING INVERSE PROBLEMS

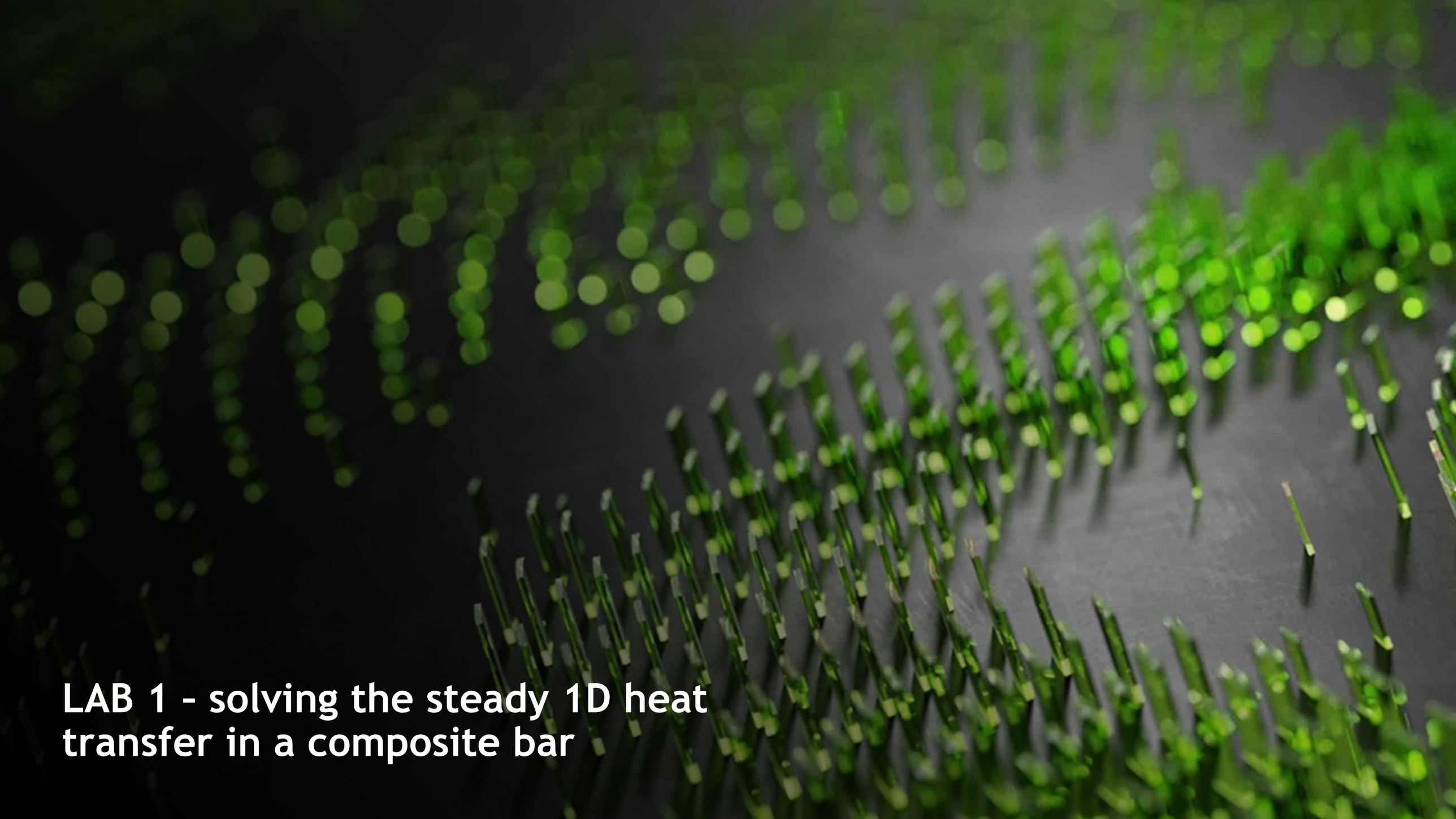
- For $u_{true}(x) = \frac{1}{48}\left(8x\left(-1+x^2\right) - \frac{3\sin(4\pi x)}{\pi^2}\right)$ the solution for $f(x)$ is $x + \sin(4\pi x)$



Comparison of the true solution for $f(x)$ and the $f_{net}(x)$ inverted out

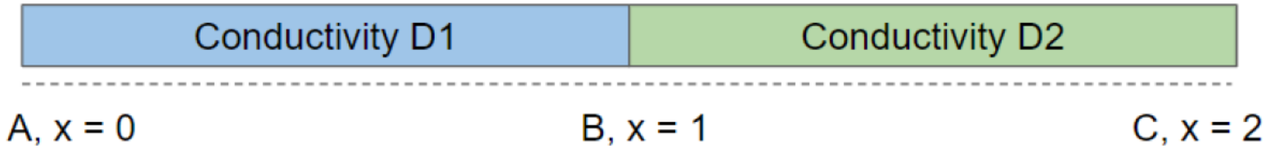


Comparison of $u_{net}(x)$ and train points from u_{true}



LAB 1 – solving the steady 1D heat transfer in a composite bar

Solution to PDEs- 1d diffusion



- Composite bar with material of conductivity $D_1 = 10$ for $x \in (0,1)$ and $D_2 = 0.1$ for $x \in (1,2)$. Point A and C are maintained at temperatures of 0 and 100 respectively

- Equations: Diffusion equation in 1D

$$\frac{d}{dx}\left(D_1 \frac{dU_1}{dx}\right) = 0 \qquad \text{When } 0 < x < 1$$

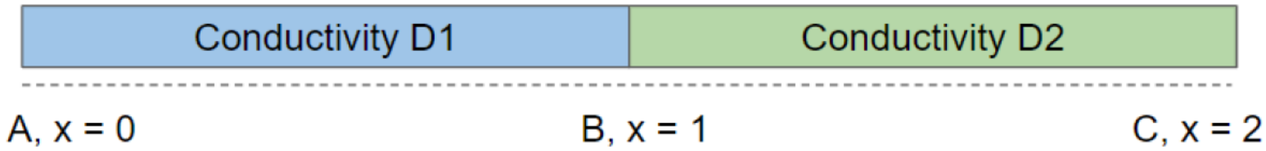
$$\frac{d}{dx}\left(D_2 \frac{dU_2}{dx}\right) = 0 \qquad \text{When } 1 < x < 2$$

- Flux and field continuity at interface $(x = 1)$

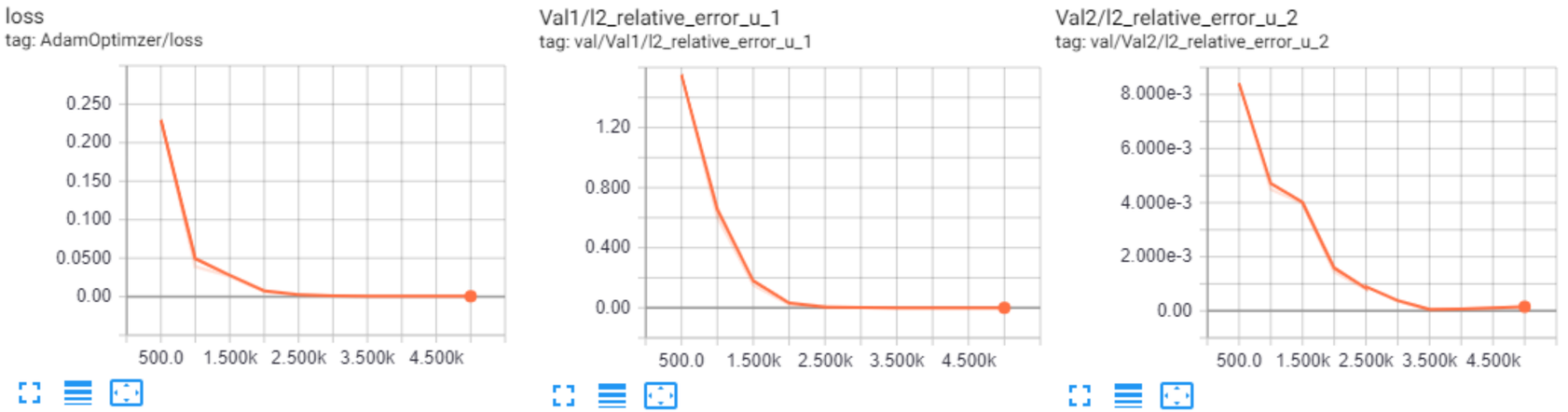
$$\left(D_1 \frac{dU_1}{dx}\right) = \left(D_2 \frac{dU_2}{dx}\right)$$

$$U_1 = U_2$$

Solution to PDEs- 1d diffusion

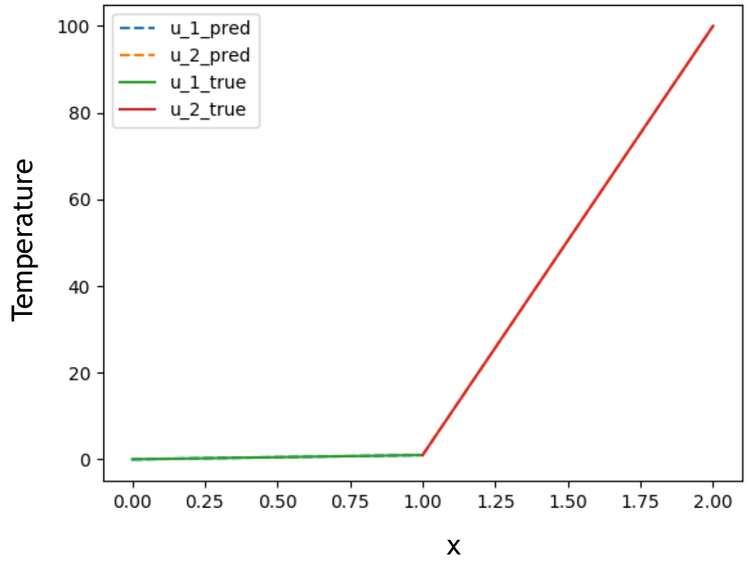


- Define the problem and train the neural network to obtain the temperature distribution in the bar
- Compare the results with analytical solution

