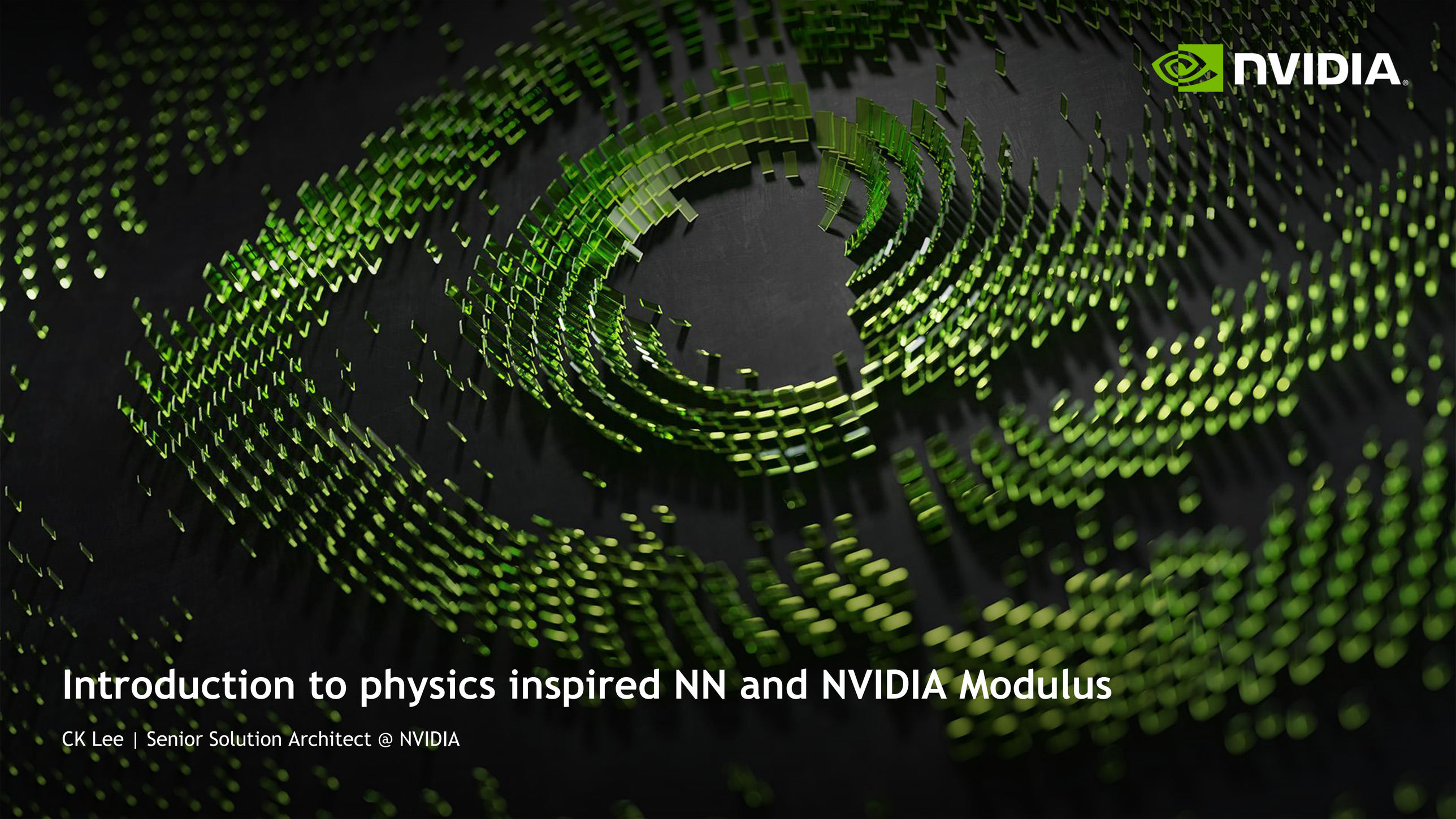


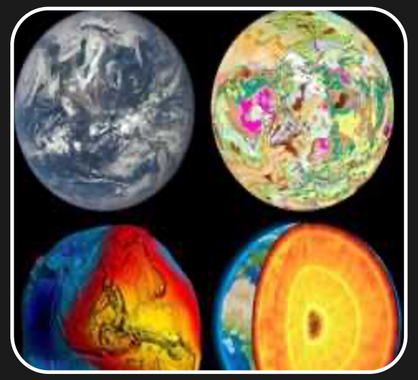
Introduction to physics inspired NN and NVIDIA Modulus

CK Lee | Senior Solution Architect @ NVIDIA

COMPUTATIONAL DOMAINS

Computational Eng.

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

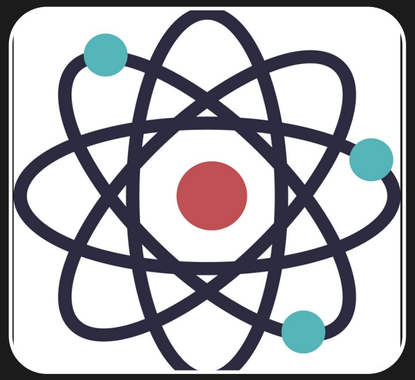


Earth Sciences

Climate Modeling, Weather Modeling, Ocean Modeling, Seismic Interpretation

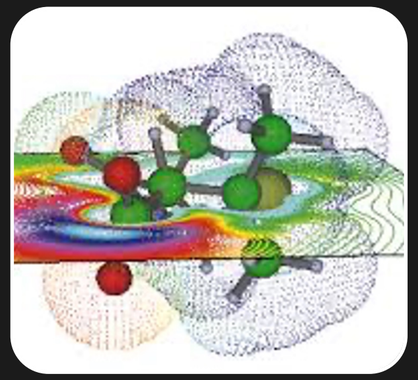
Life Sciences

Genomics, Proteomics



Computational Physics

Particle Science, Astrophysics

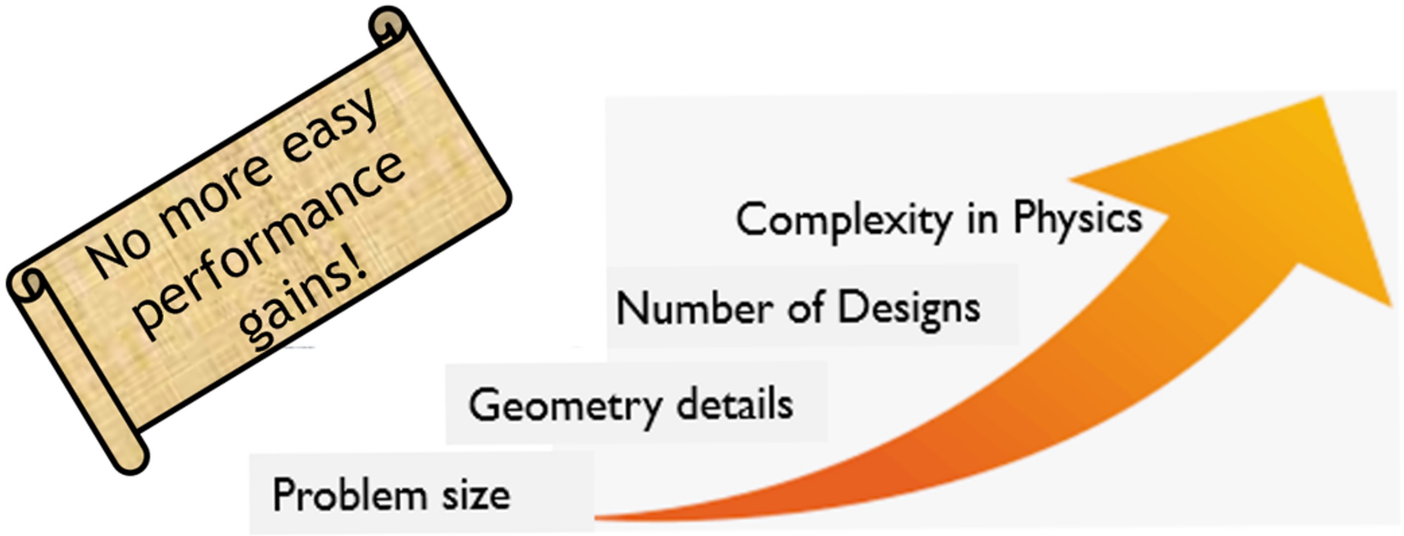


Computational Chemistry

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

THE CONVERGENCE OF HPC * AI

TO ENABLE SOLUTIONS TO GRAND CHALLENGE PROBLEMS

HPC

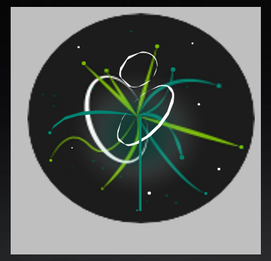
40+ years of algorithms based on first principles theory

AI

New algorithms and models with potential to increase model size and accuracy

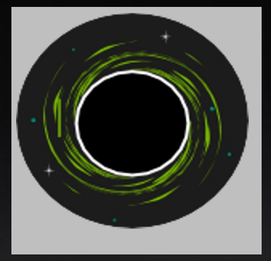
Dramatically Improves Accuracy and /or Time-to-Solution at Large Scale

Commercially viable fusion energy



Improve or validate the Standard Model of Physics

Clinically viable precision medicine

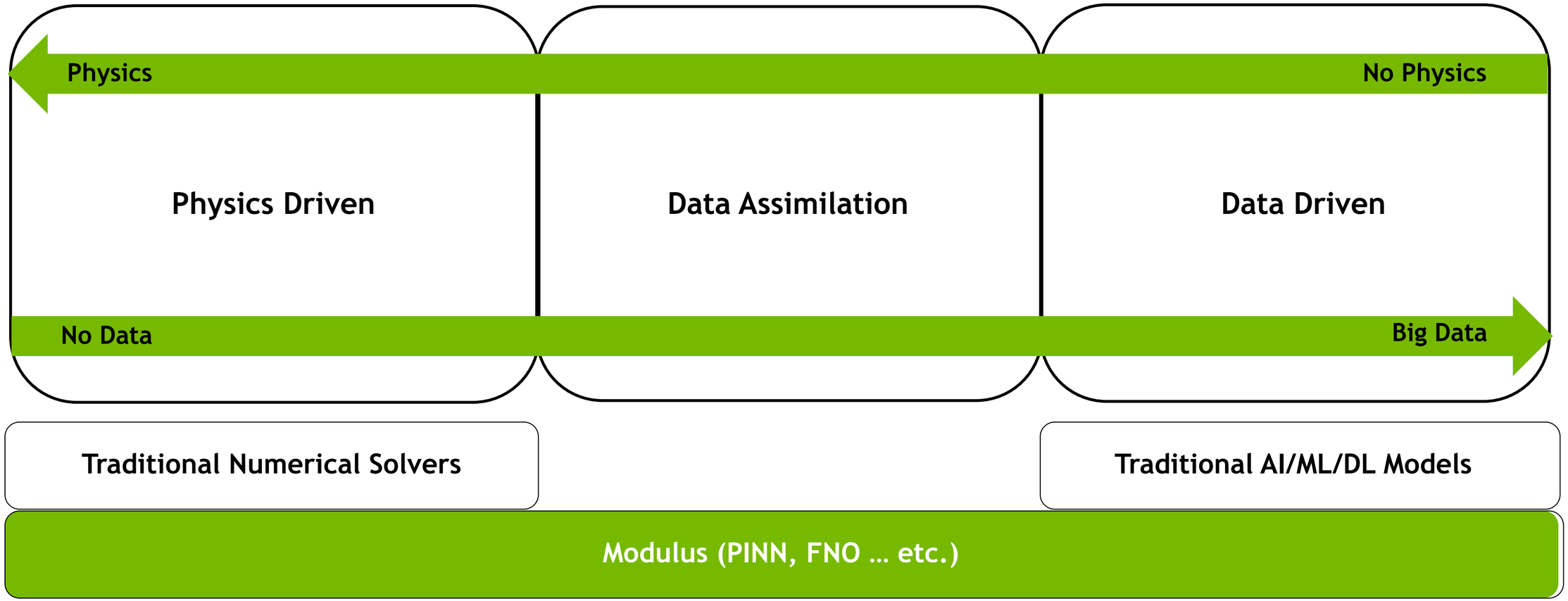


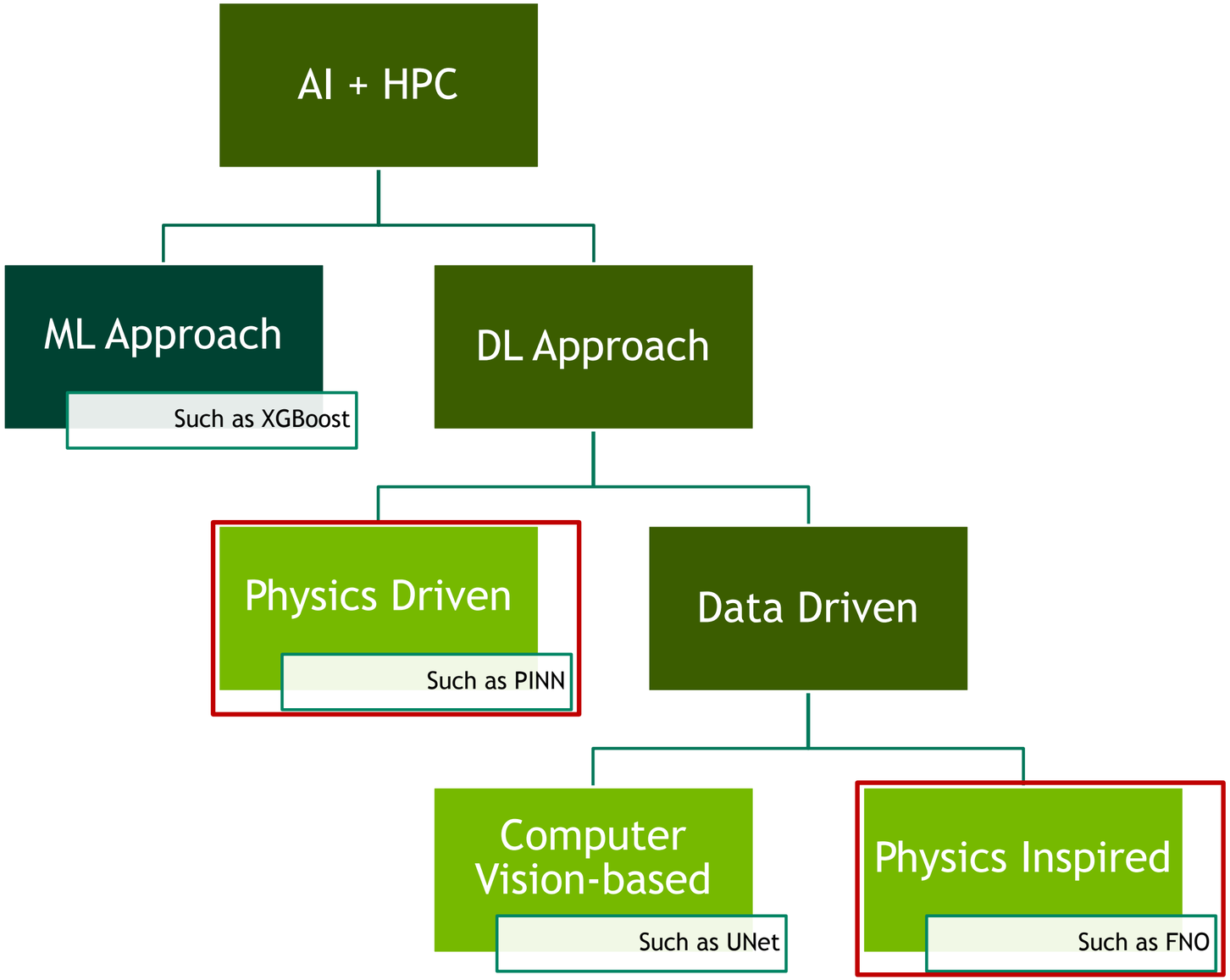
Understanding cosmological dark energy and matter

