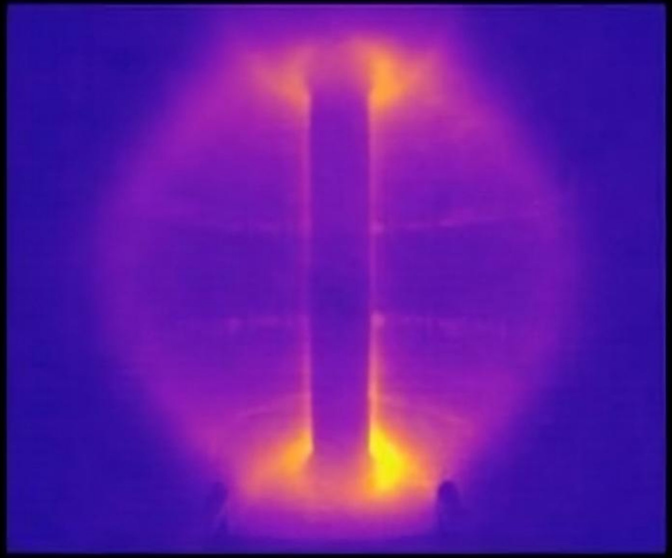
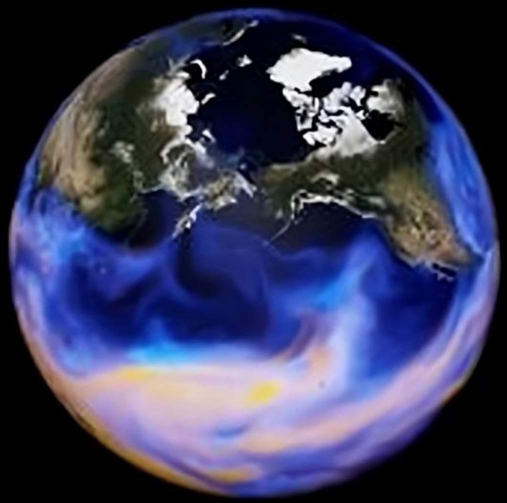
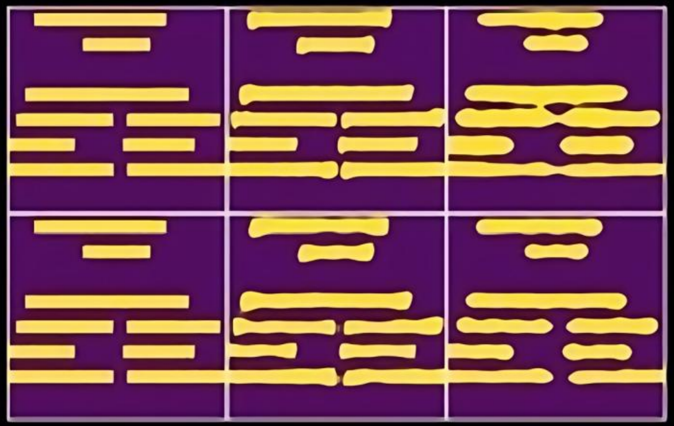
Nuclear Fusion