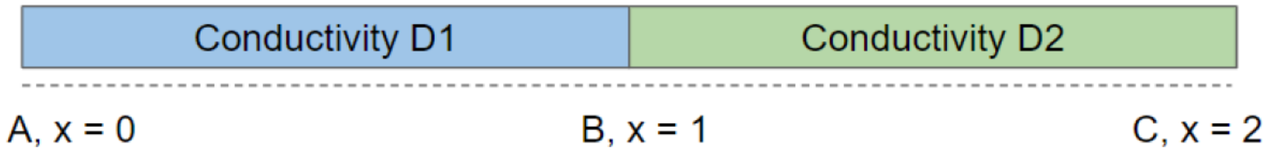


Training error

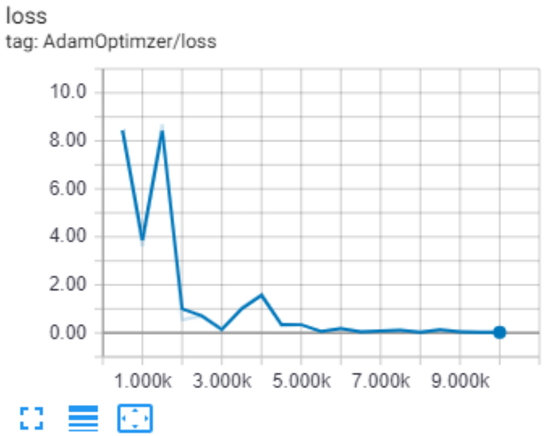
Validation error



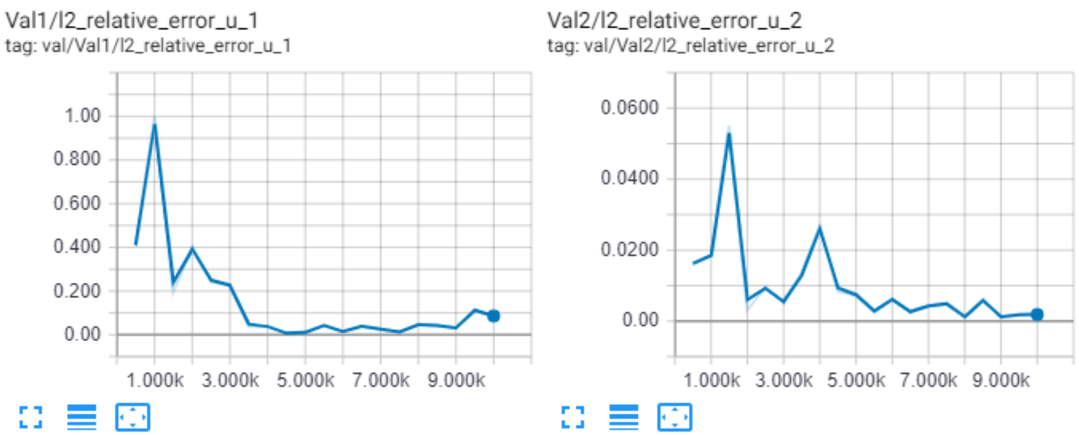
Solution to parameterized PDEs- 1d diffusion



- Composite bar with material of conductivity D_1 for $x \in (0,1)$ and $D_2 = 0.1$ for $x \in (1,2)$.
- Solve the problem for multiple values of D_1 in the range $(5, 25)$ in a single training
- Same boundary and interface conditions as before



Training error



Validation error for $D_1 = 10$

AGENDA – DAY 2

09:30-10:20: Introduce Modulus

10:30-11:20: Physics Inspired Lab 0: Solve a Darcy flow problem using Fourier Neural Operators

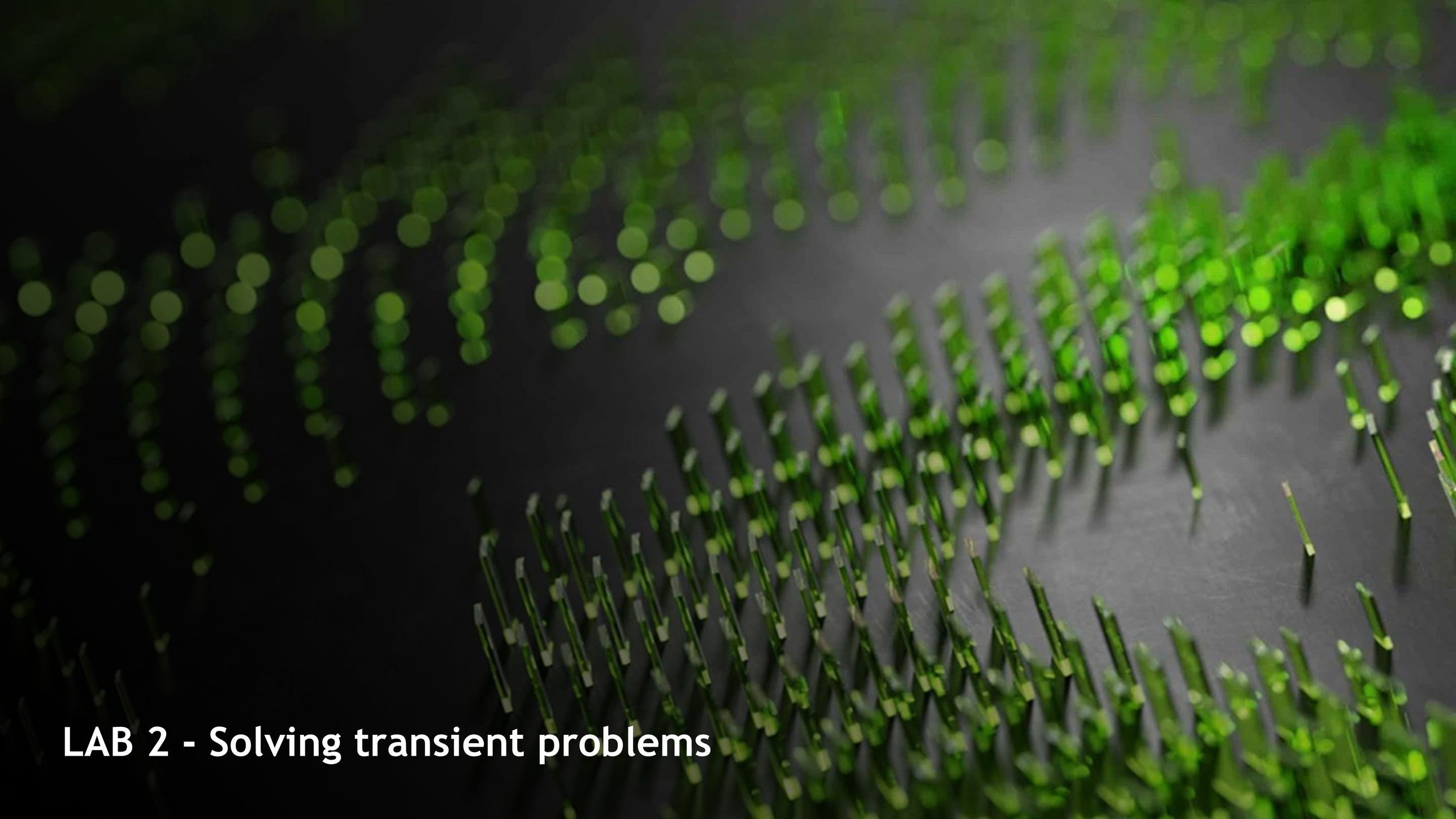
11:30-12:00: Physics Driven Lab 1: Solve Partial Differential Equations using PINN approach

12:00-13:00: Launch Break

13:00-13:50: Physics Driven Lab 1: Solve Partial Differential Equations using PINN approach

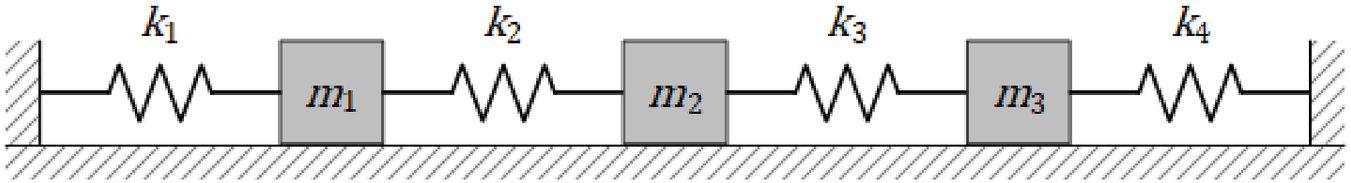
14:00-14:50: Physics Driven Lab 2: Solve transient problems and inverse problems

15:00-16:00: Physics Driven Challenge: Solve a fluid mechanics problem



LAB 2 - Solving transient problems

SOLUTION TO ODES- COUPLED SPRING MASS SYSTEM



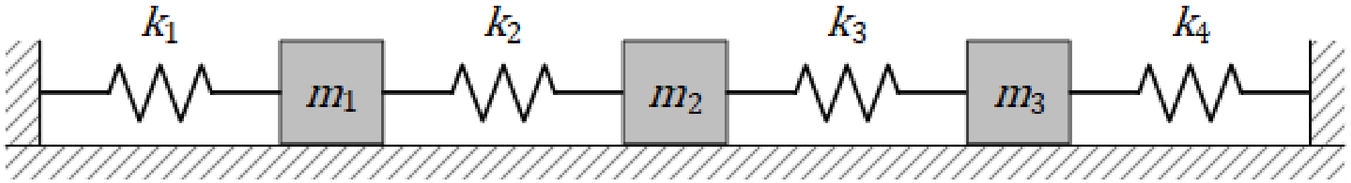
- Three masses connected by four springs
- System's equations (ordinary differential equations):

$$m_1 x_1''(t) = -k_1 x_1(t) + k_2\big(x_2(t) - x_1(t)\big)$$
$$m_2 x_2''(t) = -k_2\big(x_2(t) - x_1(t)\big) + k_3\big(x_3(t) - x_2(t)\big)$$
$$m_3 x_3''(t) = -k_3\big(x_3(t) - x_2(t)\big) - k_4 x_3(t)$$