Climate/weather forecasts with ultra-high fidelity

AI IN COMPUTATIONAL SCIENCES

Primary Driver: Data vs. Physics



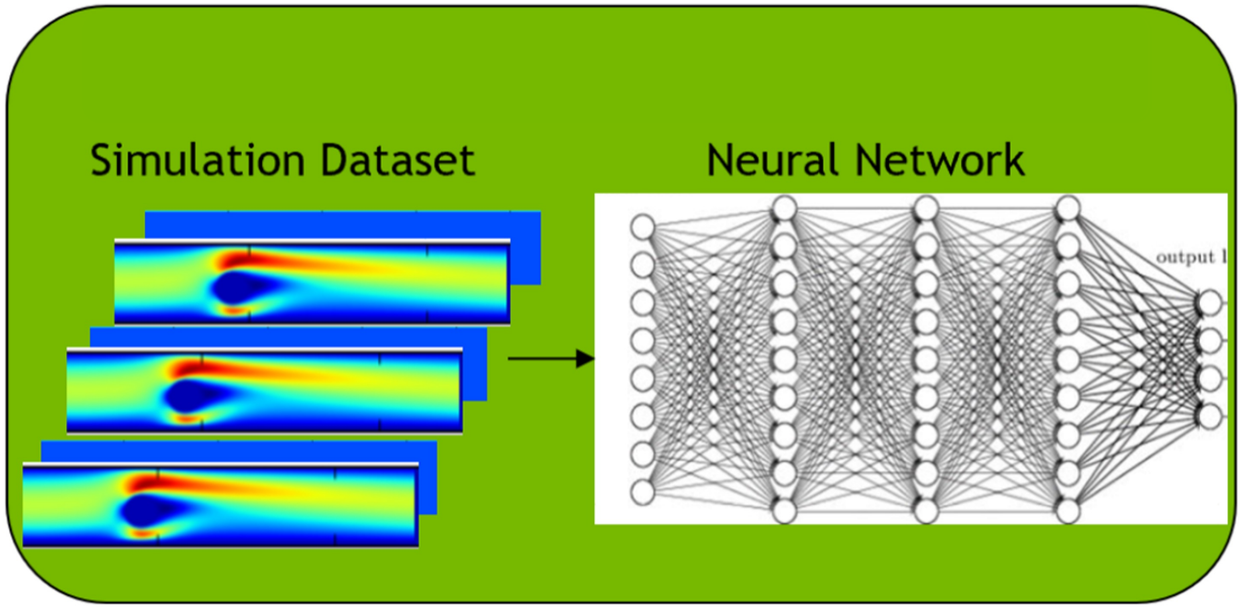

AI + HPC
ML Approach
Such as XGBoost
DL Approach
Physics Driven
Such as PINN
Data Driven
Computer Vision-based
Such as UNet
Physics Inspired
Such as FNO

NVIDIA Modulus

Solving PDEs with neural networks

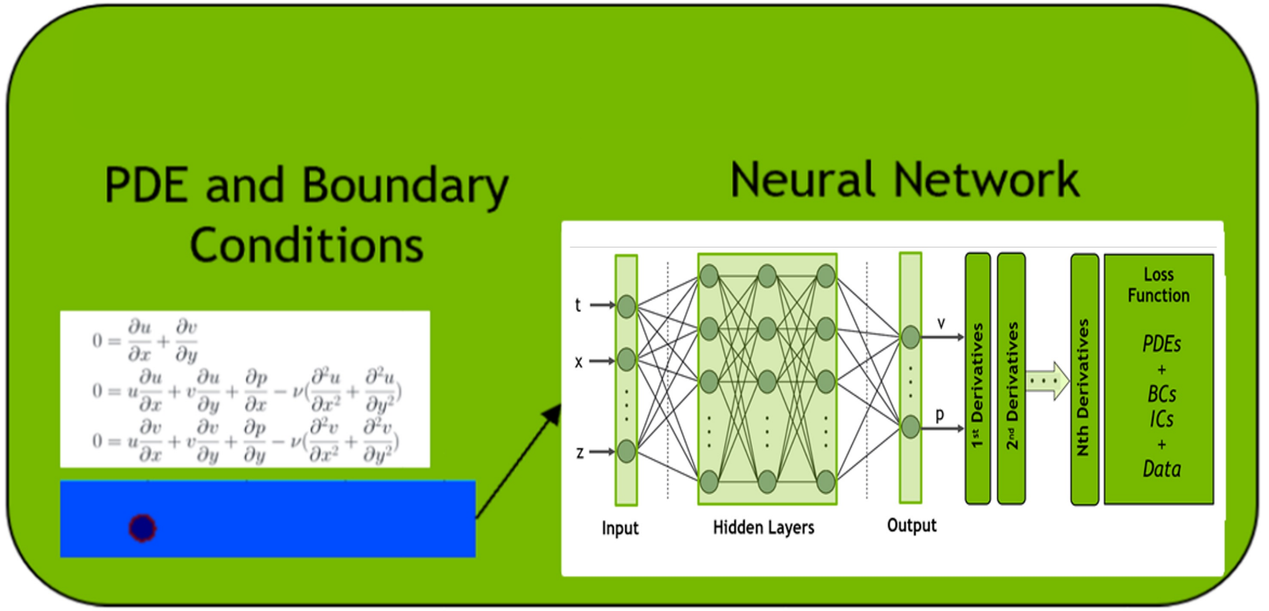
https://docs.nvidia.com/deeplearning/modulus/index.html

Physics Inspired Neural Operator
(learn the entire family of PDEs)

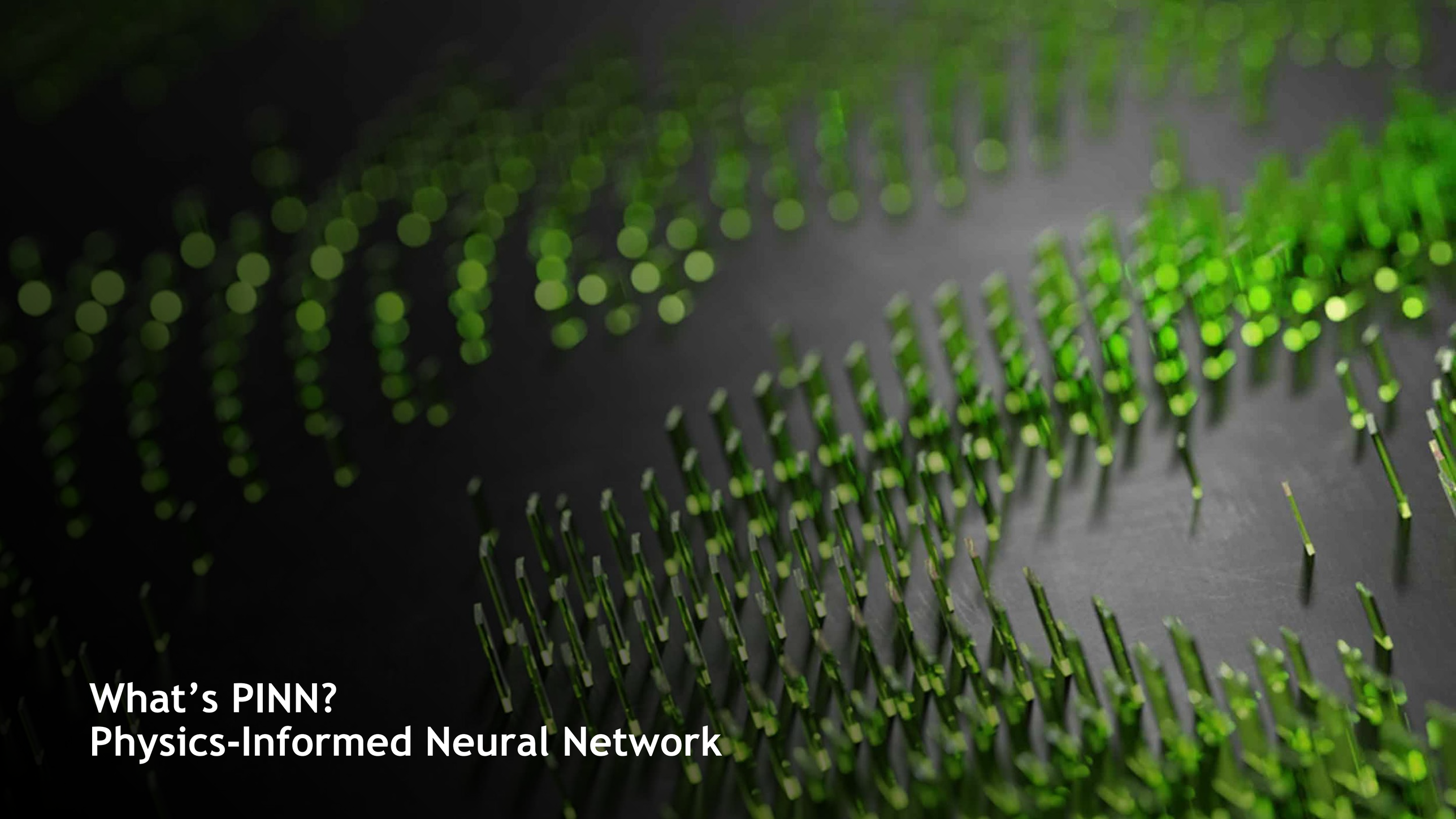


N layers

Physics Informed Neural Network
(No training data is required)



m*N layers (for mth order PDE)