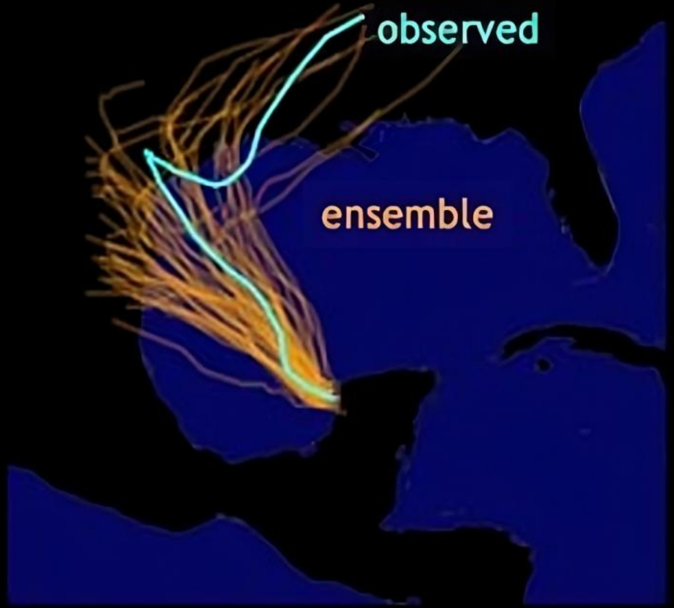


Weather and Climate



Chip Design/Lithography





Neural Operators: Machine Learning in Function Spaces

Robert Joseph George

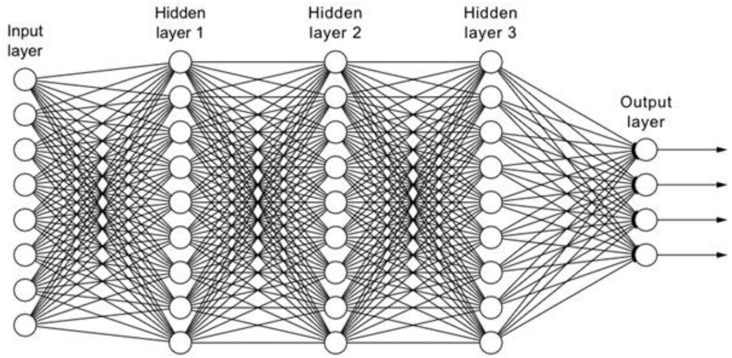
PhD Student Math & CS

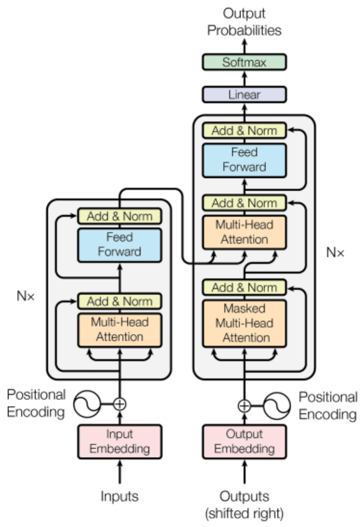
NASA JPL Caltech

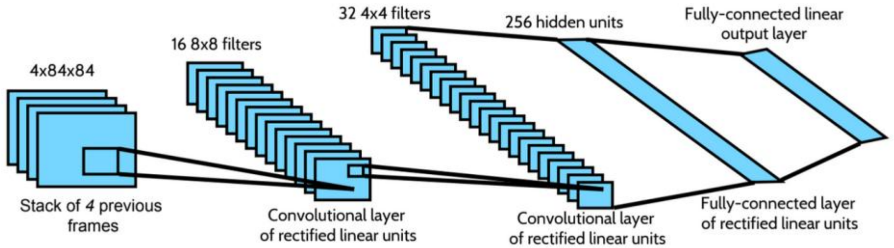
Rise of Deep Learning

- Neural Networks such as CNN, AlexNet, LSTM, ResNet, UNet, EfficientNet, MobileNet, Transformer, ViT, Diffusion Models.