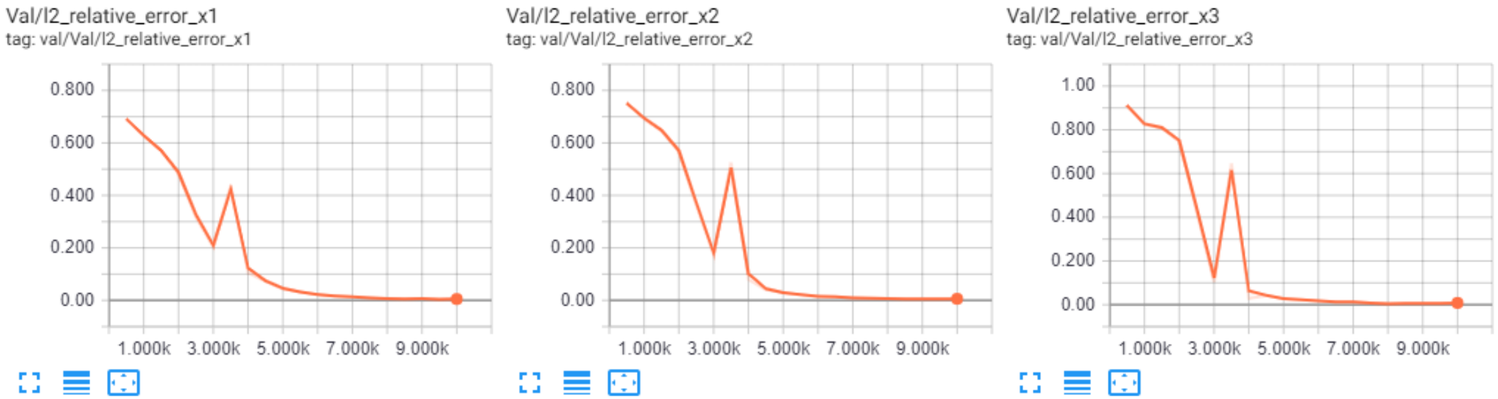
- For given values masses, spring constants and boundary conditions

$$[m_1, m_2, m_3] = [1, 1, 1]$$
$$[k_1, k_2, k_3, k_4] = [2, 1, 1, 2]$$
$$[x_1(0), x_2(0), x_3(0)] = [1, 0, 0]$$
$$[x_1'(0), x_2'(0), x_3'(0)] = [0, 0, 0]$$

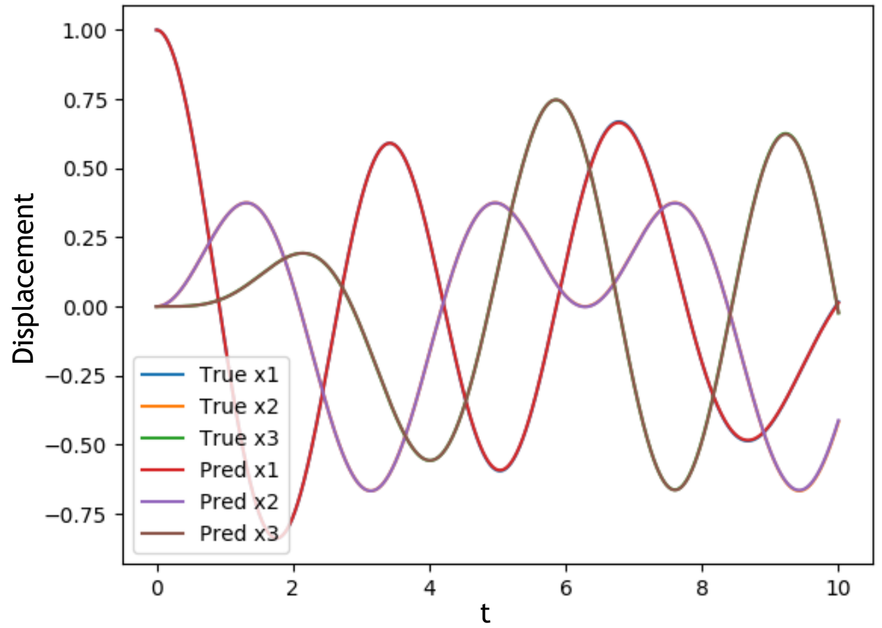
SOLUTION TO ODES- COUPLED SPRING MASS SYSTEM



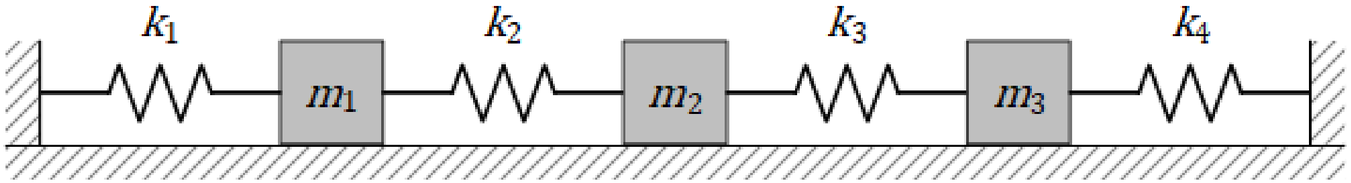
- Define the transient problem for time, $t = (0, 10)$ and train the neural network to obtain the displacement of each mass
- Compare the results with analytical solution



Validation error for each mass



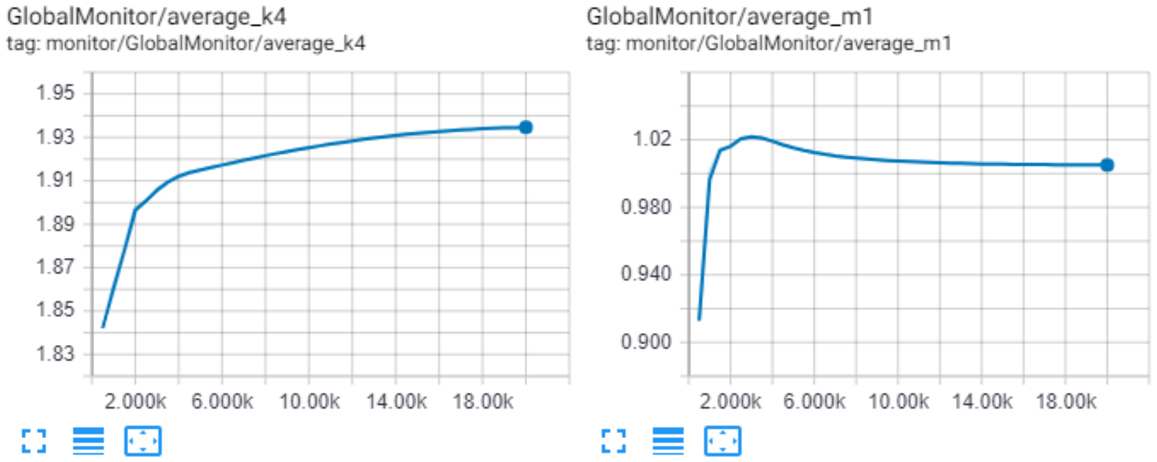
INVERSE PROBLEMS- COUPLED SPRING MASS SYSTEM



- For the same system, assume we know the analytical solution which is given by:

$$x_1(t) = \frac{1}{6}\cos(t) + \frac{1}{2}\cos\left(\sqrt{3}t\right) + \frac{1}{3}\cos(2t), \qquad x_2(t) = \frac{2}{6}\cos(t) - \frac{1}{3}\cos(2t), \qquad x_3(t) = \frac{1}{6}\cos(t) - \frac{1}{2}\cos\left(\sqrt{3}t\right) + \frac{1}{3}\cos(2t)$$

- With the above data and the values for m_2, m_3, k_1, k_2, k_3 same as before, use the neural network to find the values of m_1 and k_4



AGENDA – DAY 2

09:30-10:20: Introduce Modulus

10:30-11:20: Physics Inspired Lab 0: Solve a Darcy flow problem using Fourier Neural Operators

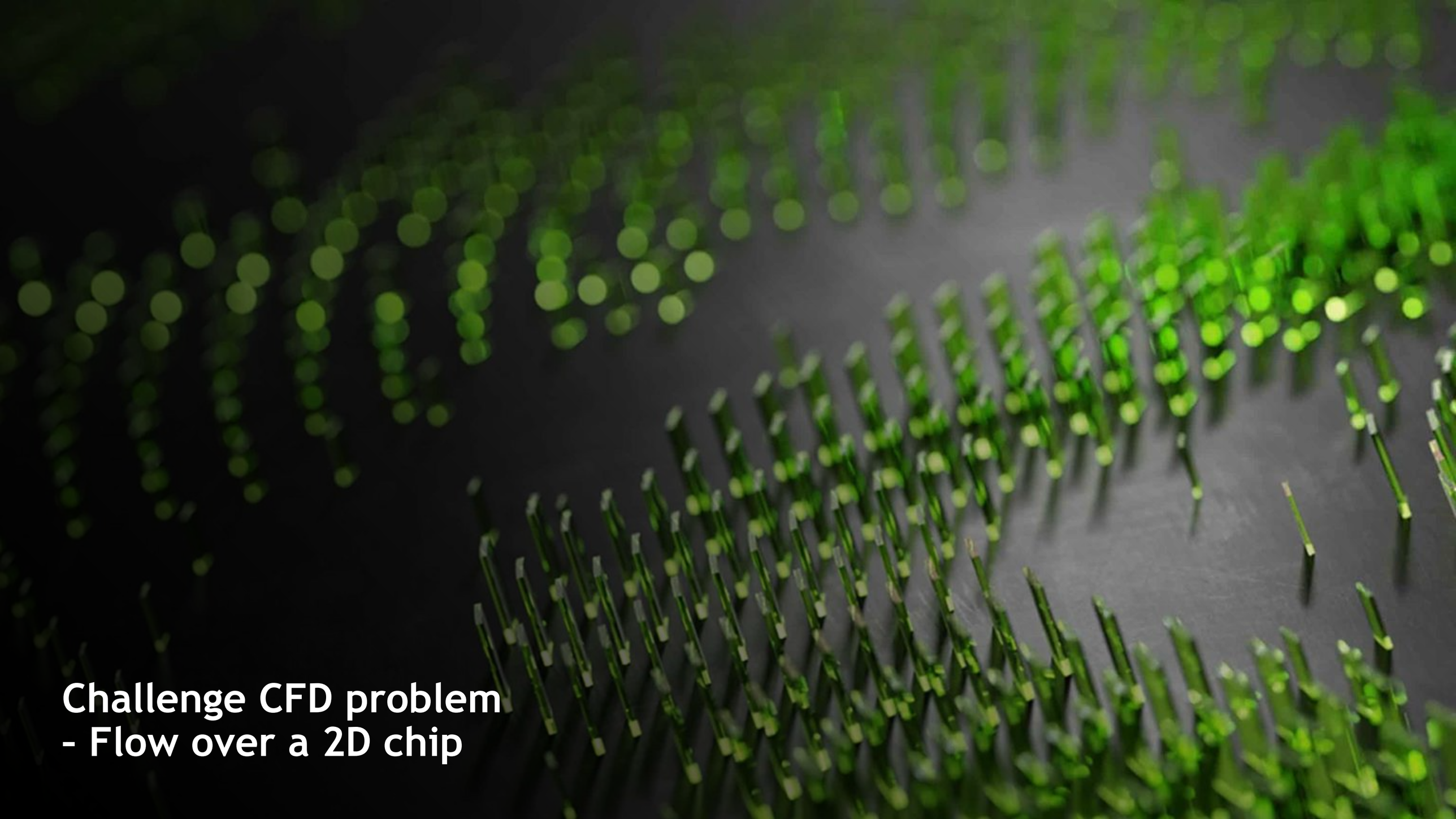
11:30-12:00: Physics Driven Lab 1: Solve Partial Differential Equations using PINN approach