What’s PINN?
Physics-Informed Neural Network

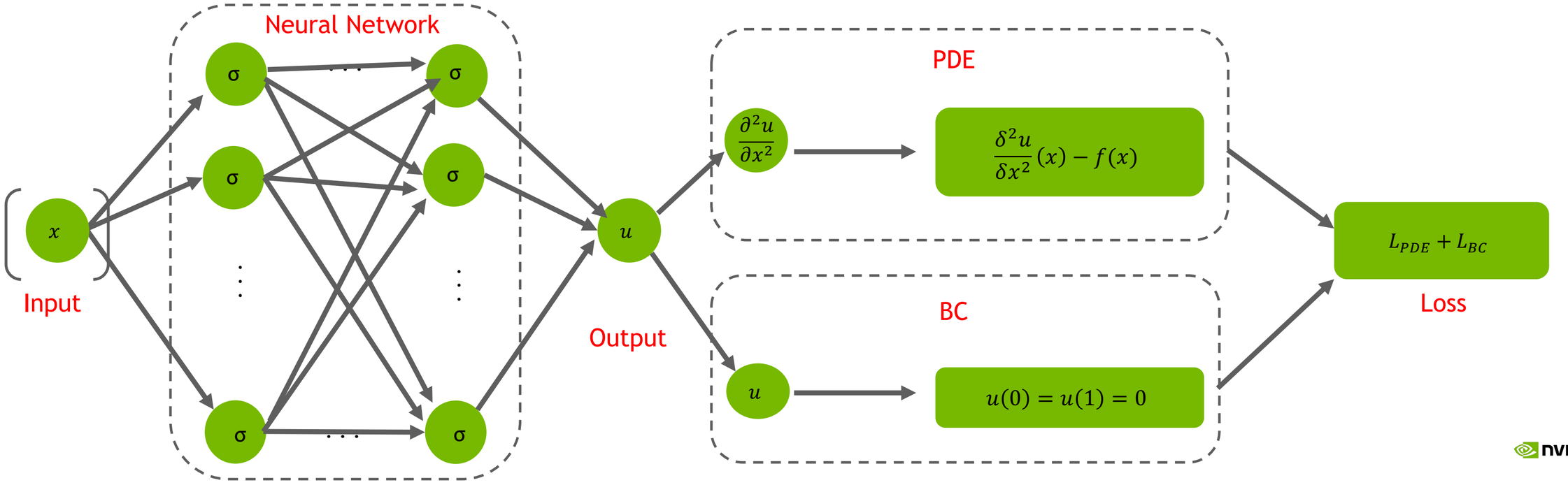
NEURAL NETWORK SOLVER THEORY

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

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

Forward Problem

- 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

- 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**

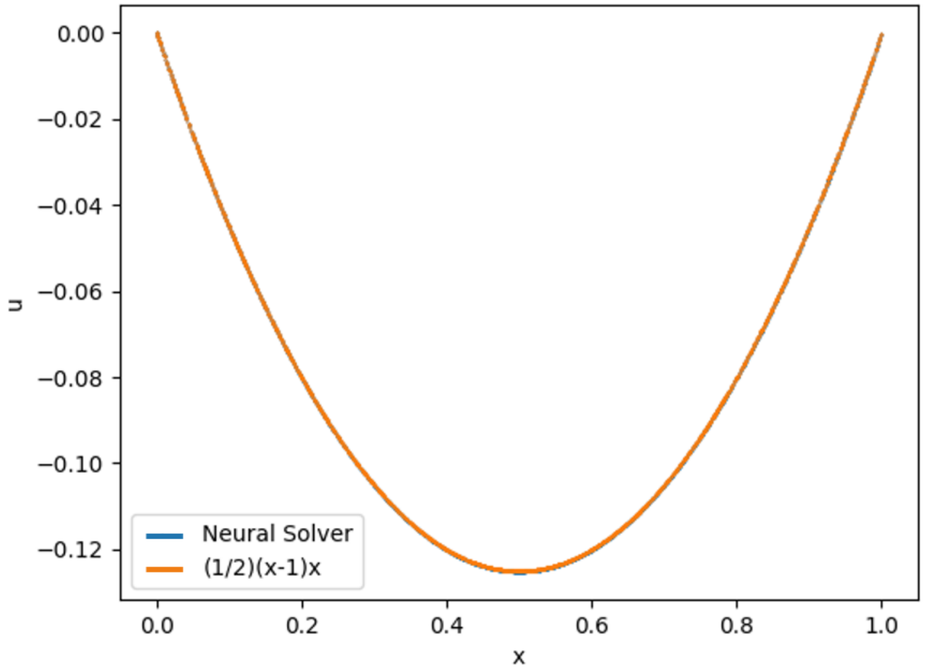
$$L_{BC} = u_{net}(0)^2 + u_{net}(1)^2$$

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

- 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

NEURAL NETWORK SOLVER THEORY

- 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:

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

Parameterized Problem

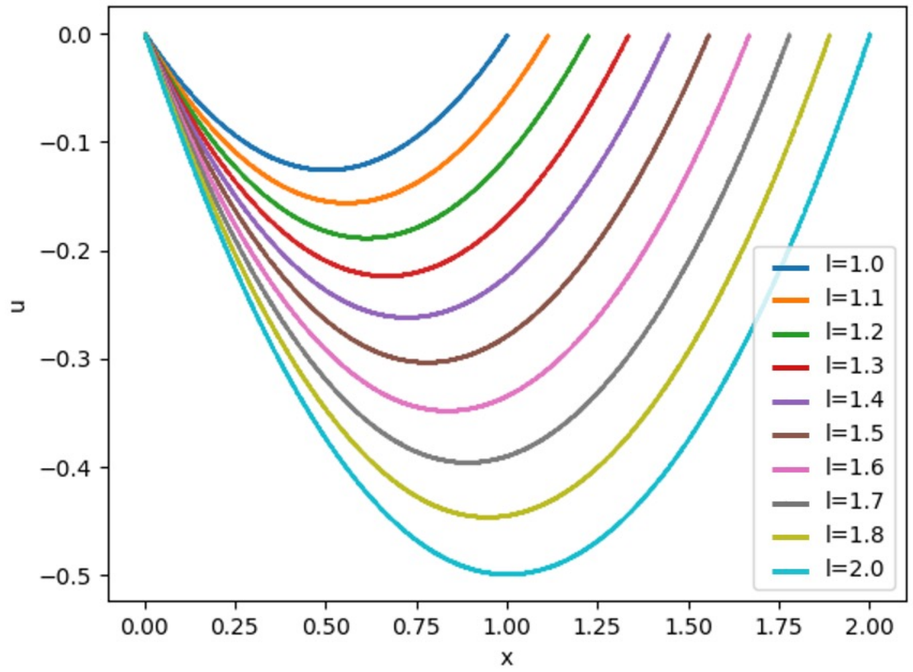
- This time, we construct a neural network $u_{net}(x, l)$ which has x and l as input and single value output $u_{net}(x, l) \in \mathbb{R}$.
- The losses become

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

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

SOLVING PARAMETERIZED PROBLEMS

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



Solution to the parametric problem

SOLVING INVERSE PROBLEMS

- 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 $f(x)$

$$\mathbf{P}: \begin{cases} u_{true}(x) = \dfrac{1}{48}\left(8x(-1+x^2) - \dfrac{3\sin(4\pi x)}{\pi^2}\right) \\ \dfrac{\delta^2 u_{net}}{\delta x^2}(x) = f_{net}(x) \\ u_{net}(0) = u_{net}(1) = 0 \end{cases}$$

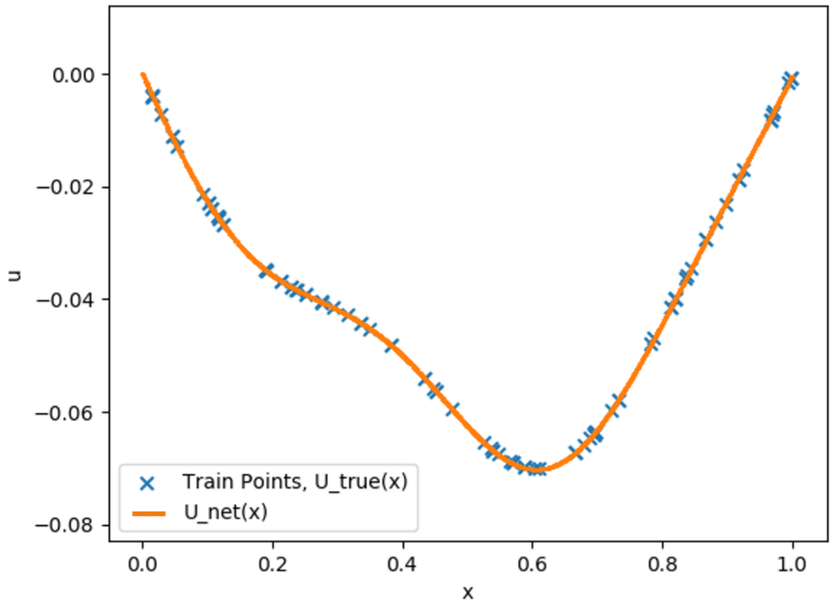
Inverse Problem

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

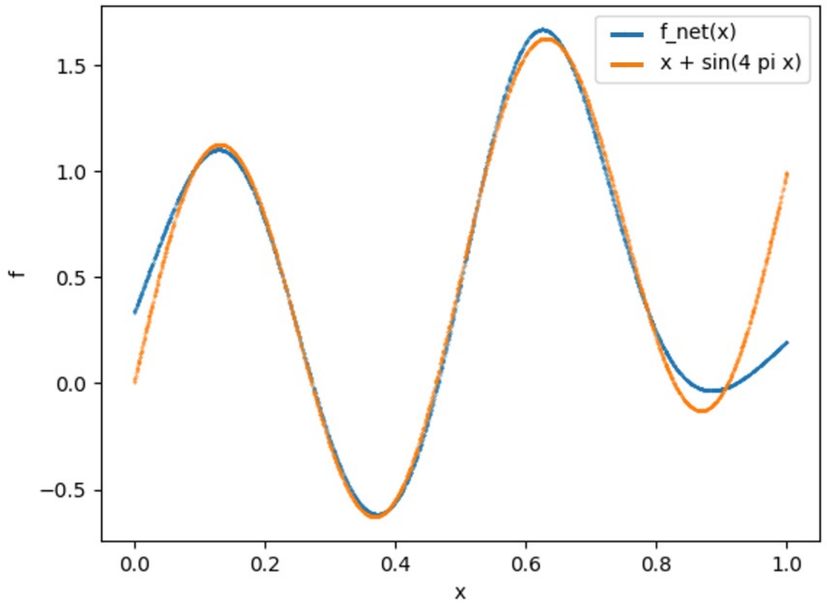
$$L_{Data} = \frac{1}{100}\sum_{i=0}^{100}\left(u_{net}(x_i) - u_{true}(x_i)\right)^2$$

SOLVING INVERSE PROBLEMS

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



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



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

MODULUS

is capable to solve …

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

Forward Problem

P: $\begin{cases} \dfrac{\delta^2 u_{net}}{\delta x^2}(x, \boldsymbol{l}) = f(x) \\ u_{net}(0, \boldsymbol{l}) = u_{net}(\boldsymbol{l}, \boldsymbol{l}) = 0 \end{cases}$

Parameterized Problem

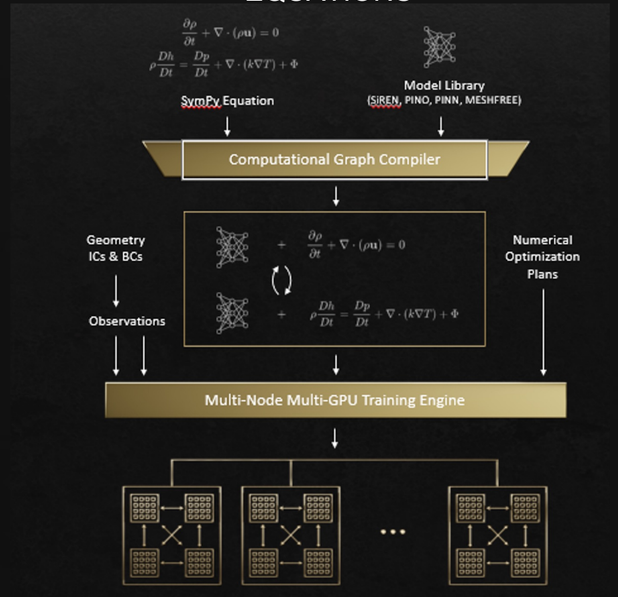
P: $\begin{cases} u_{true}(x) = \dfrac{1}{48}\left(8x(-1 + x^2) - \dfrac{3\sin(4\pi x)}{\pi^2}\right) \\ \dfrac{\delta^2 u_{net}}{\delta x^2}(x) = f_{net}(x) \\ u_{net}(0) = u_{net}(1) = 0 \end{cases}$

Inverse Problem

NVIDIA MODULUS

Framework for developing physics machine learning neural network models

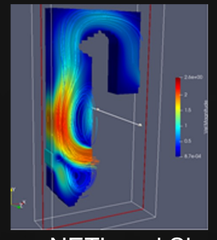
TRAINING USING BOTH DATA AND THE GOVERNING EQUATIONS

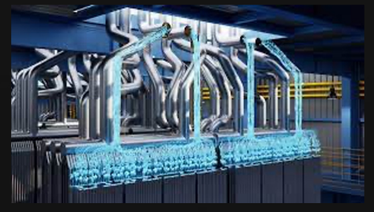


Modulus Latest Release (v. 22.09) Key Highlights:

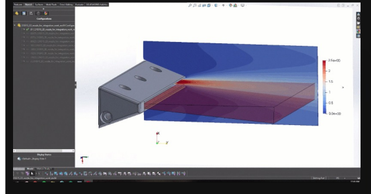
- FNO/AFNO integration to create climate physics-ML models
- Omniverse integration to visualize, infer, and interact in real-time with physics-ML model outputs

DEVELOP HIGH FIDELITY DIGITAL TWINS

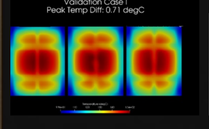


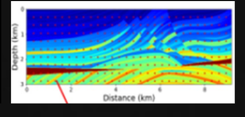


NETL and Siemens Energy use Modulus to build Digital Twins – high fidelity surrogate models – 10,000x faster



Using parameterized models for design optimization





Generalizable methodology for different domains such as fluids, Solid Mechanics, Multiphysics, etc.