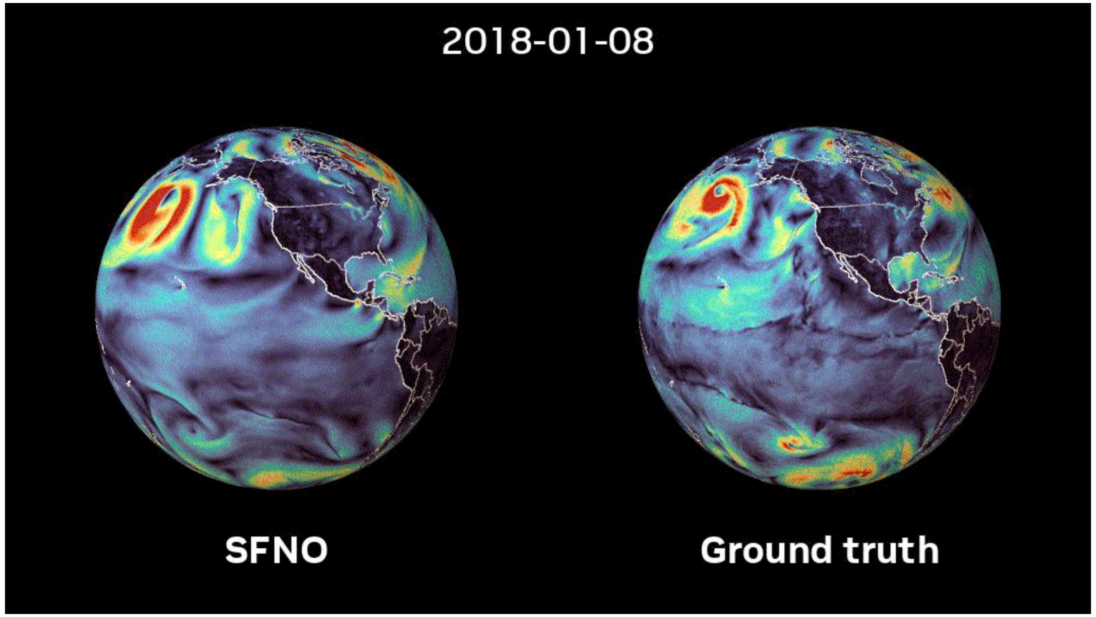
Compute + Domain Specific metrics → Progress in CV + Language



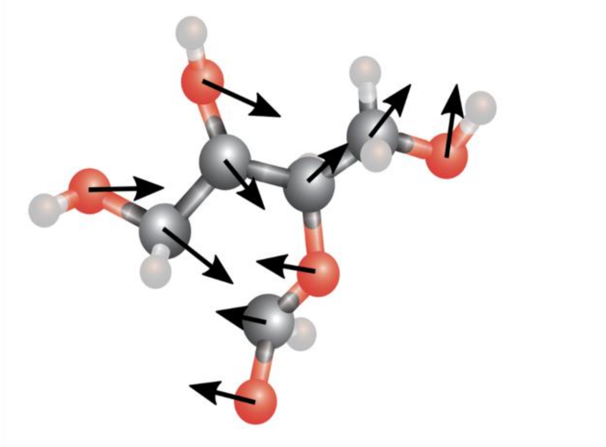




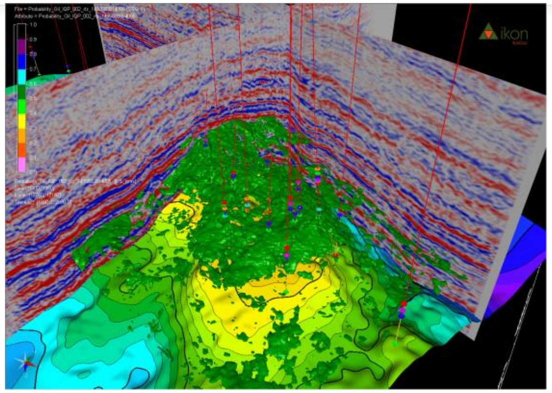
Real world domain (Function) -> Data (Function)



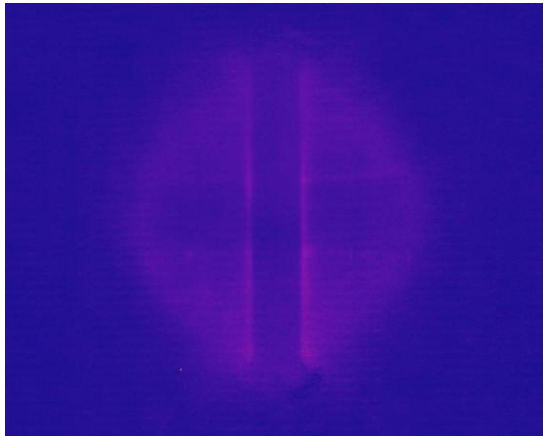
Weather forecasting



Protein Engineering



Los Angeles Basin



Fusion

However, we should view them as functions and not just time-series or pictures.