12:00-13:00: Launch Break

13:00-13:50: Physics Driven Lab 1: Solve Partial Differential Equations using PINN approach

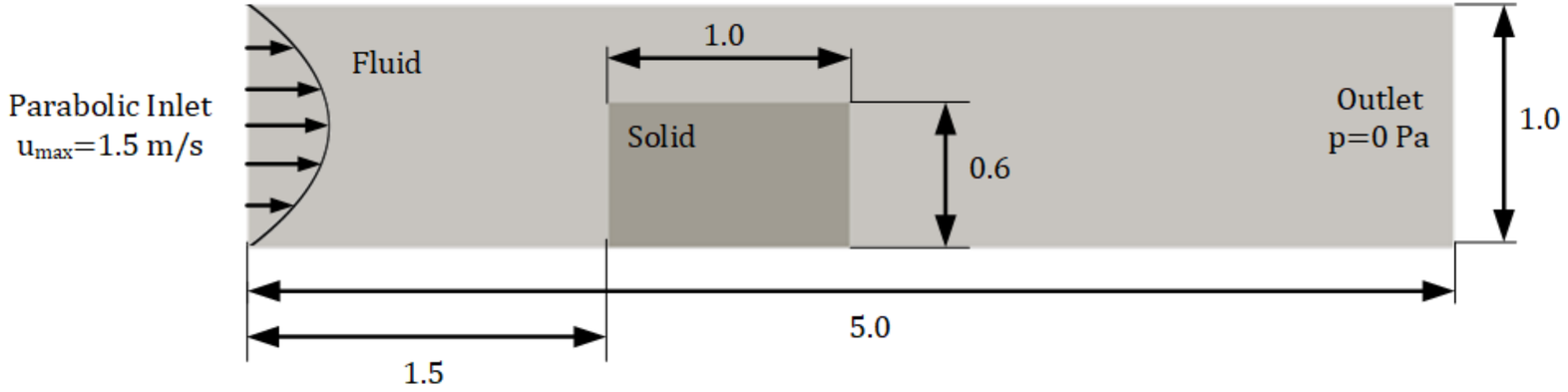
14:00-14:50: Physics Driven Lab 2: Solve transient problems and inverse problems

15:00-16:00: Physics Driven Challenge: Solve a fluid mechanics problem



Challenge CFD problem
- Flow over a 2D chip

CHALLENGE: FLOW OVER 2D CHIP

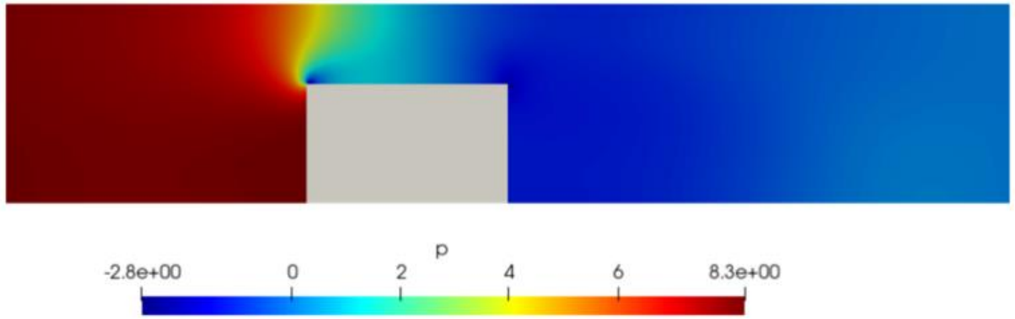


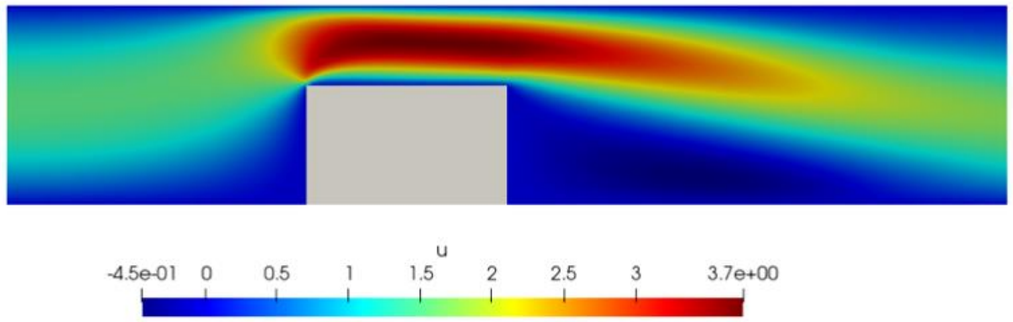
- Solve the flow over 2D chip for the given boundary conditions. The challenge problem has 3 parts:

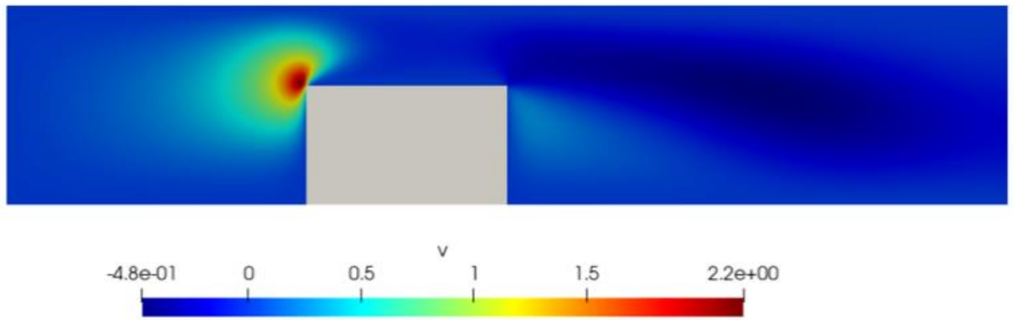
1. Solve the fluid flow for the given boundary conditions and geometry
2. Solve the fluid flow for the parameterized Chip geometry
3. Solve the inverse problem where, given a flow field, use it to invert out the viscosity of the flow