ADOPTION BY LEADING RESEARCH INSTITUTIONS

SIEMENS energy

NETL NATIONAL ENERGY TECHNOLOGY LABORATORY

kinetic vision

COLLABORATION PARTNERS

BROWN

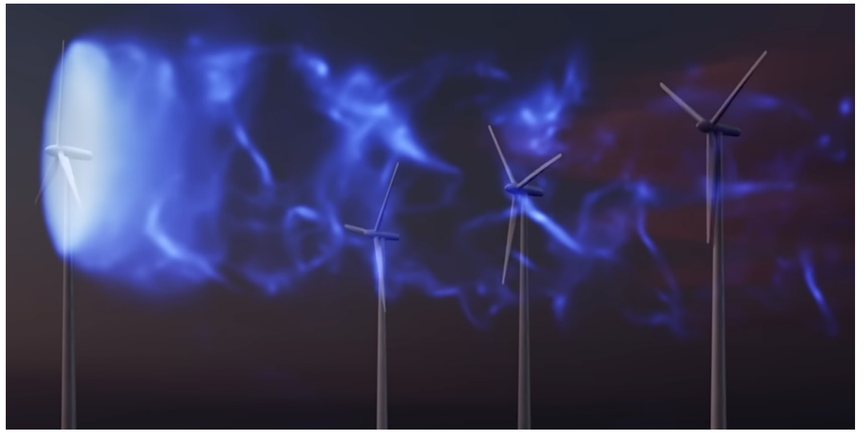
Caltech

rescale

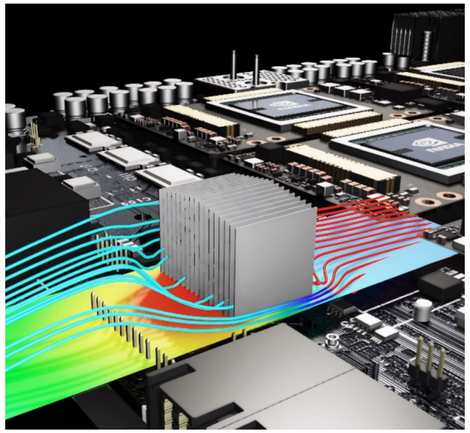
NVIDIA Modulus

NVIDIA Modulus is a neural network framework that blends the power of physics in form of governing partial differential equations (PDEs) with data to build high-fidelity, parameterized surrogate models with near-real-time latency. It offers:

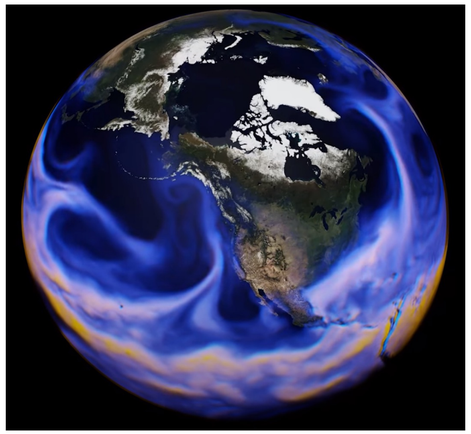
- **AI Toolkit**
  - Offers building blocks for developing physics-ML surrogate models
- **Scalable Performance**
  - Solves larger problems faster by scaling from single GPU to multi-node implementation
- **Near-Real-Time Inference**
  - Provides parameterized system representation that solves for multiple scenarios in near real time, trains once offline to infer in real time repeatedly
- **Easy Adoptability**
  - Includes APIs for domain experts to work at a higher level of abstraction. Extensible to new applications with reference applications serving as starting points



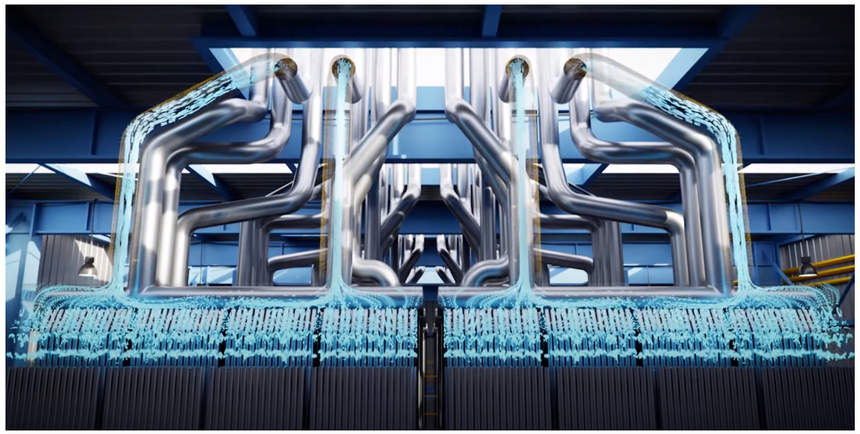
Wind Farm Super Resolution



FPGA Heatsink Design Optimization



Extreme Weather Prediction

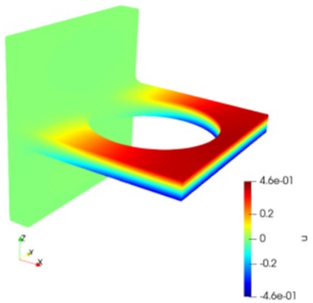


Industrial Digital Twin

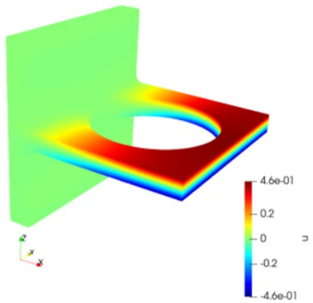
What is Modulus?

Modulus is a PDE solver

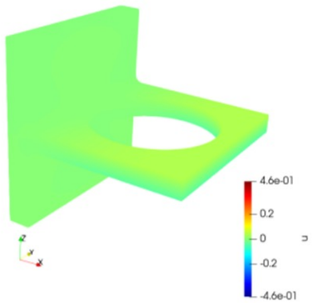
- Like traditional solvers such as Finite Element, Finite Difference, Finite Volume, and Spectral solvers, Modulus can solve PDEs



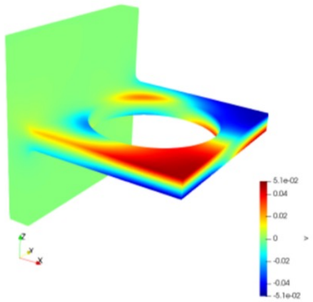
(a) u (Modulus)



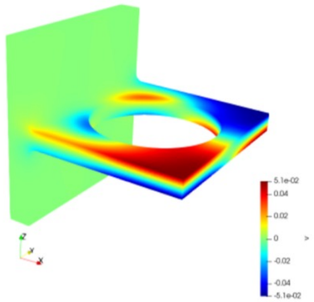
(b) u (Commercial solver)



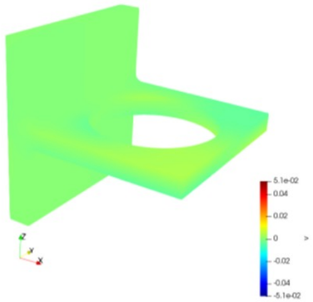
(c) u (Difference)



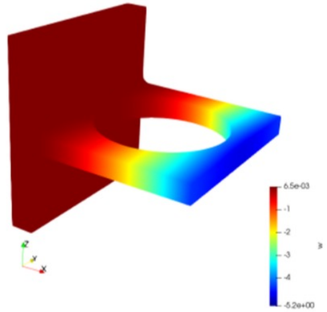
(d) v (Modulus)



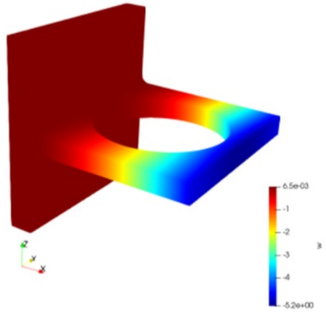
(e) v (Commercial solver)



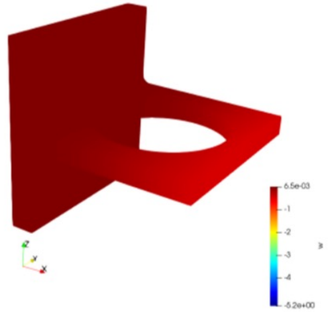
(f) v (Difference)



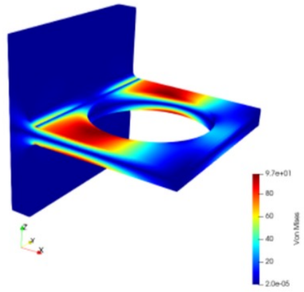
(g) w (Modulus)



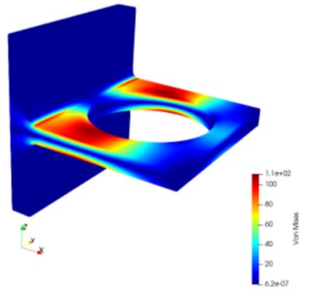
(h) w (Commercial Solver)



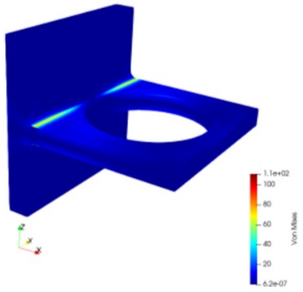
(i) w (Difference)



(j) σ_{VM} (Modulus)



(k) σ_{VM} (Commercial Solver)



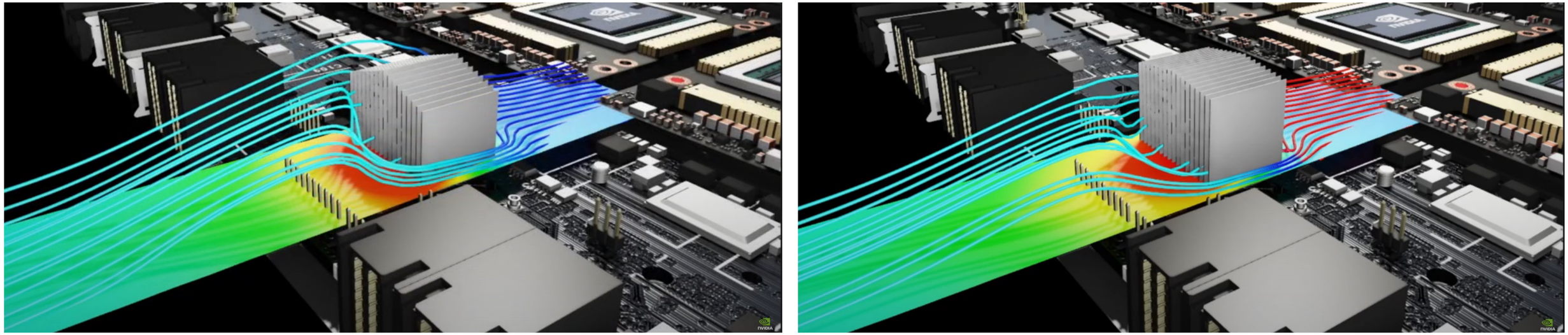
(l) σ_{VM} (Difference)

A comparison between Modulus and commercial solver results for a bracket deflection example. Linear elasticity equations are solved here.

What is Modulus?

Modulus is a tool for efficient design optimization for multi-physics problems

- With Modulus, professionals in Manufacturing and Product Development can explore different configurations and scenarios of a model, in near-real time by, changing its parameters, allowing them to gain deeper insights about the system or product, and to perform efficient design optimization of their products.

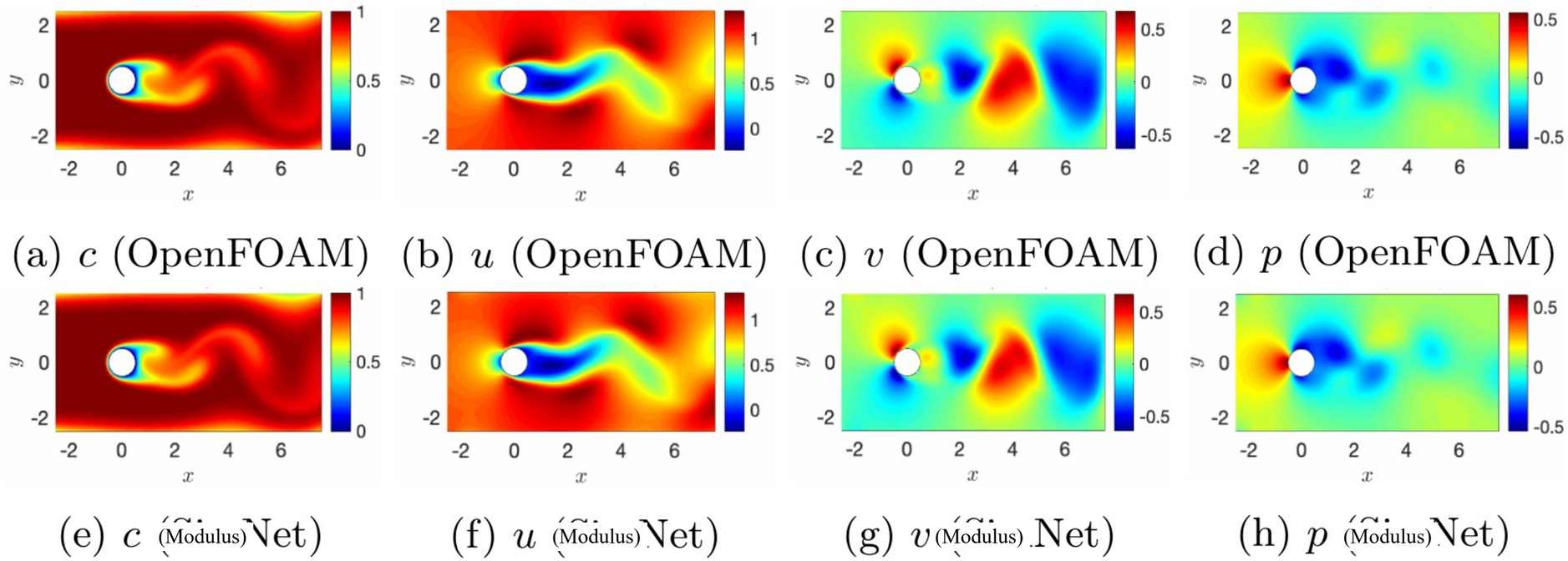


Efficient design space exploration of the heatsink of a Field-Programmable Gate Array (FPGA) using Modulus.

What is Modulus?

Modulus is a solver for inverse problems

- Many applications in science and engineering involve inferring unknown system characteristics given measured data from sensors or imaging.
- By combining data and physics, Modulus can effectively solve inverse problems.



A comparison between Modulus and OpenFOAM results for the flow velocity, pressure and passive scalar concentration fields. Modulus has inferred the velocity and pressure fields using scattered data from passive scalar concentration.

What is Modulus?

Modulus is a tool for developing digital twins

- A digital twin is a virtual representation (a true-to-reality simulation of physics) of a real-world physical asset or system, which is continuously updated via stream of data.
- Digital twin predicts the future state of the real-world system under varying conditions.

What is Modulus?