Numerical Solvers

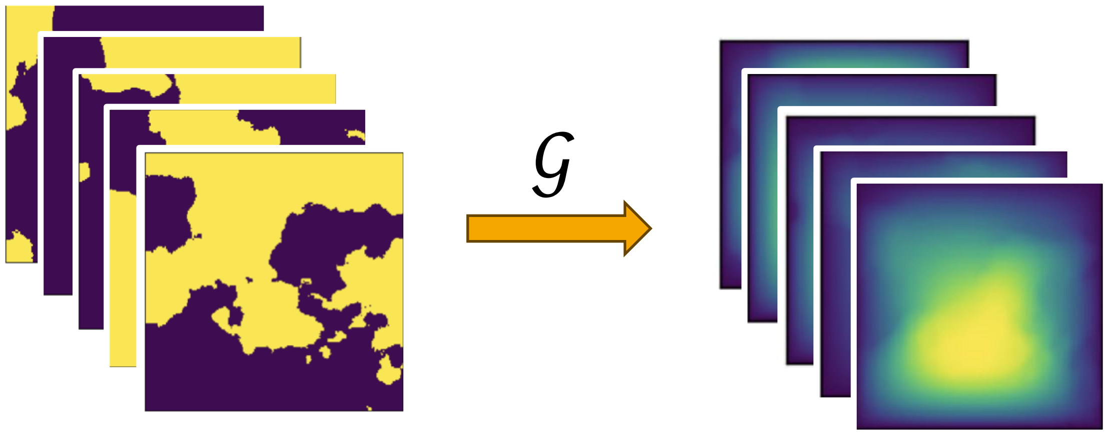
- Traditionally we model these phenomena using differential and algebraic equations. Examples include Darcy, Navier-Stokes, Helmotz etc.)
- Create numerical solvers to solve these equations at a certain discretization (resolution). For example: Finite difference, elements etc.

Finer discretization ➡ Converge to ground truth operator (more accurate solution)

$$\begin{aligned} \partial_t u(x,t) + u(x,t)\cdot\nabla u(x,t) + \nabla p(x,t) &= \nu\Delta u(x,t) + f(x), \\ \nabla\cdot u(x,t) &= 0, \\ u(x,0) &= u_0(x), \end{aligned}$$

$$i\hbar\frac{\partial}{\partial t}\Psi(x,t) = \left[-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x,t)\right]\Psi(x,t)$$

Darcy Flow: $-\nabla.\big(a(x)\nabla u(x)\big) = f(x)$



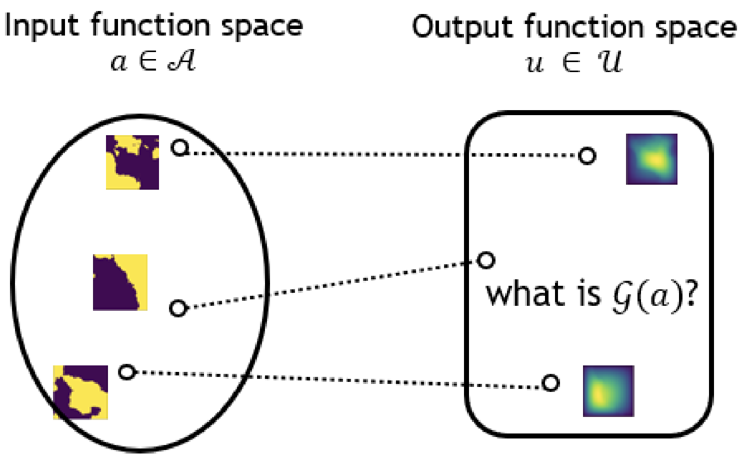
Input: diffusion coefficients, $a's$ **Output**: solutions, $u's$

Limitations

- Generating good data is hard.
- Solvers are not differentiable; Not good for inverse problems
- Hard to incorporate domain knowledge into the solver
- Massive computation
- Discretization dependent

How about we learn the solution operator?