CHALLENGE: FLOW OVER 2D CHIP

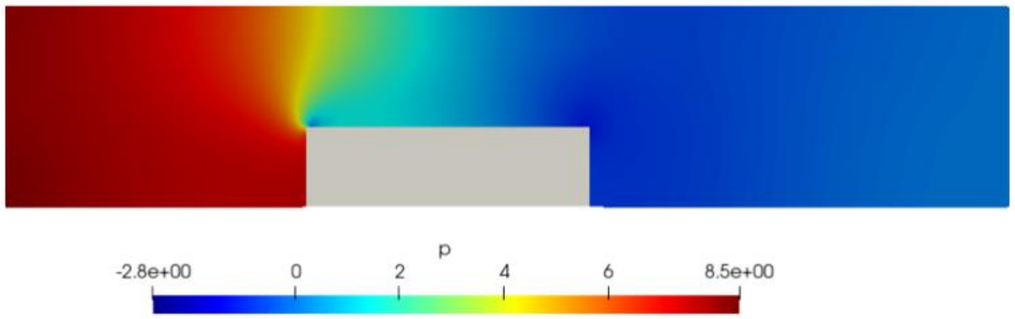
h: 0.6, w: 1.0

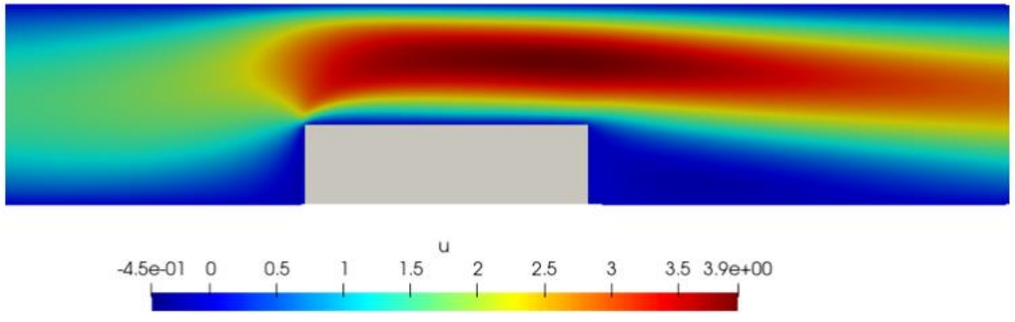


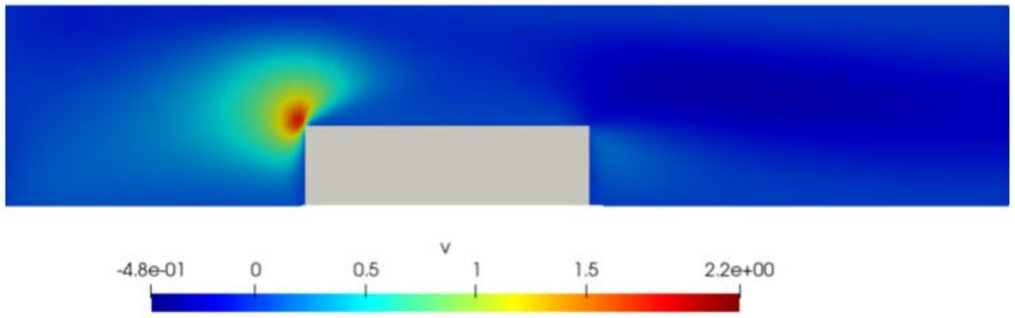




h: 0.4, w: 1.4







TODO

chip_2d_template.py

```python
@modulus.main(config_path="conf", config_name="config")
def run(cfg: ModulusConfig) -> None:
    print(to_yaml(cfg))

    #TODO: Replace all the placeholders with appropriate values
    # make list of nodes to unroll graph on
    ns = NavierStokes(nu=0.02, rho=1.0, dim=2, time=False)
    normal_dot_vel = NormalDotVec(["u", "v"])
    flow_net = instantiate_arch(
        input_keys=[placeholder],
        output_keys=[placeholder],
        cfg=cfg.arch.fourier_net,
    )
    nodes = (
        ns.make_nodes()
        + normal_dot_vel.make_nodes()
        + [flow_net.make_node(name="flow_network", jit=cfg.jit)]
    )
```

```python
# define sympy variables to parametrize domain curves
x, y = Symbol("x"), Symbol("y")

#TODO: Replace x1, y1, x2, y2, and X's with appropriate values

# define geometry
channel = Channel2D(
    (channel_length[0], channel_width[0]), (channel_length[1], channel_width[1])
)
inlet = Line(
    (x1, y1),
    (x1, y2),
    normal=1,
)
outlet = Line(
    (x2, y1),
    (x2, y2),
    normal=1,
)
rec = Rectangle(
    (x1, y1),
    (x2, y2),
)
flow_rec = Rectangle(
    (chip_pos - 0.25, channel_width[0]),
    (chip_pos + chip_width + 0.25, channel_width[1]),
)
geo = channel - rec
geo_hr = geo & flow_rec
geo_lr = geo - flow_rec
```

```python
#TODO: Replace all the placeholders with appropriate values

# make domain
domain = Domain()

# inlet
inlet_parabola = parabola(y, channel_width[0], channel_width[1], inlet_vel)
inlet = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=inlet,
    outvar={"u": inlet_parabola, "v": 0},
    batch_size=cfg.batch_size.inlet,
)
domain.add_constraint(inlet, "inlet")

# outlet
outlet = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=outlet,
    outvar={placeholder},
    batch_size=cfg.batch_size.outlet,
    criteria=Eq(x, channel_length[1]),
)
domain.add_constraint(outlet, "outlet")

# no slip
no_slip = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=geo,
    outvar={placeholder},
    batch_size=cfg.batch_size.no_slip,
)
domain.add_constraint(no_slip, "no_slip")
```

```python
# interior lr
interior_lr = PointwiseInteriorConstraint(
    nodes=nodes,
    geometry=geo_lr,
    outvar={placeholder},
    batch_size=cfg.batch_size.interior_lr,
    bounds={x: channel_length, y: channel_width},
    lambda_weighting={placeholder},
)
domain.add_constraint(interior_lr, "interior_lr")

# interior hr
interior_hr = PointwiseInteriorConstraint(
    nodes=nodes,
    geometry=geo_hr,
    outvar={placeholder},
    batch_size=cfg.batch_size.interior_hr,
    bounds={x: channel_length, y: channel_width},
    lambda_weighting={placeholder},
)
domain.add_constraint(interior_hr, "interior_hr")

# integral continuity
integral_continuity = IntegralBoundaryConstraint(
    nodes=nodes,
    geometry=integral_line,
    outvar={"normal_dot_vel": 1},
    batch_size=cfg.batch_size.num_integral_continuity,
    integral_batch_size=cfg.batch_size.integral_continuity,
    lambda_weighting={"normal_dot_vel": 1},
    criteria=geo.sdf > 0,
    param_ranges=x_pos_range,
)
domain.add_constraint(integral_continuity, "integral_continuity")
```

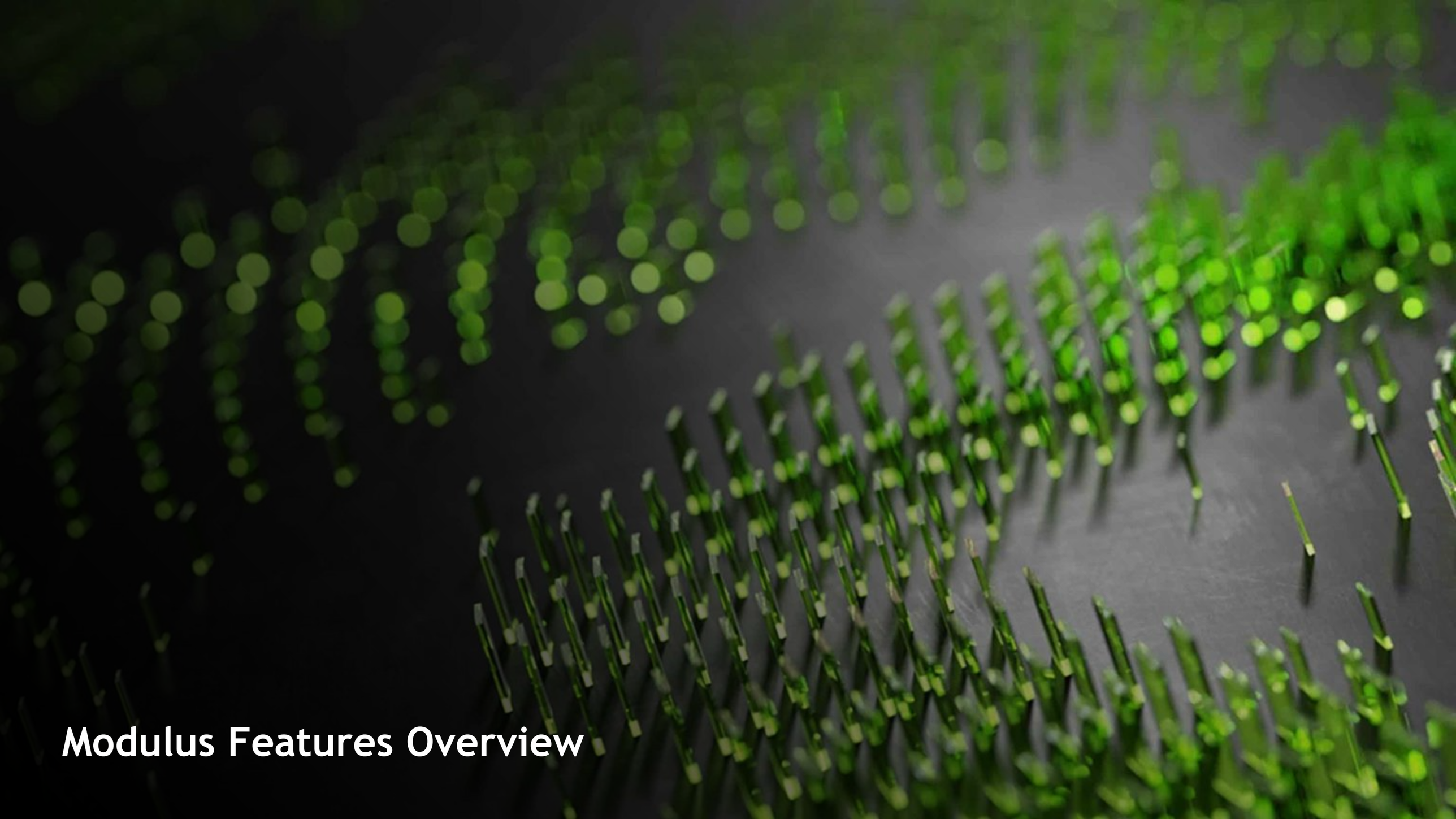
```python
#TODO: Set the appropriate normalization for the validation data
# The validation data has domain extents of (0,0) to (5,1). Normalize this based on your definition of the domain

# add validation data
mapping = {"Points:0": "x", "Points:1": "y", "U:0": "u", "U:1": "v", "p": "p"}
openfoam_var = csv_to_dict(to_absolute_path("openfoam/2D_chip_fluid0.csv"), mapping)
openfoam_var["x"] -= placeholder  # TODO normalize pos
openfoam_var["y"] -= placeholder  # TODO normalize pos
openfoam_invar_numpy = {
    key: value for key, value in openfoam_var.items() if key in ["x", "y"]
}
openfoam_outvar_numpy = {
    key: value for key, value in openfoam_var.items() if key in ["u", "v", "p"]
}
openfoam_validator = PointwiseValidator(
    placeholder, placeholder, nodes
)
domain.add_validator(openfoam_validator)

# make solver
slv = Solver(cfg, domain)

# start solver
slv.solve()
```

NVIDIA



Modulus Features Overview

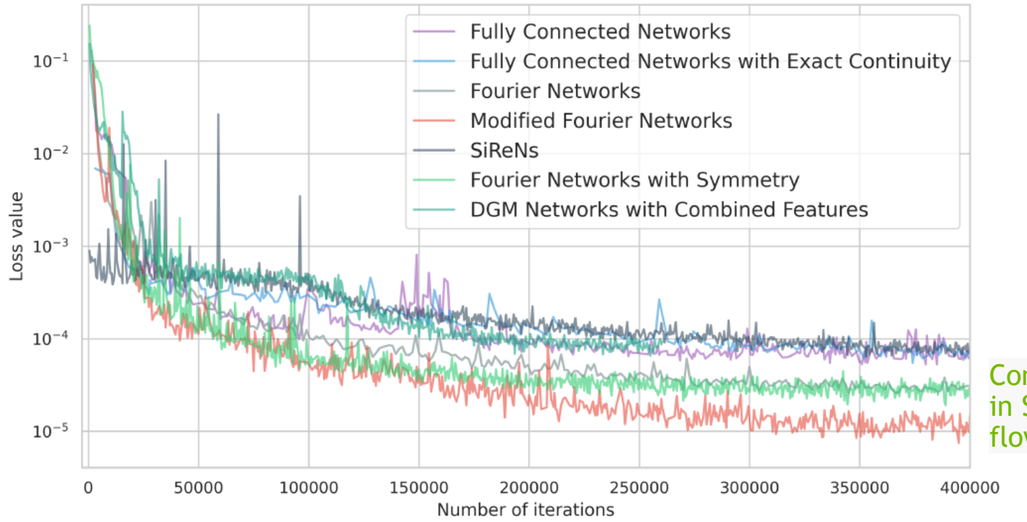
MODULUS FEATURES AND ADVANCEMENTS

Several neural network architectures:

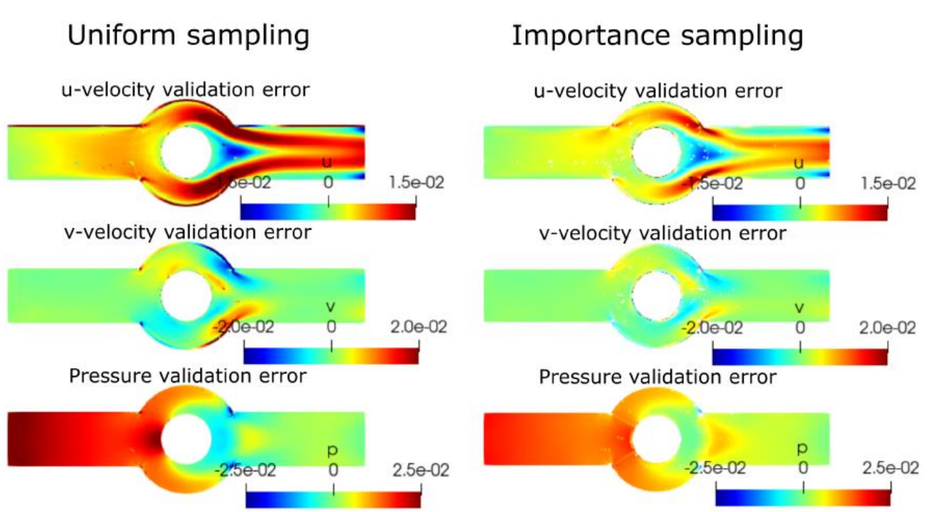
- Fully connected Network
- Fourier Feature Network
- Sinusoidal Representation Network (SiReN)
- Modified Fourier Network
- Deep Galerkin Method Network
- Modified Highway Network
- Multiplicative Filter Networks

Other Features include:

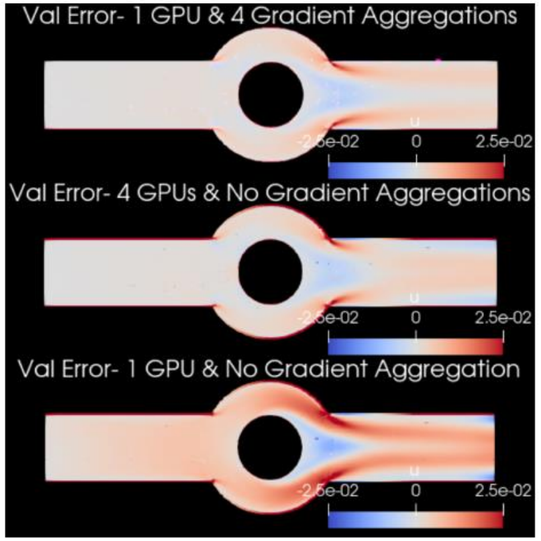
- Global and local learning rate annealing
- Global adaptive activation functions
- Halton sequences for low-discrepancy point cloud creation
- Gradient Accumulation
- Time-stepping schemes for transient problems
- Temporal loss weighting and time marching for the continuous time approach
- Importance sampling
- Homoscedastic task uncertainty quantification for loss weighting



Comparisons of various networks in SimNet applied to solve the flow over a heatsink



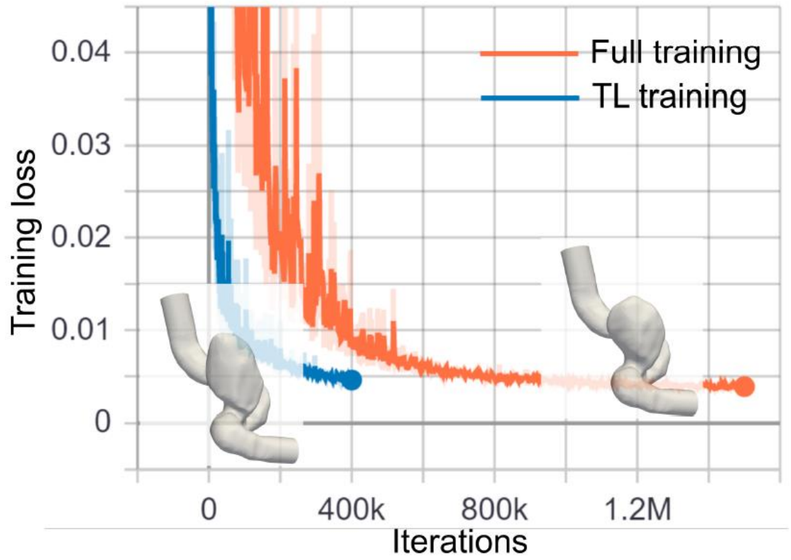
Importance sampling



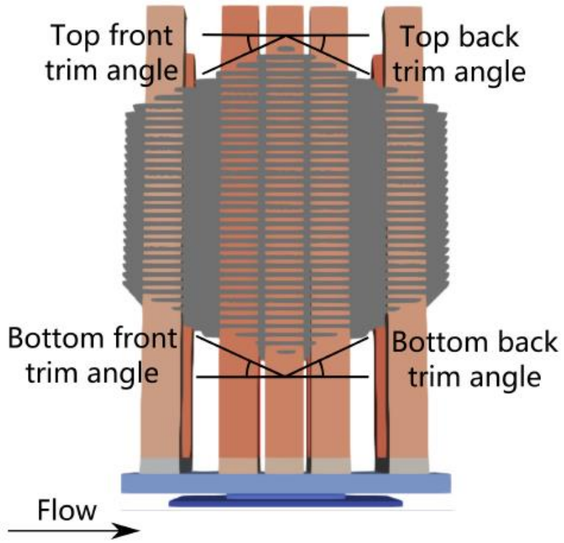
Handling larger batch sizes using Multi-GPU and/or Gradient Aggregation

MODULUS FEATURES AND ADVANCEMENTS

- APIs to automatically generate point clouds from Boolean compositions of geometry primitives or import point cloud for complex geometry (e.g., STL files)
- Parameterized system representation that solves several configurations concurrently for analytical geometry using SimNet CSG module
- Transfer learning for efficient surrogate-based parameterization of STL and constructive solid geometries
- Polynomial Chaos Expansion method for assessing how uncertainties in a model input manifest in its output



STL geometry and Transfer Learning



Geometry parameterization using SimNet's CSG module

MODULUS FEATURES AND ADVANCEMENTS

- Running jobs using TF32 math mode
  - TensorFloat-32 (TF32) is a new math mode available on NVIDIA A100 GPUs for handing matrix math and tensor operations used during the training of a neural network.
- Running jobs using Just-In-Time (JIT) compilation
  - JIT compilation is a feature where elements of the computational graph can be compiled from native PyTorch to the TorchScript backend. This allows for optimizations like avoiding python's Global Interpreter Lock (GIL) as well as compute optimizations including dead code elimination, common substring elimination and pointwise kernel fusion.
- Running jobs using multiple GPUs
  - To boost performance and to run larger problems, Modulus supports multi-GPU and multi-node scaling.

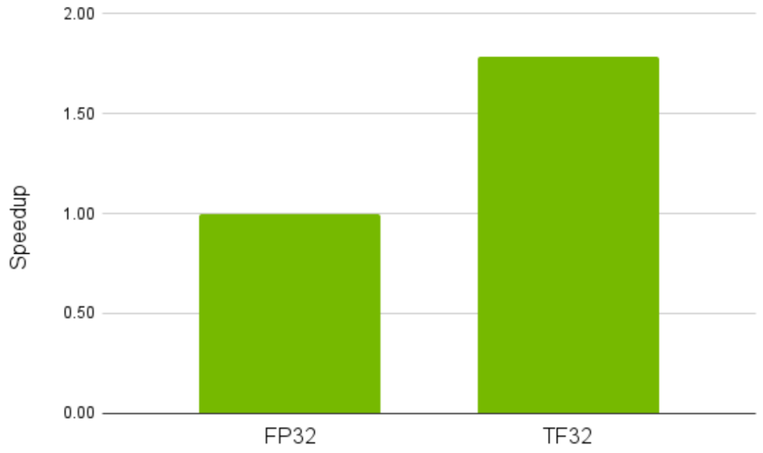


Fig. 35 *Achieved speedup using the TF32 compute mode on an A100 GPU for the FPGA example*

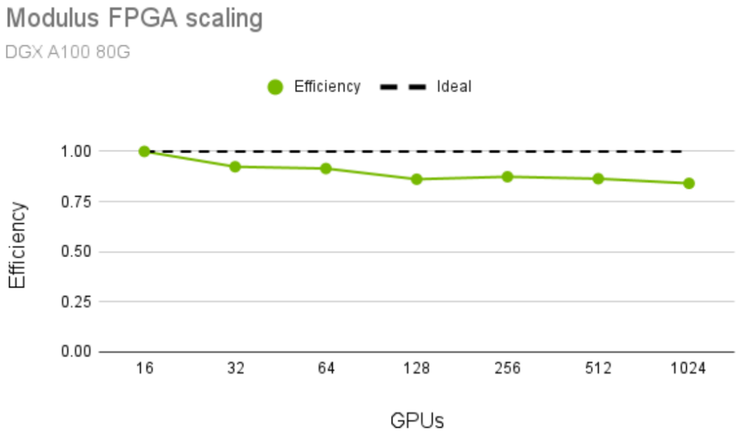
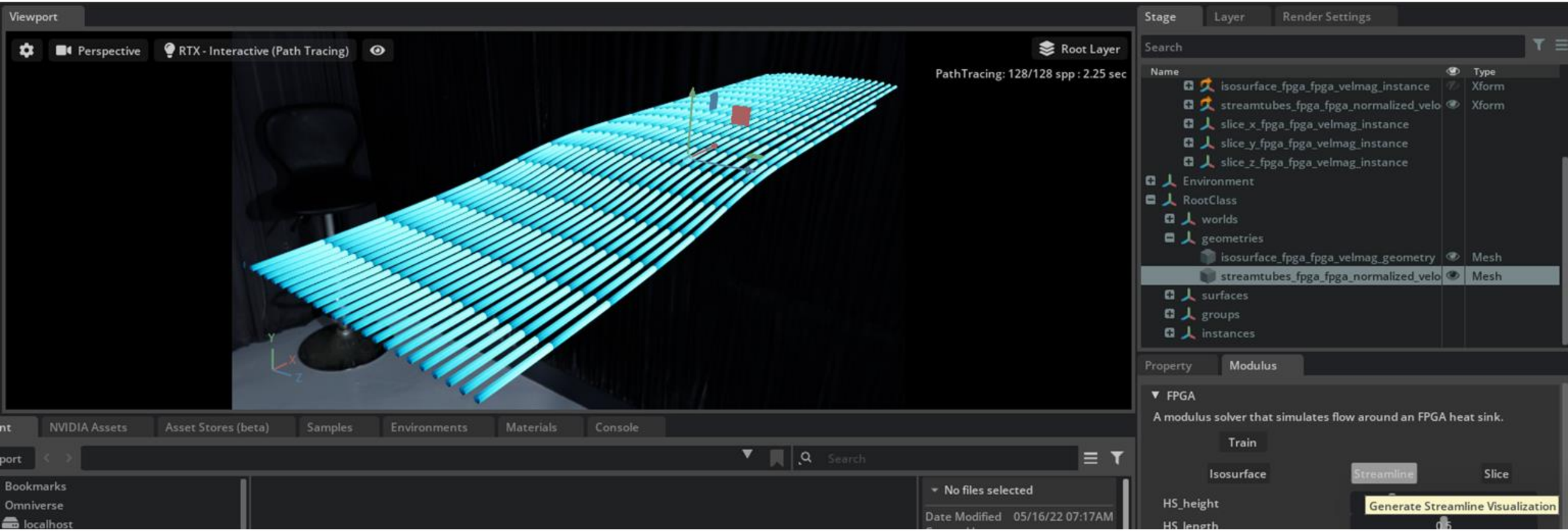