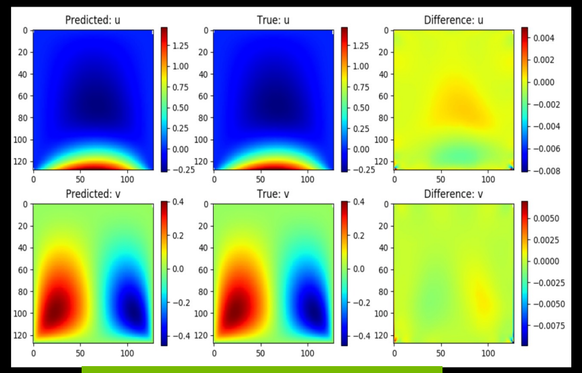
Modulus is a tool for developing data-driven solutions to engineering problems

- Modulus contains a variety of APIs for developing data-driven machine learning solutions to challenging engineering systems, including:
  - Data-driven modeling of physical systems
  - Super resolution of low-fidelity results computed by traditional solvers

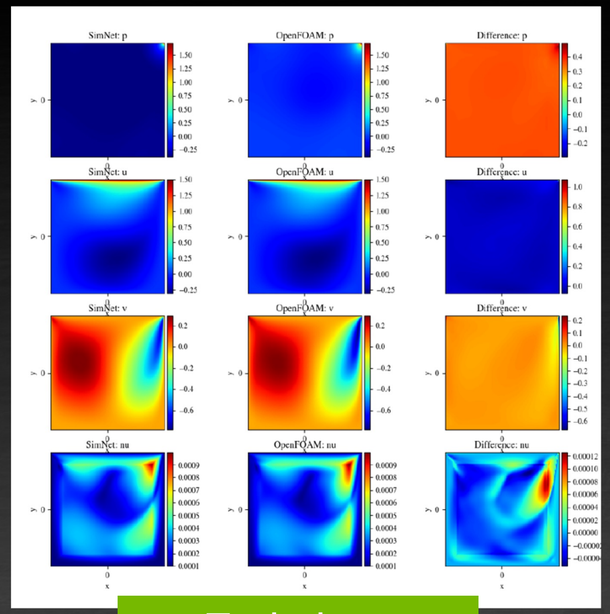
Super-resolution of flow in a wind farm using Modulus

Modulus Framework - Generalization

CFD

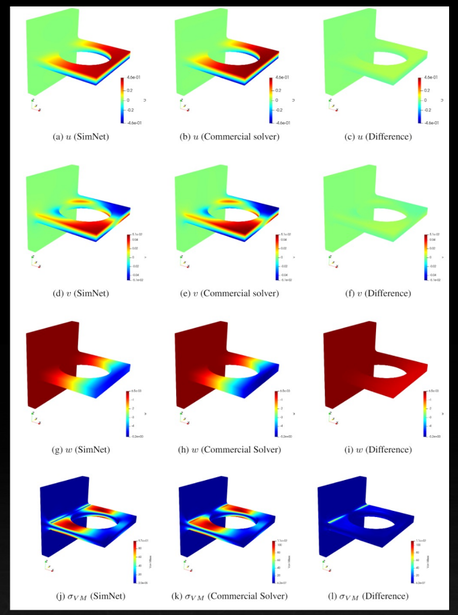


Laminar



Turbulent

Solid Mechanics

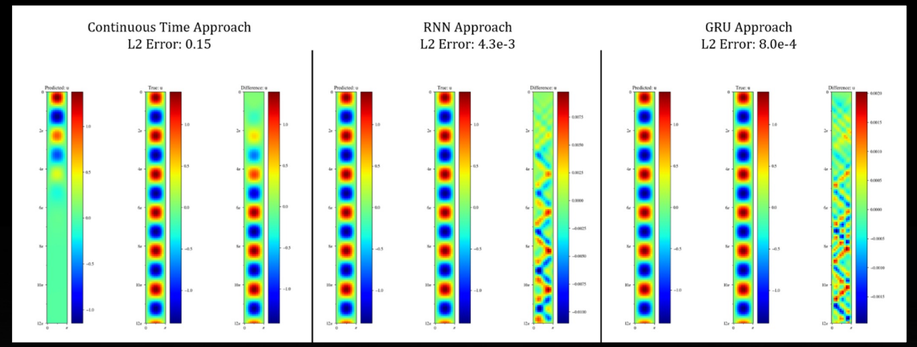


Equilibrium: $\sigma_{ji,j} + f_i = 0,$

Stress-Strain: $\sigma_{ij} = \lambda \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij},$

Strain-Displacement: $\epsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right).$

Acoustics



$$
\begin{aligned}
u_{tt} &= c^2 u_{xx} \\
u(0,t) &= 0, \\
u(\pi,t) &= 0, \\
u(x,0) &= \sin(x), \\
u_t(x,0) &= \sin(x).
\end{aligned}
$$

Modulus Framework - Generalization

Electromagnetics

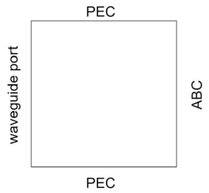


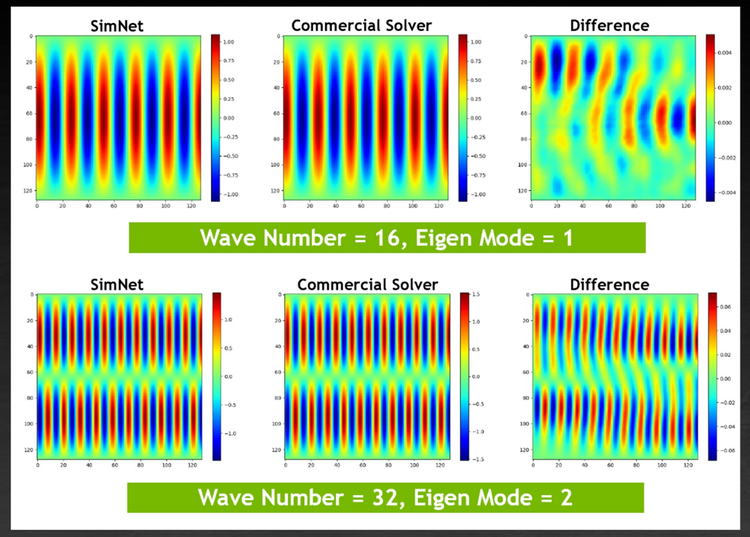
Figure 42: Domain of 2D waveguide

In this example we will solve this waveguide problem by transverse-magnetic (TM_z) mode, so that our unknown variable is $E_z(x, y)$. The governing equation in Ω is

$$\Delta E_z(x, y) + k^2 E_z(x, y) = 0,$$

where k is the wavenumber. Notice in 2D scalar case, the PEC and ABC will be simplified in the following form, respectively:

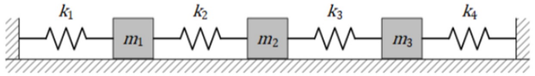
$$E_z(x, y) = 0 \text{ on top and bottom boundaries, } \frac{\partial E_z}{\partial y} = 0 \text{ on right boundary.}$$



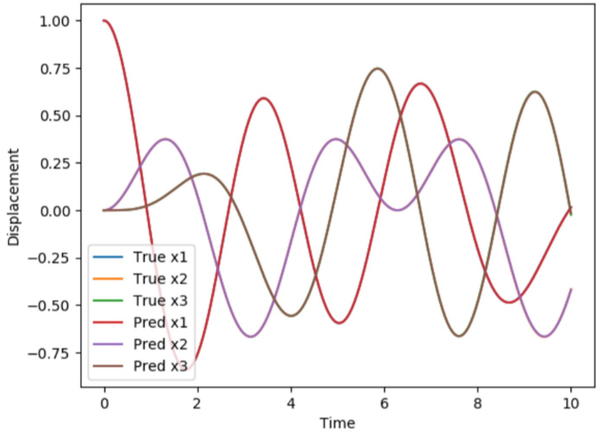
Vibrations

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

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

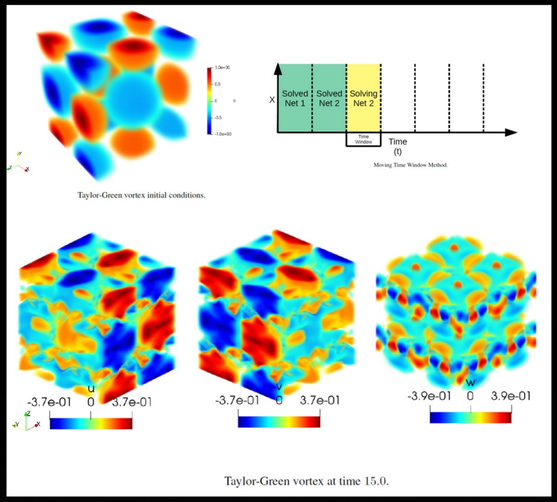


Three masses connected by four springs on a friction-less surface

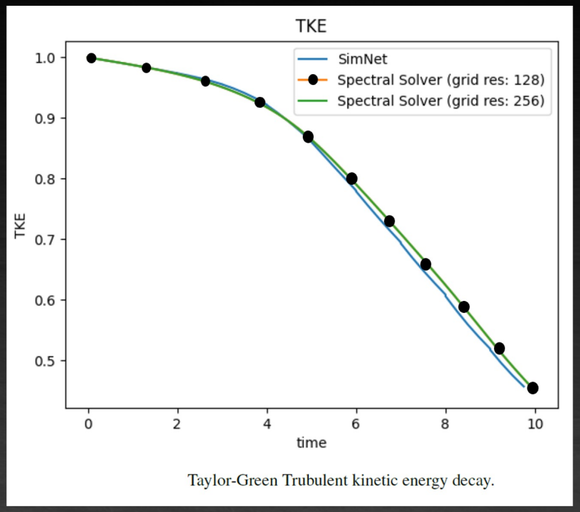


Comparison of SimNet results with analytical solution

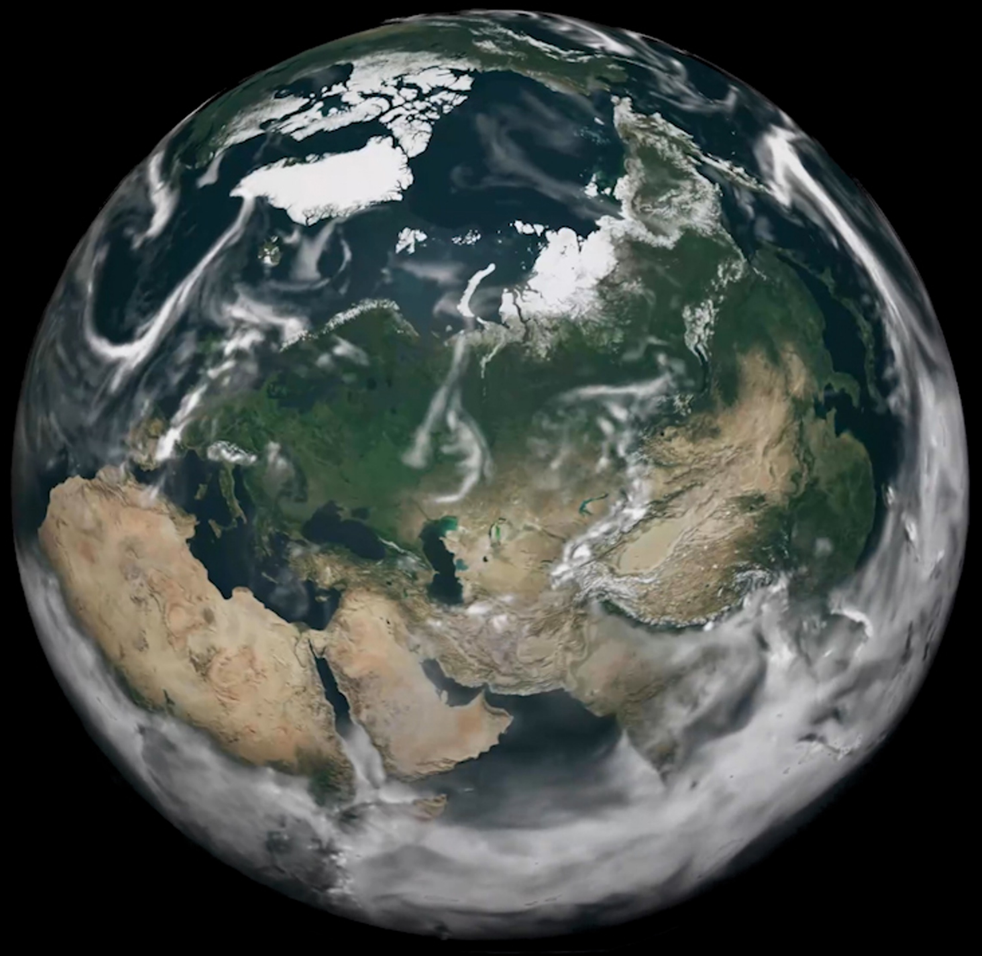
Turbulence



Taylor-Green vortex at time 15.0.



Taylor-Green Trubulent kinetic energy decay.