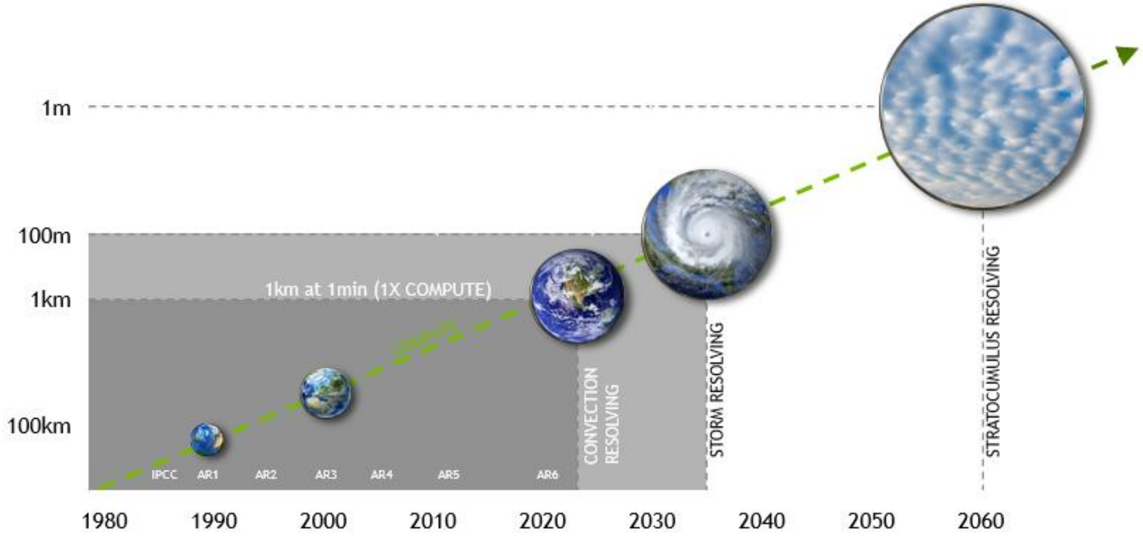
Given a, predict u



Infinite dimension

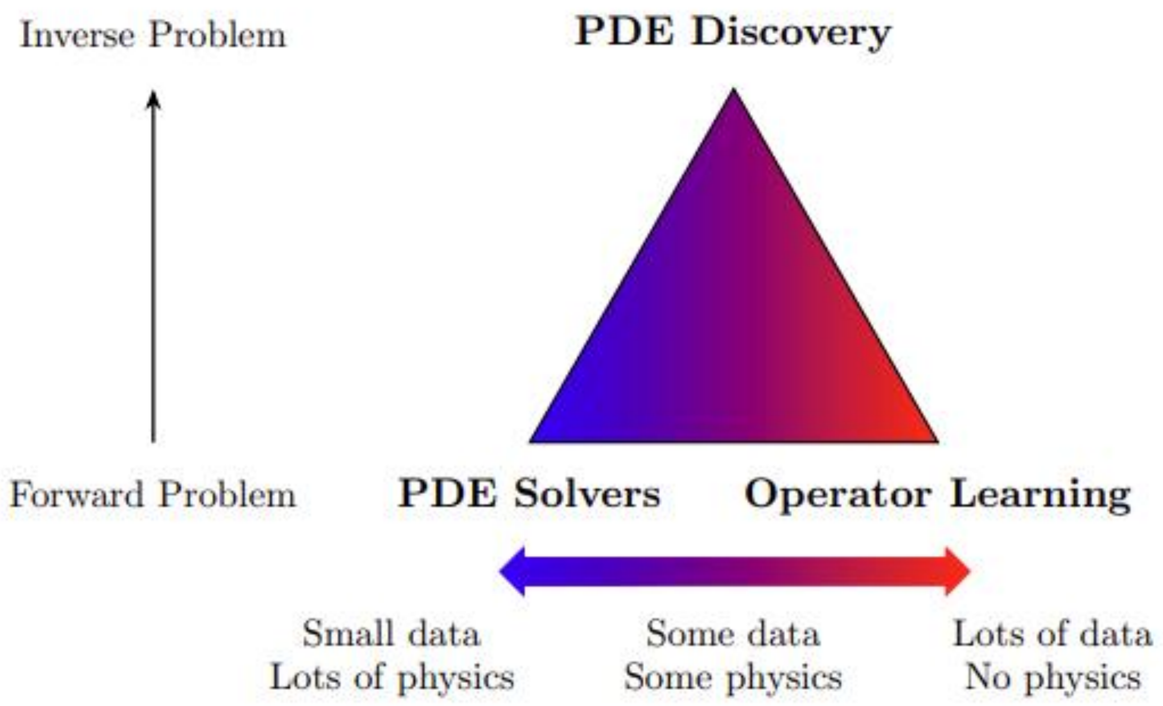
Computational constraints limit model resolution