Fig. 37 *Multi-node scaling efficiency for the FPGA example*

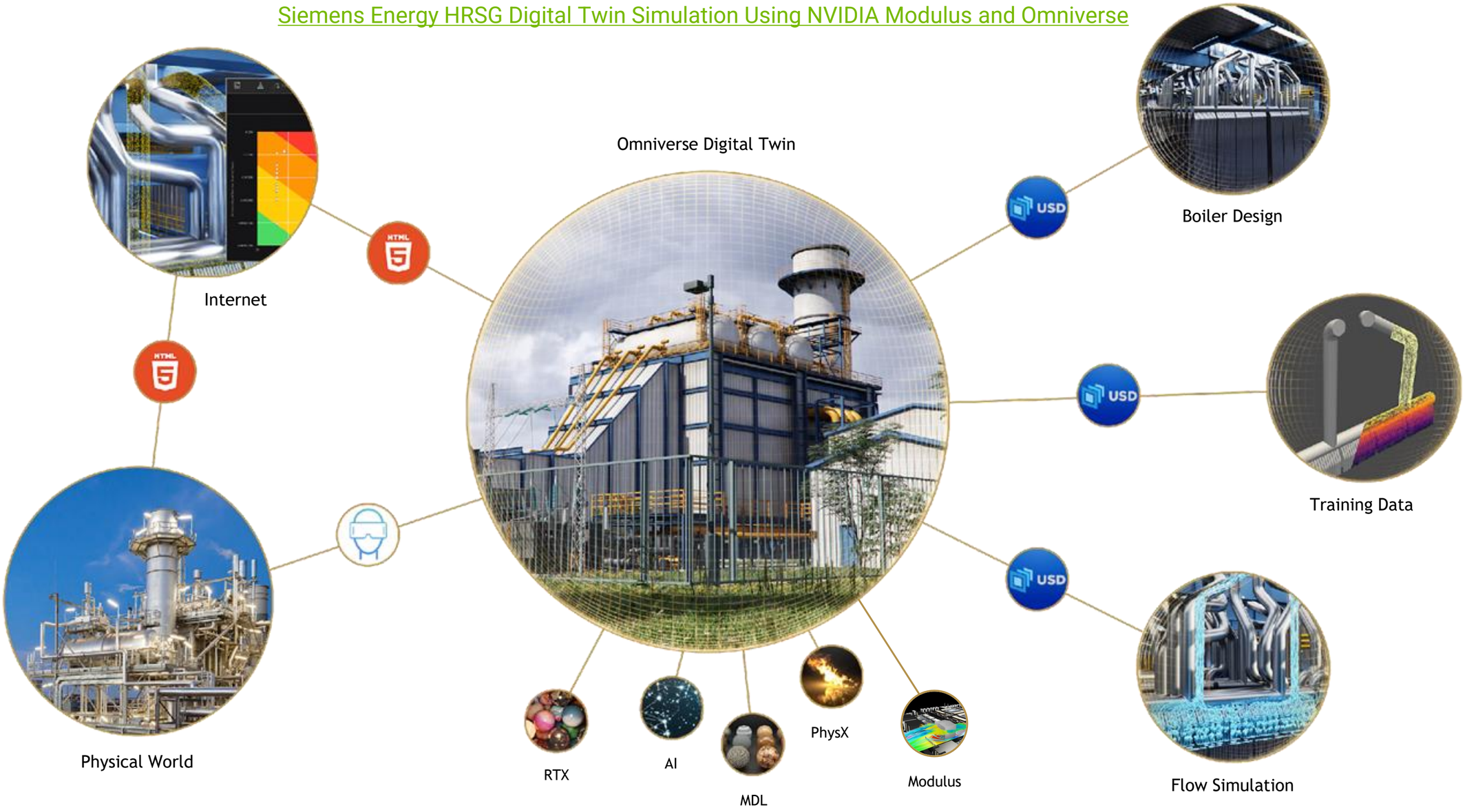
MODULUS – OMNIVERSE INTEGRATION



- Common visualization modes such as Isosurface, Streamlines, Slices are available.
- Each mode will populate the geometry, which will be updated as you change parameters.

SIEMENS ENERGY
BUILD HRSG DIGITAL TWIN USING NVIDIA MODULUS AND OMNIVERSE

Siemens Energy HRSG Digital Twin Simulation Using NVIDIA Modulus and Omniverse



NVIDIA MODULUS

V22.03.1

https://developer.nvidia.com/modulus

A Framework for Developing Physics Machine Learning Neural Network Models

NVIDIA Modulus is a neural network framework that blends the power of physics in the form of governing partial differential equations (PDEs) with data to build high-fidelity, parameterized surrogate models with near-real-time latency. Whether you're looking to get started with AI-driven physics problems or designing digital twin models for complex non-linear, multi-physics systems, NVIDIA Modulus can support your work.

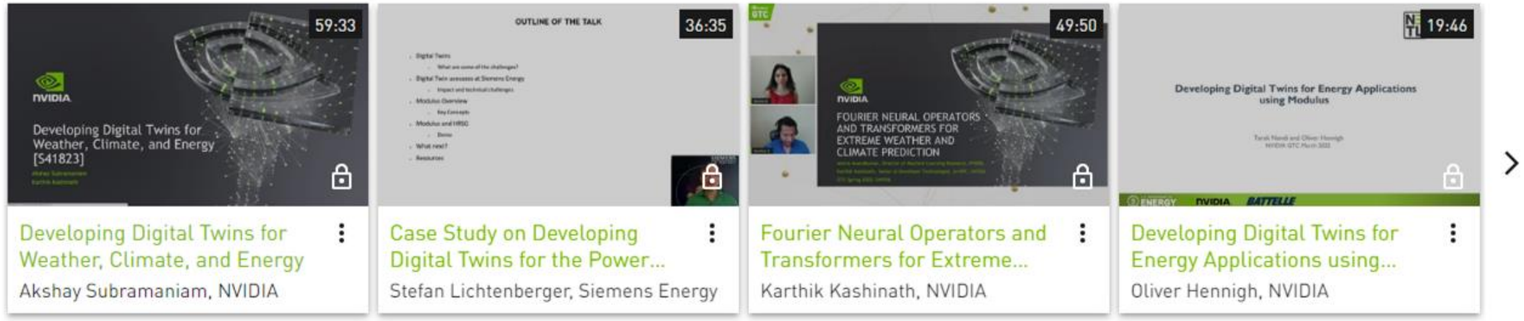
DOWNLOAD NOW | NGC DOWNLOAD

NVIDIA Developer Program Membership Required

The file or page you have requested requires membership in the NVIDIA Developer Program. Please either log in or join the program to access this material. Learn more about the benefits of the NVIDIA Developer Program.

Login | Join now

MODULUS FEATURED CONTENT (17 SESSIONS) SEE ALL



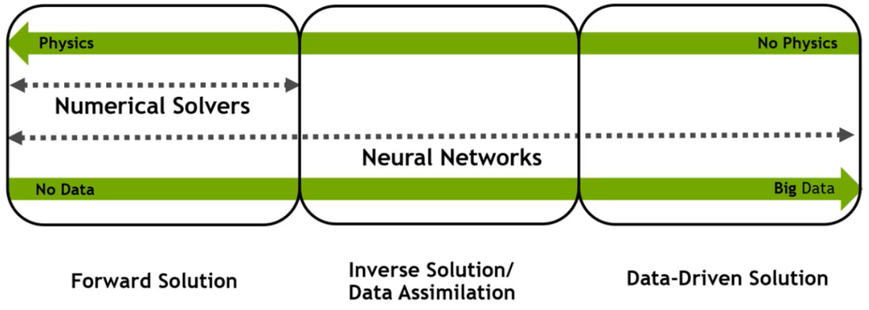
DOCUMENTATION

https://docs.nvidia.com/deeplearning/modulus/index.html

Modulus User Guide

NVIDIA Modulus is a neural network framework that blends the power of physics and partial differential equations (PDEs) with AI to build more robust models for better analysis.

There is a plethora of ways in which ML/NN models can be applied for physics-based systems. These can depend based on the availability of observational data and the extent of understanding of underlying physics. Based on these aspects, the ML/NN based methodologies can be broadly classified into forward (physics-driven), data-driven and hybrid approaches that involve both the physics and data assimilation.



With NVIDIA Modulus, we aim to provide researchers and industry specialists, various tools that will help accelerate your development of such models for the scientific discipline of your need. Experienced users can start with exploring the Modulus APIs and building the models while beginners can use this User Guide as a portal to explore the possibilities of AI in the domain of scientific computation. The User Guide comes in with several examples that will help you jumpstart your development of AI driven models.

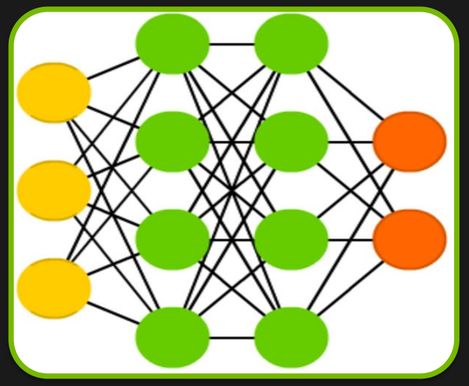
For Beginners

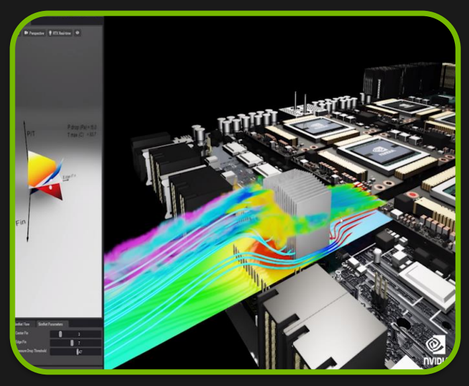
New to Modulus? No worries, we will get you up and running with physics driven AI in a few minutes. Check out the following tutorials to get started.