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

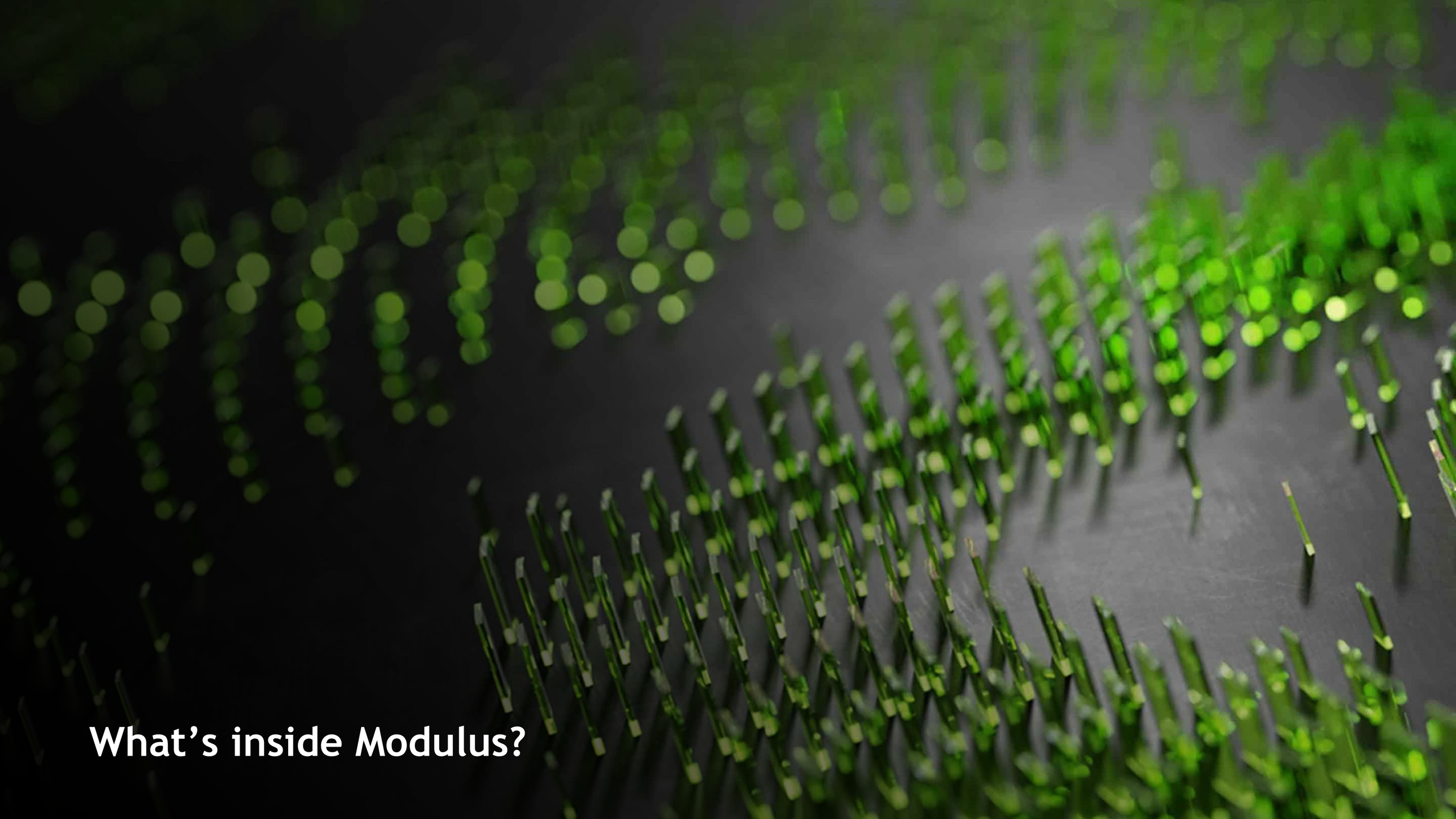
- Hardware: NVIDIA A100
- Software: NVIDIA Omniverse, NVIDIA Modulus

Outcome

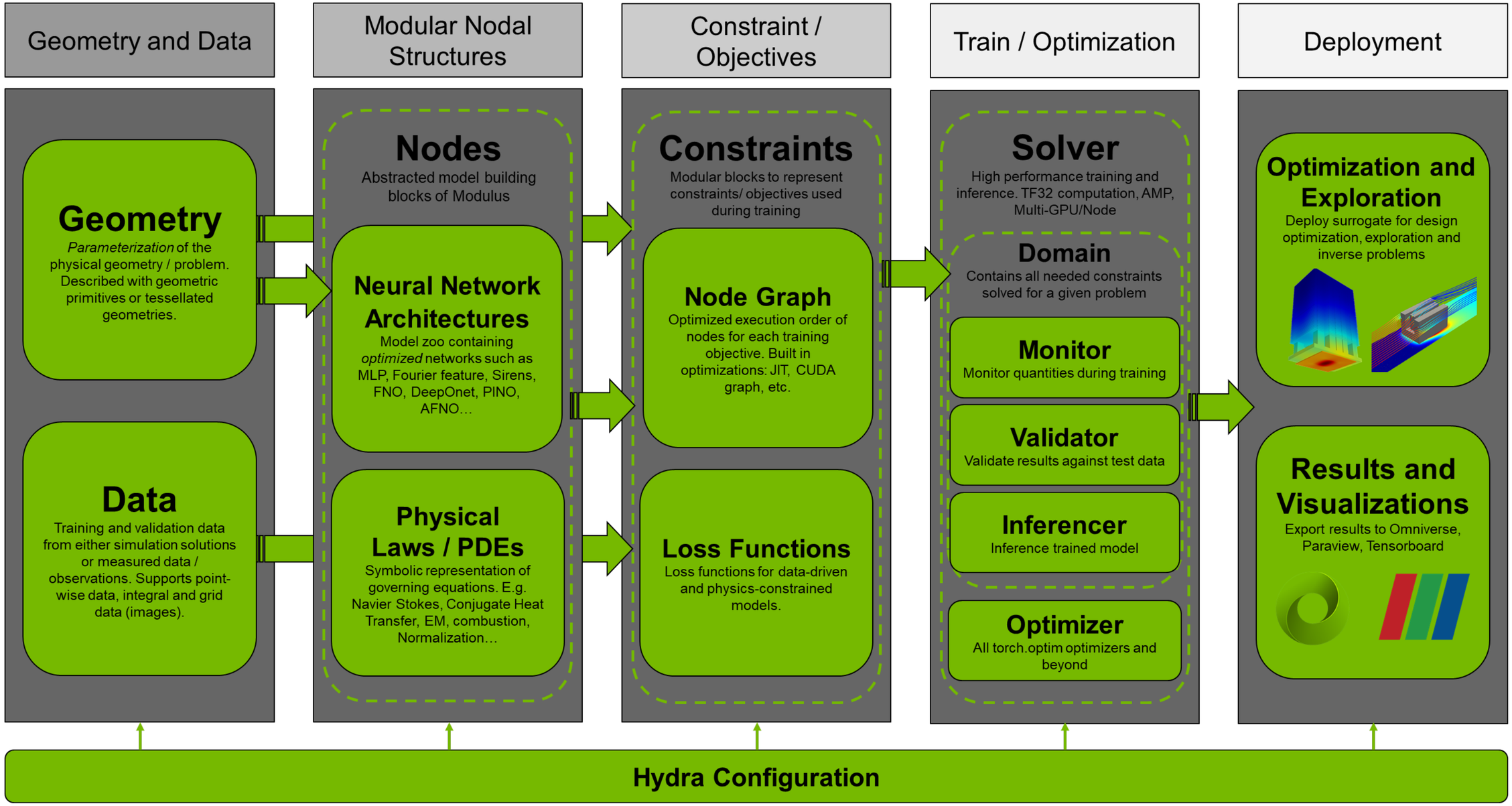
- Powered by the Fourier Neural Operator, this GPU-accelerated AI-enabled digital twin, called FourCastNet, is trained on 10 TB of Earth system data.
- Using this data, together with NVIDIA Modulus and Omniverse, we are able to forecast the precise path of catastrophic atmospheric rivers a full week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

Demo

NVIDIA.

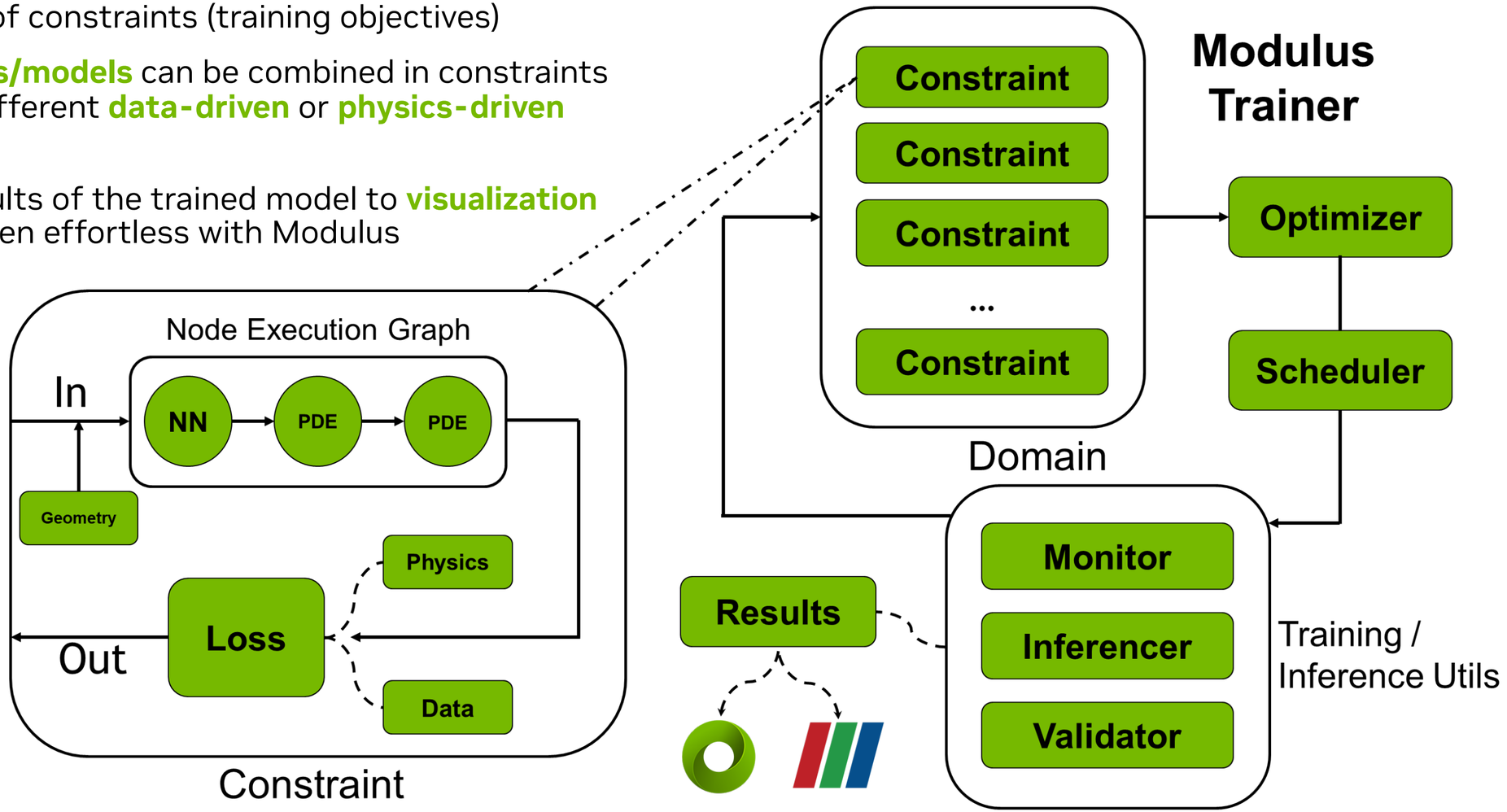

What’s inside Modulus?

Modulus Architecture



Modulus Training

- Modulus allows for **complex problems** to be described through sets of constraints (training objectives)
- Multiple **nodes/models** can be combined in constraints for learning different **data-driven** or **physics-driven** loss functions
- Exporting results of the trained model to **visualization software** is then effortless with Modulus



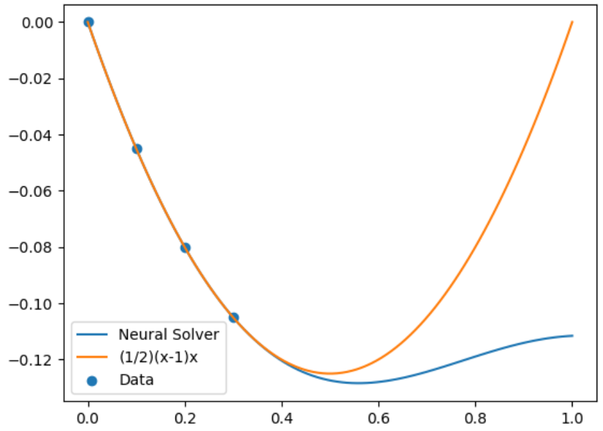
Physics Informed Neural Networks (PINNs) in Modulus

A problem with NNs and the promise of PINNs

Data Only

$$L_{data} = \sum_{x_i \in data} (u_{net}(x_i) - u_{true}(x_i))^2$$

$$L_{total} = L_{data}$$

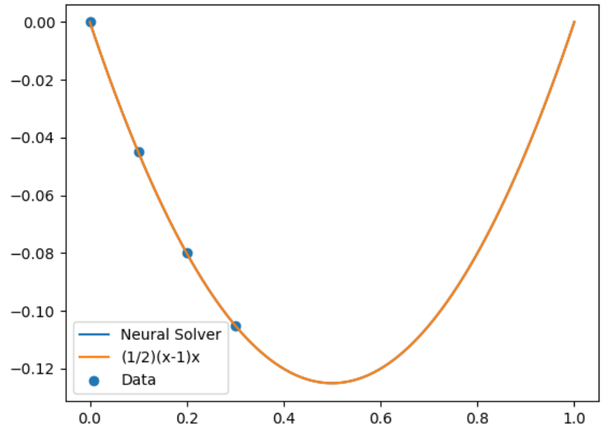


Data + Physics

$$\frac{\delta^2 u}{\delta x^2}(x) = 1$$

$$L_{physics} = \sum_{x_j \in domain} \left(\frac{\delta^2 u_{net}}{\delta x^2}(x_i) - f(x_i) \right)^2$$

$$L_{total} = L_{data} + L_{physics}$$

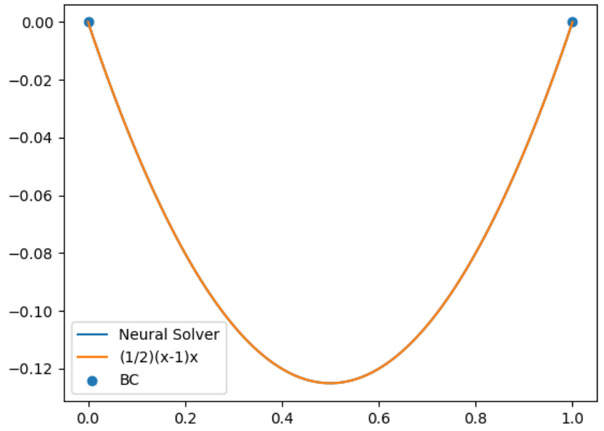


Physics only

$$\frac{\delta^2 u}{\delta x^2}(x) = 1 \qquad u(0) = u(1) = 0$$

$$L_{physics} = L_{residual} + L_{BC}$$

$$L_{total} = L_{physics}$$



PINNs Theory: Neural Network Solver

Problem description

- 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 $u_{net}(x)$ 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