Reasonable solution operator requires high resolution → much more computes



Moving on to learning Functions

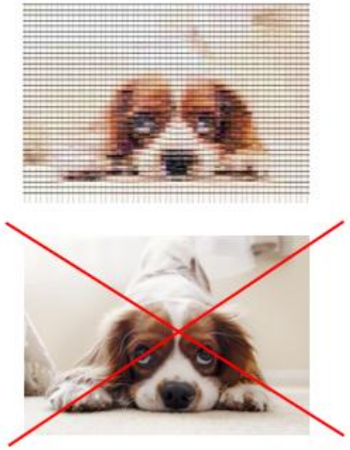
- The classical development of neural networks has been primarily for mappings between a finite-dimensional Euclidean space.
- However, many problems in physics and math involve learning the mapping between ***function spaces***, which poses a limitation on the classical neural network-based methods.
- For a bold example, images should be considered as functions of light defined on a continuous region instead of as 32 x 32-pixel vectors.

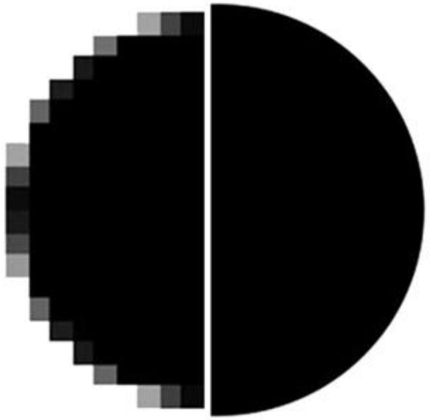


One ML model for any discretization

Neural Network

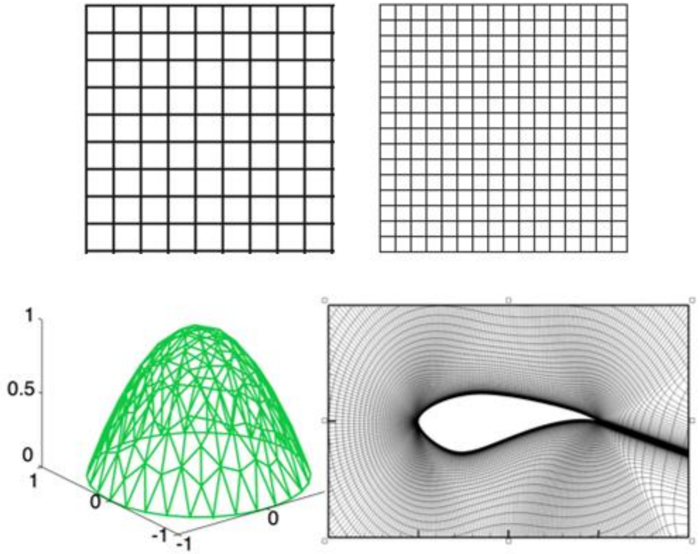
Input and output at fixed resolution





Neural Operator

Input and output at any resolution



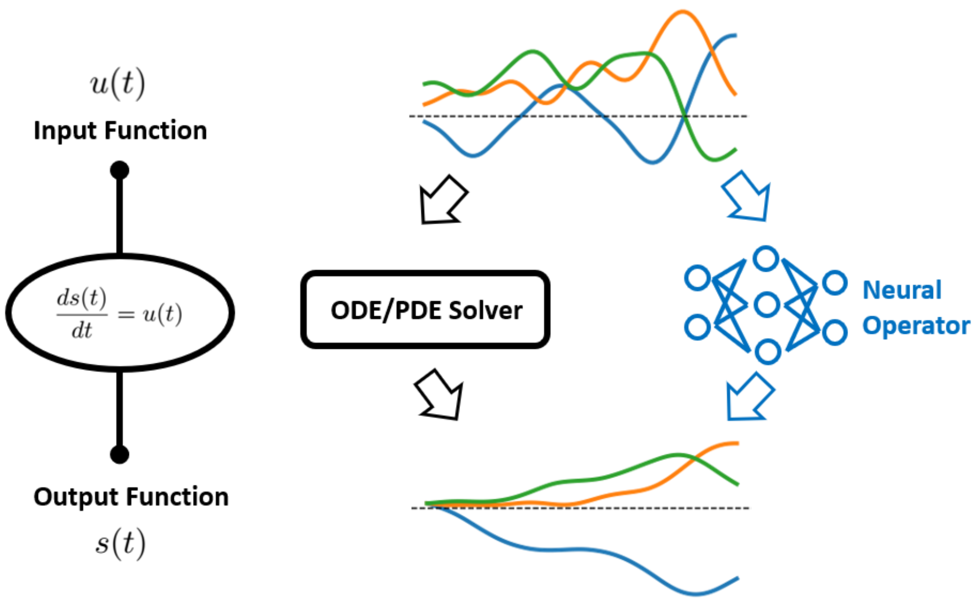
Discretization Agnostic Learning