Install | Learn the Basics | Theory | Examples

MODULUS

AI-accelerated Physics Simulation Toolkit





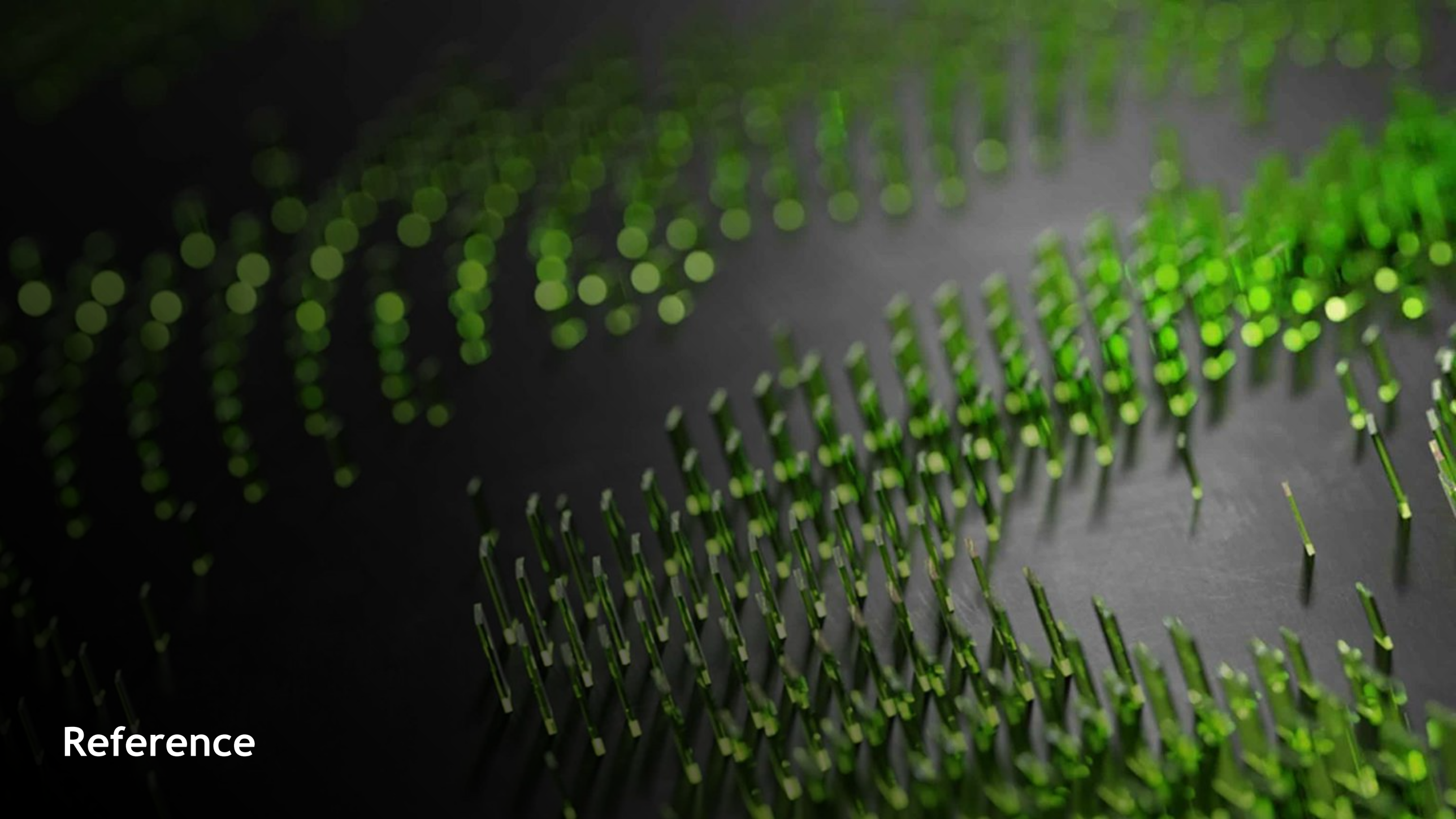
Solve larger problems faster
XLA, AMP TF32
Multi-GPU
Multi-Node

Advanced Model
Multiple Physics
Forward
Inverse
Data Assimilation

Solve
multiple scenarios simultaneously

APIs for
Physics
Geometry
Domains

User Guide examples



Reference

SIMULATION INTELLIGENCE: TOWARDS A NEW GENERATION OF SCIENTIFIC METHODS

https://arxiv.org/abs/2112.03235

(2021/12/06)

SIMULATION INTELLIGENCE: TOWARDS A NEW GENERATION OF SCIENTIFIC METHODS

Alexander Lavin* — Institute for Simulation Intelligence
Hector Zenil — Alan Turing Institute
Brooks Paige — Alan Turing Institute
David Krakauer — Santa Fe Institute
Justin Gottschlich — Intel Labs
Tim Mattson — Intel
Anima Anandkumar — Nvidia
Sanjay Choudry — Nvidia
Kamil Rocki — Neuralink
Atılım Güneş Baydin — University of Oxford
Carina Prunkl — University of Oxford
Olexandr Isayev — Carnegie Mellon University
Erik Peterson — Carnegie Mellon University
Peter L. McMahon — Cornell University
Jakob H. Macke — University of Tübingen
Kyle Cranmer — New York University
Jiaxin Zhang — Oak Ridge National Lab
Haruko Wainwright — Lawrence Berkeley National Lab
Adi Hanuka — SLAC National Accelerator Lab
Samuel Assefa — US Bank AI Innovation
Stephan Zheng — Salesforce Research
Manuela Veloso — JPM AI Research
Avi Pfeffer — Charles River Analytics

ABSTRACT

The original "Seven Motifs" set forth a roadmap of essential methods for the field of scientific computing, where a motif is an algorithmic method that captures a pattern of computation and data movement.[1] We present the *Nine Motifs of Simulation Intelligence*, a roadmap for the development and integration of the essential algorithms necessary for a merger of scientific computing, scientific simulation, and artificial intelligence. We call this merger simulation intelligence (SI), for short. We argue the motifs of simulation intelligence are interconnected and interdependent, much like the components within the layers of an operating system. Using this metaphor, we explore the nature of each layer of the simulation intelligence "operating system" stack (SI-stack) and the motifs therein:

1. Multi-physics and multi-scale modeling
2. Surrogate modeling and emulation
3. Simulation-based inference
4. Causal modeling and inference
5. Agent-based modeling
6. Probabilistic programming
7. Differentiable programming
8. Open-ended optimization
9. Machine programming

We believe coordinated efforts between motifs offers immense opportunity to accelerate scientific discovery, from solving inverse problems in synthetic biology and climate science, to directing nuclear energy experiments and predicting emergent behavior in socioeconomic settings. We elaborate on each layer of the SI-stack, detailing the state-of-art methods, presenting examples to highlight challenges and opportunities, and advocating for specific ways to advance the motifs and the synergies from their combinations. Advancing and integrating these technologies can enable a robust and efficient hypothesis–simulation–analysis type of scientific method, which we introduce with several use-cases for human-machine teaming and automated science.

FOURIER FEATURES LET NETWORKS LEARN HIGH FREQUENCY FUNCTIONS IN LOW DIMENSIONAL DOMAINS

https://bmild.github.io/fourfeat/

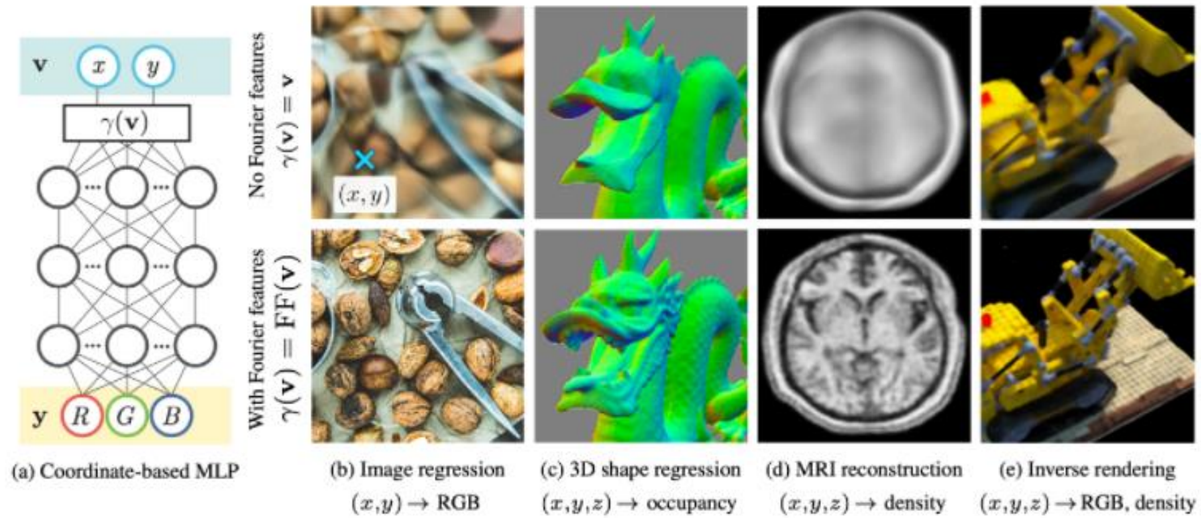
Matthew Tancik* Pratul Srinivasan* Ben Mildenhall* Sara Fridovich-Keil
UC Berkeley UC Berkeley UC Berkeley UC Berkeley

Nithin Raghavan Utkarsh Singhal Ravi Ramamoorthi Jonathan T. Barron Ren Ng
UC Berkeley UC Berkeley UC San Diego Google Research UC Berkeley

*denotes equal contribution

Paper Code

Abstract



(a) Coordinate-based MLP (b) Image regression $(x,y) \rightarrow$ RGB (c) 3D shape regression $(x,y,z) \rightarrow$ occupancy (d) MRI reconstruction $(x,y,z) \rightarrow$ density (e) Inverse rendering $(x,y,z) \rightarrow$ RGB, density

NVIDIA.


反饋問卷
https://forms.gle/i9pdsodoYCRuZEjL7
NVIDIA