PINNs Theory: Neural Network Solver

Loss formulation

- 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

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

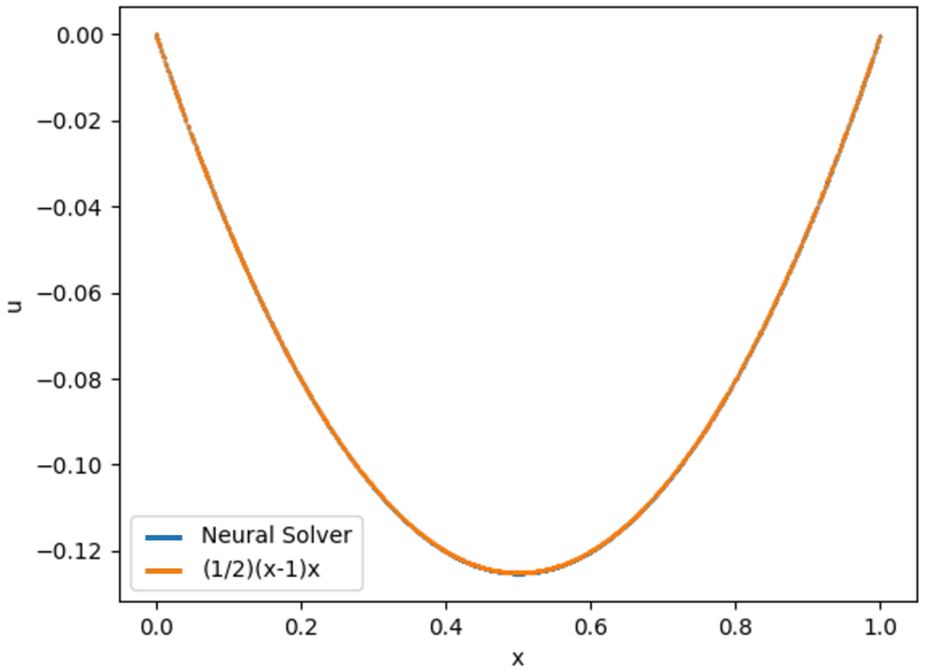
$$L_{Residual} = \int_0^1 \left(\frac{\delta^2 u_{net}}{\delta x^2}(x) - f(x)\right)^2 dx \approx \left(\int_0^1 dx\right)\frac{1}{N}\sum_{i=0}^{N}\left(\frac{\delta^2 u_{net}}{\delta x^2}(x_i) - f(x_i)\right)^2$$

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

PINNs Theory: Neural Network Solver

Results

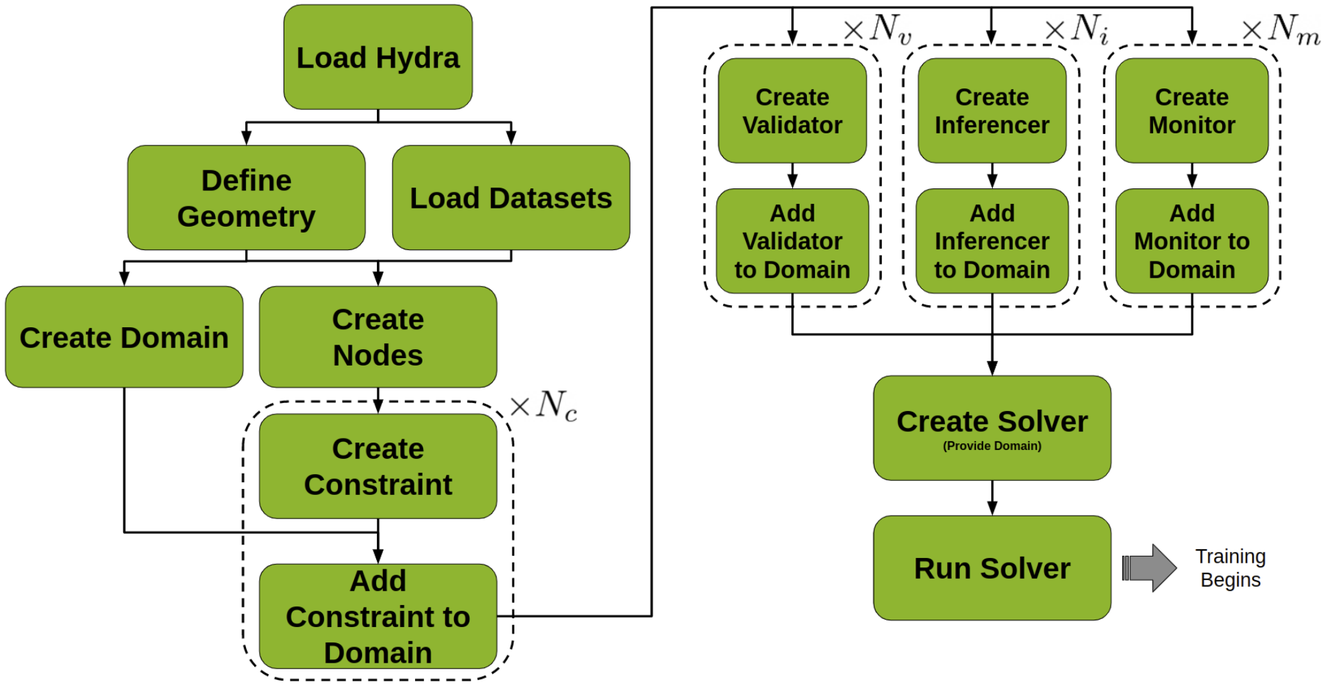
- For $f(x) = 1$, the true solution is $\frac{1}{2}(x-1)x$. After sufficient training we have,



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

Modulus: Anatomy of a project

Overview

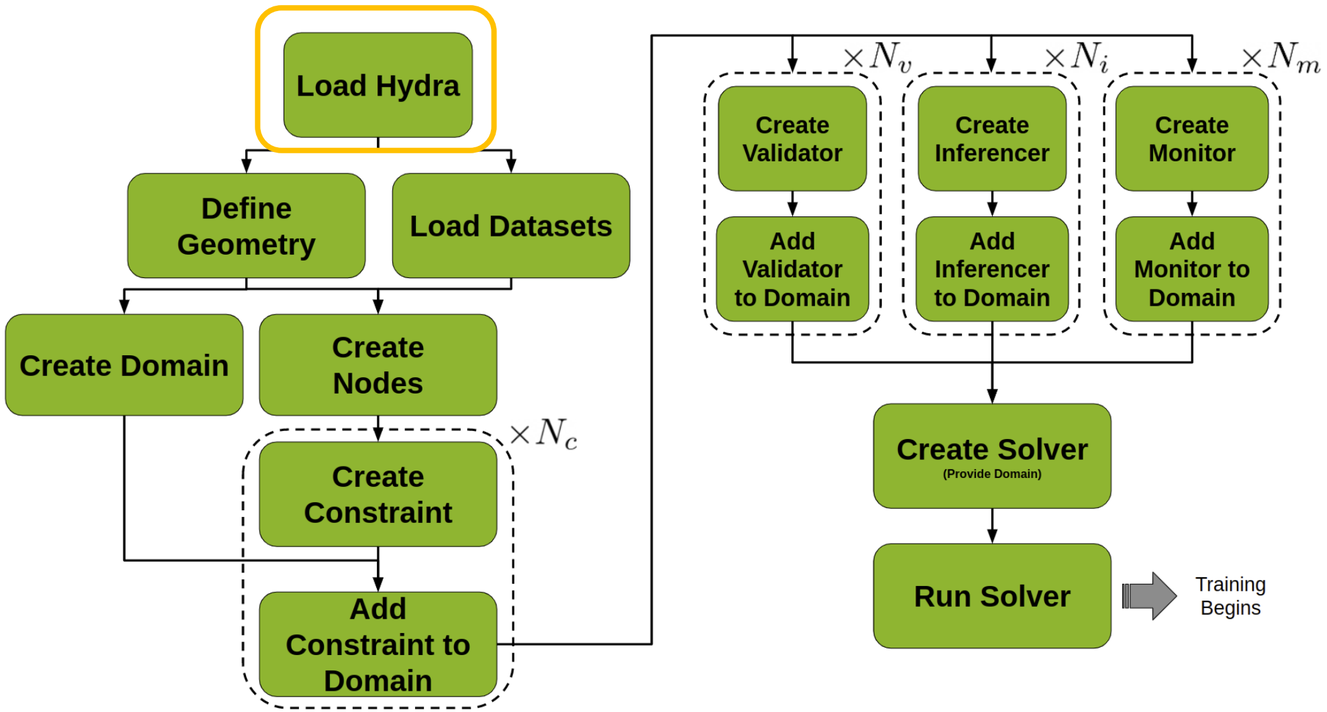


Modulus works by:

- Writing models which include at least one adaptable function (a NN)
- Writing objective functions as a combination of these models
- Describing the geometry/dataset where the models should be evaluated
- Minimizing the objective functions by using the provided data, by sampling the geometry, or both
- Running the models to obtain the desired effect

Modulus: Anatomy of a project

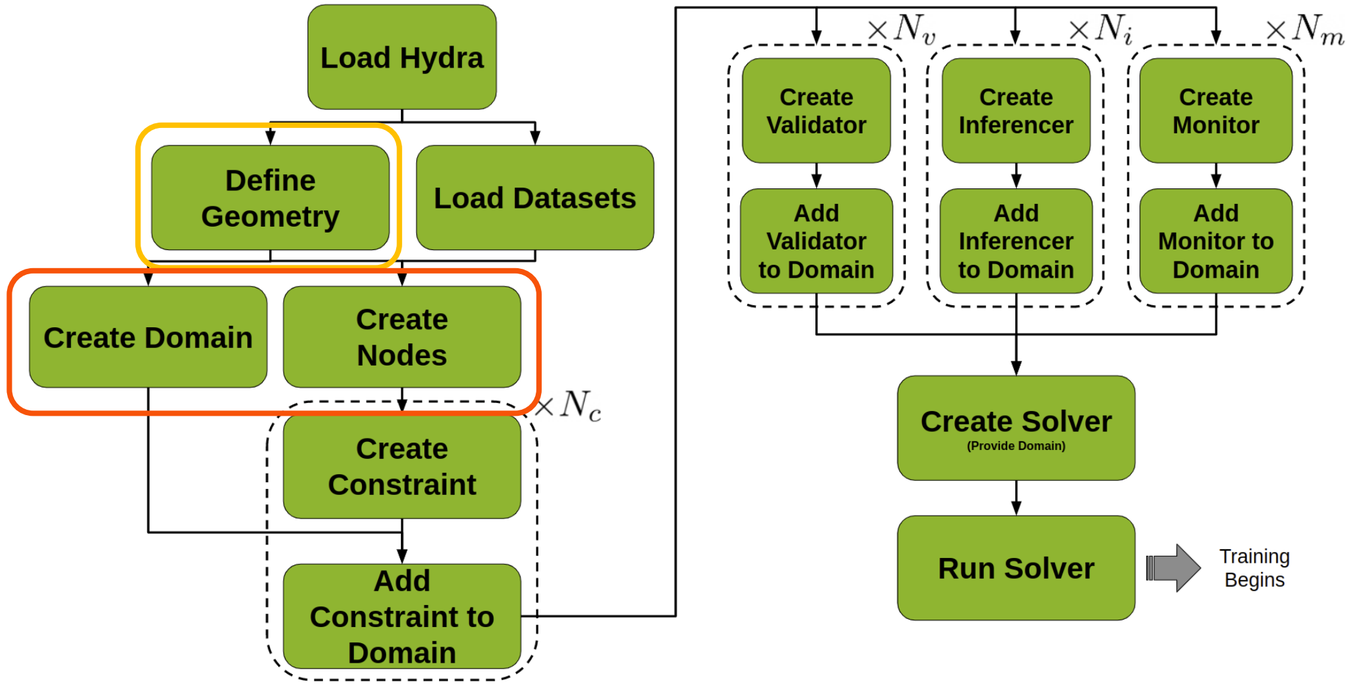
Load Hydra




```yaml
defaults :
  - modulus_default
  - scheduler: tf_exponential_lr
  - optimizer: adam
  - loss: sum
  - _self_

scheduler:
  decay_rate: 0.95
  decay_steps: 200

save_filetypes : "vtk,npz"

training:
  rec_results_freq : 1000
  rec_constraint_freq: 1000
  max_steps : 5000
```