Neural Operator (What are they?)

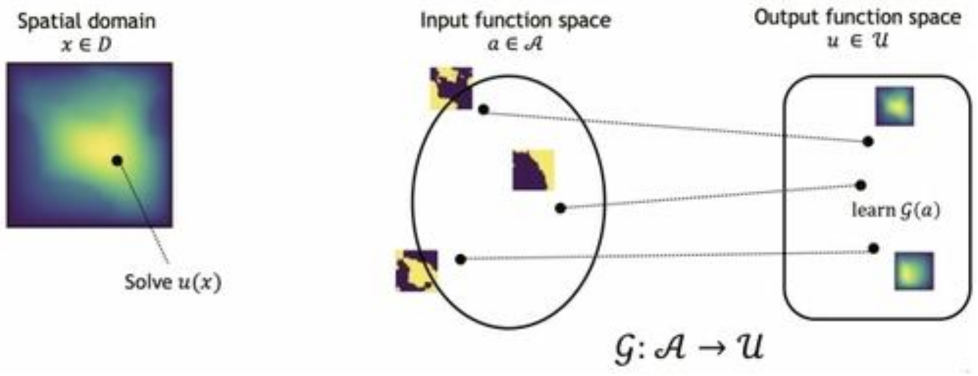
In mathematics, operators are usually referring to the mappings between function spaces. Consider a general differential equation represented as

$$Lu = f$$

where u and f are functions defined on the physical domain. Effectively, the task is akin to learning an operator, often seen as the inverse of L, capable of mapping the given function f back to the desired function u.



To deal with this problem, we propose operator learning. By encoding specific structures, we let the neural network learn the mapping of functions and generalize among different resolutions. As a result, we can first use a numerical method to generate some less-accurate, low-resolution data, but the learned solver can still give reasonable, high-resolution predictions. In some sense, both training and evaluation can be pain-free.