Modulus: Anatomy of a project

Create geometry, domain and nodes

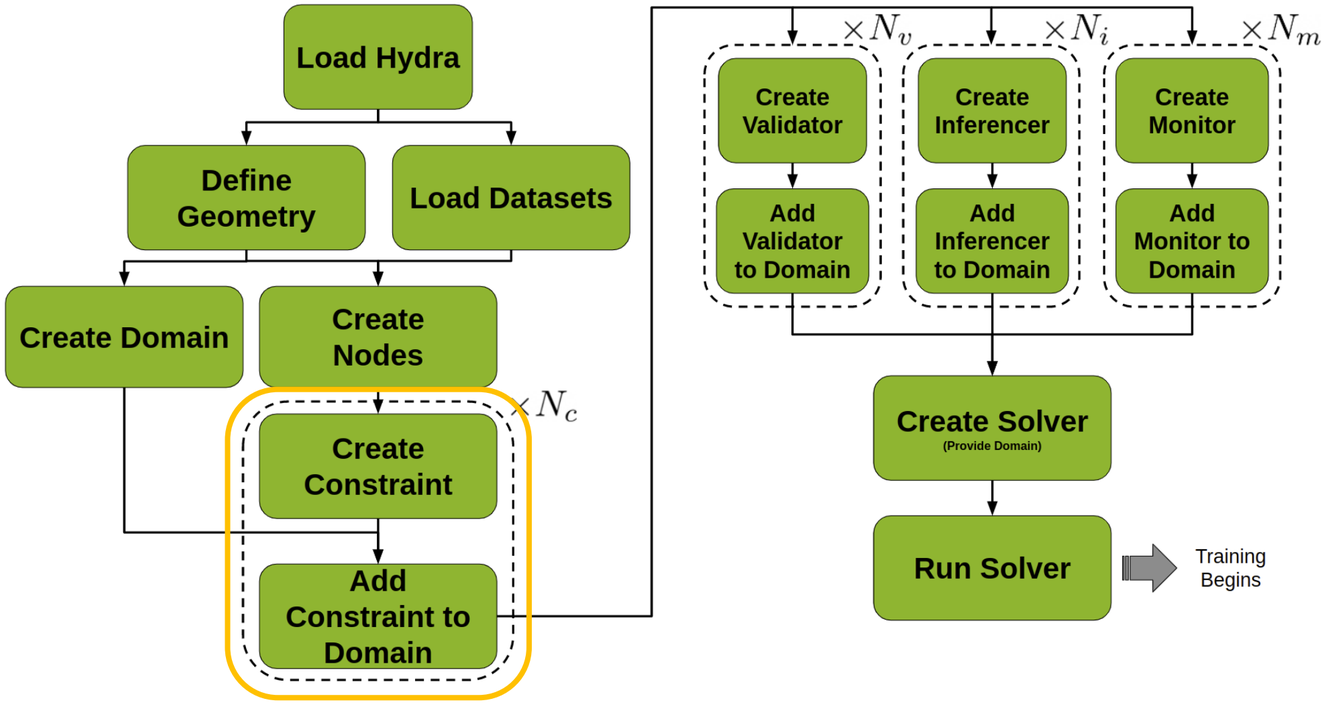



```python
@modulus.main(config_path="conf", config_name="config")
def run(cfg: ModulusConfig) -> None:

    # make geometry
    x = Symbol("x")
    geo = Line1D(0, 1)
```

```python
    # make list of nodes to unroll graph on
    eq = CustomPDE(f=1.0)
    u_net = FullyConnectedArch(
        input_keys=[Key("x")],
        output_keys=[Key("u")],
        nr_layers=3,
        layer_size=32
    )

    nodes = eq.make_nodes() + [u_net.make_node(name="u_network")]

    # make domain
    domain = Domain()
```

Modulus: Anatomy of a project

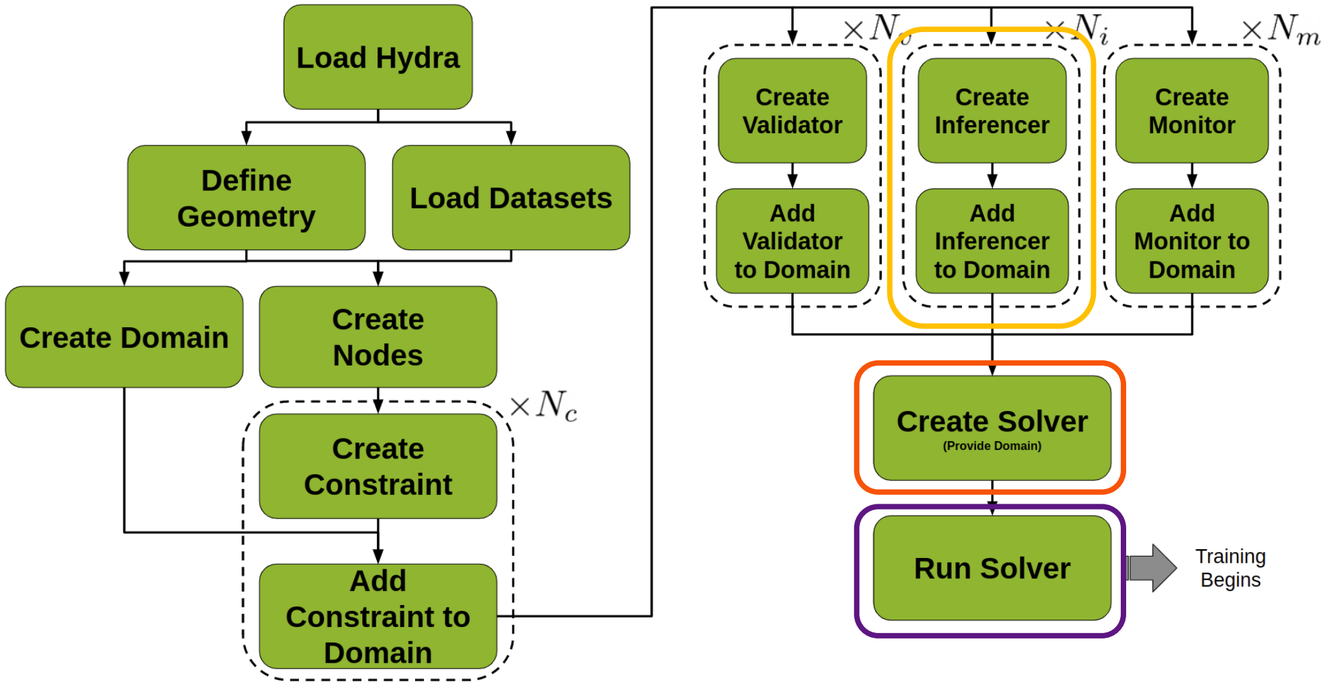
Add constraints




```python
# add constraints to solver

# bcs
bc = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=geo,
    outvar={"u": 0},
    batch_size=2,
)
domain.add_constraint(bc, "bc")

# interior
interior = PointwiseInteriorConstraint(
    nodes=nodes,
    geometry=geo,
    outvar={"custom_pde": 0},
    batch_size=100,
    bounds={x: (0, 1)},
)
domain.add_constraint(interior, "interior")
```

Modulus: Anatomy of a project

Create utils to visualize the results and run the solver



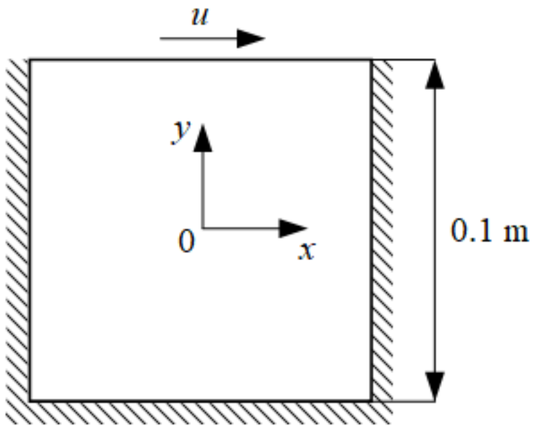

```
    # add inferencer
    inference = PointwiseInferencer(
        nodes=nodes,
        invar={"x": np.linspace(0, 1.0, 100).reshape(-1,1)},
        output_names=["u"],
    )
    domain.add_inferencer(inference, "inf_data")
```

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

    # start solver
    slv.solve()

if __name__ == "__main__":
    run()
```

```
python <script_name>.py
mpirun -np <#GPU> <script_name>.py
```

LID DRIVEN CAVITY BACKGROUND

- 2D flow for Lid Driven Cavity (LDC)




```yaml
defaults:
  - modulus_default
  - arch:
      - fully_connected
  - scheduler: tf_exponential_lr
  - optimizer: adam
  - loss: sum
  - _self_
scheduler:
  decay_rate: 0.95
  decay_steps: 4000

training:
  rec_validation_freq: 1000
  rec_inference_freq: 2000
  rec_monitor_freq: 1000
  rec_constraint_freq: 2000
  max_steps: 10000

batch_size:
  TopWall: 1000
  NoSlip: 1000
  Interior: 4000

graph:
  func_arch: true
```

```python
# make geometry
    height = 0.1
    width = 0.1
    x, y = Symbol("x"), Symbol("y")
    rec = Rectangle((-width / 2, -height / 2), (width / 2, height / 2))
```

```python
# make list of nodes to unroll graph on
    ns = NavierStokes(nu=0.01, rho=1.0, dim=2, time=False)
    flow_net = instantiate_arch(
        input_keys=[Key("x"), Key("y")],
        output_keys=[Key("u"), Key("v"), Key("p")],
        cfg=cfg.arch.fully_connected,
    )
    nodes = ns.make_nodes() + [flow_net.make_node(name="flow_network")]
```

```python
# add validator
    mapping = {"Points:0": "x", "Points:1": "y", "U:0": "u", "U:1": "v", "p": "p"}
    openfoam_var = csv_to_dict(
        to_absolute_path("openfoam/cavity_uniformVel0.csv"), mapping
    )
    openfoam_var["x"] += -width / 2  # center OpenFoam data
    openfoam_var["y"] += -height / 2  # center OpenFoam data
    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"]
    }
    openfoam_validator = PointwiseValidator(
        nodes=nodes,
        invar=openfoam_invar_numpy,
        true_outvar=openfoam_outvar_numpy,
        batch_size=1024,
        plotter=ValidatorPlotter(),
    )
    ldc_domain.add_validator(openfoam_validator)
```

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

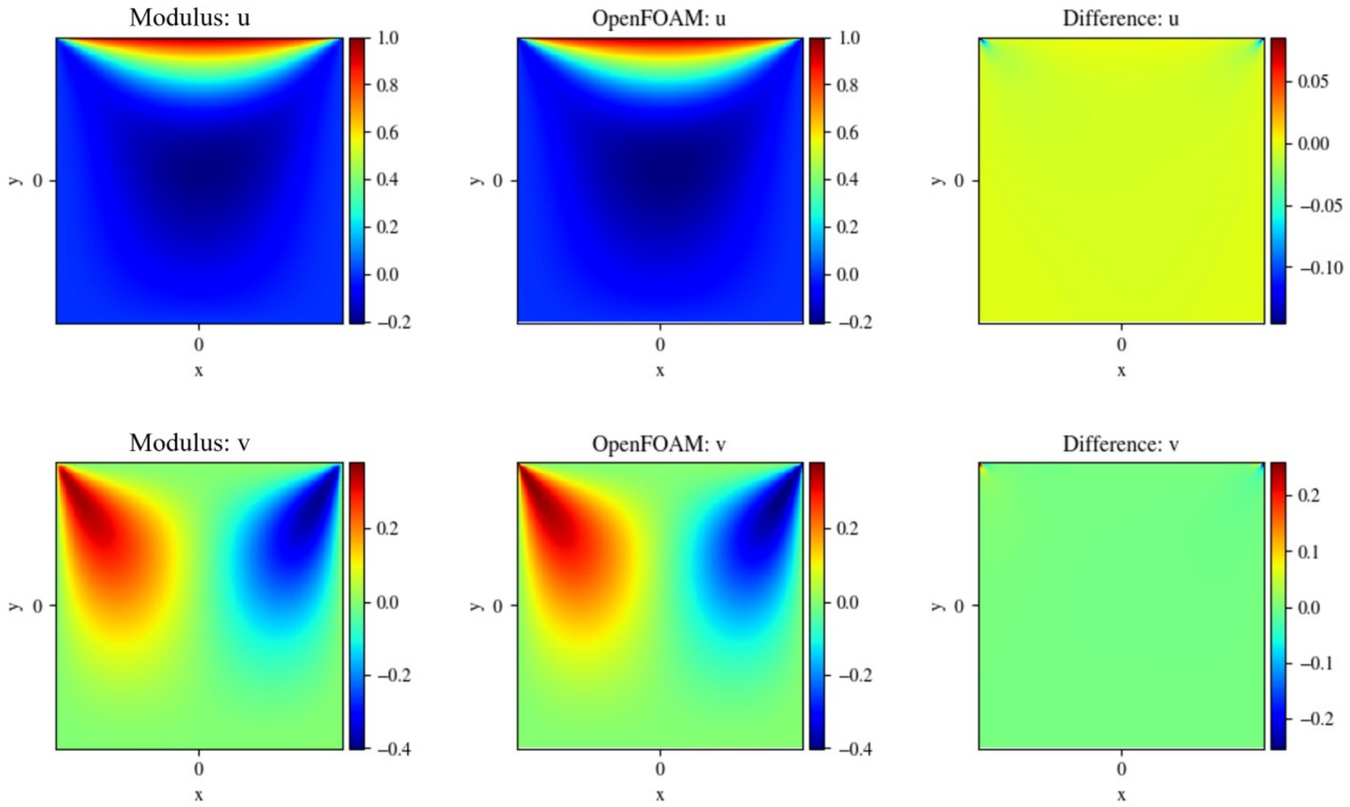
```python
# top wall
    top_wall = PointwiseBoundaryConstraint(
        nodes=nodes,
        geometry=rec,
        outvar={"u": 1.0, "v": 0},
        batch_size=cfg.batch_size.TopWall,
        lambda_weighting={"u": 1.0 - 20 * Abs(x), "v": 1.0},  # weight edges to be zero
        criteria=Eq(y, height / 2),
    )
    ldc_domain.add_constraint(top_wall, "top_wall")

    # no slip
    no_slip = PointwiseBoundaryConstraint(
        nodes=nodes,
        geometry=rec,
        outvar={"u": 0, "v": 0},
        batch_size=cfg.batch_size.NoSlip,
        criteria=y < height / 2,
    )
    ldc_domain.add_constraint(no_slip, "no_slip")

    # interior
    interior = PointwiseInteriorConstraint(
        nodes=nodes,
        geometry=rec,
        outvar={"continuity": 0, "momentum_x": 0, "momentum_y": 0},
        batch_size=cfg.batch_size.Interior,
        lambda_weighting={
            "continuity": Symbol("sdf"),
            "momentum_x": Symbol("sdf"),
            "momentum_y": Symbol("sdf"),
        },
    )
    ldc_domain.add_constraint(interior, "interior")
```

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

    # start solver
    slv.solve()
```

NVIDIA

MODULUS RESULTS ON LDC

Compare to OpenFOAM



WHAT IS MODULUS?

Putting it all together

- Modulus is a PDE solver (category I)
- Modulus is a tool for efficient design optimization & design space exploration (category I)
- Modulus is a solver for inverse problems (category II)
- Modulus is a tool for developing digital twins (category II)
- Modulus is a tool for developing AI solutions to engineering problems (category III)

These are all done by developing deep neural network models in Modulus that are physics-informed and/or data-informed.