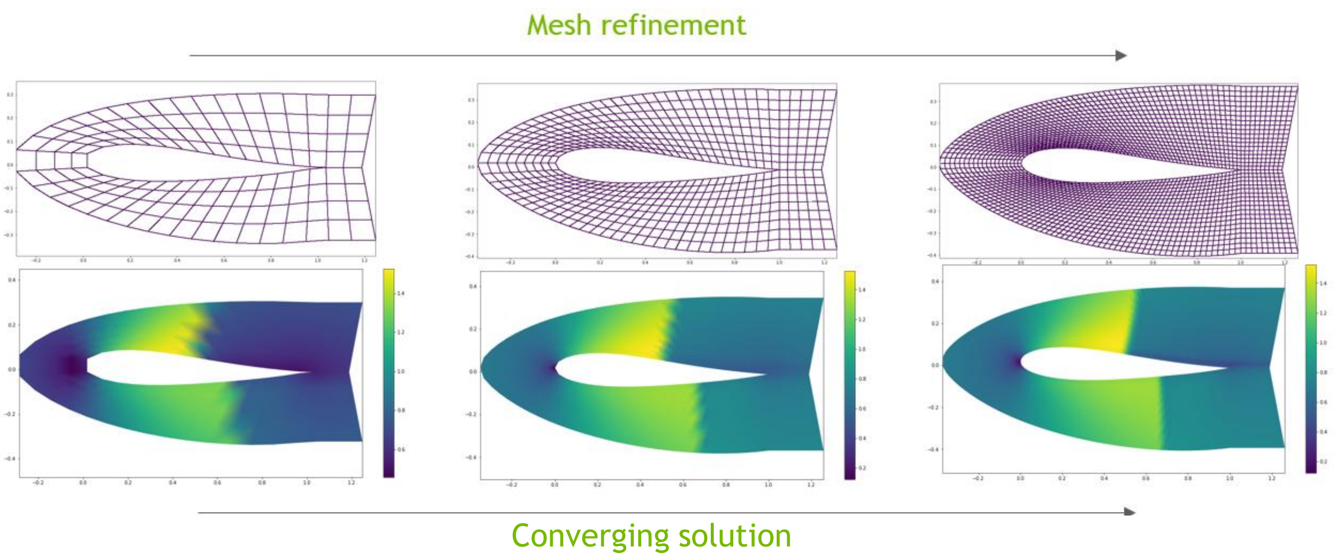


Discretization -Convergent

One ML model for any discretization

Definition: a trained AI model is discretization-convergent if

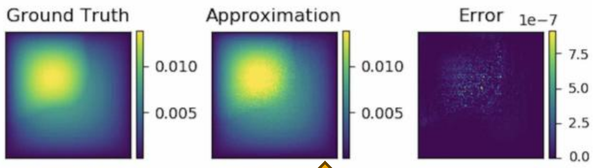
- We can query at any point.
- Converges upon mesh refinement to a limit.



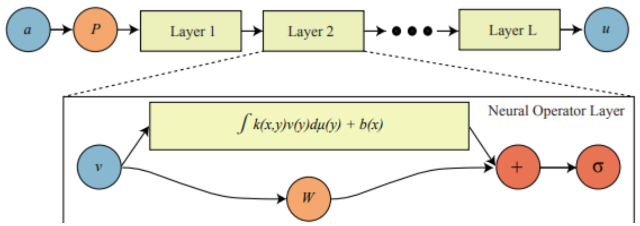
Property \ Model	NNs	DeepONets	Interpolation	Neural Operators
Discretization Invariance	✗	✗	✓	✓
Is the output a function?	✗	✓	✓	✓
Can query the output at any point?	✗	✓	✓	✓
Can take the input at any point?	✗	✗	✓	✓
Universal Approximation	✗	✓	✗	✓

Math of Neural Operators

Solution



Architecture



Darcy Flow PDE

$$-\nabla \cdot (a(x)\nabla u(x)) = f(x), \quad x \in D$$

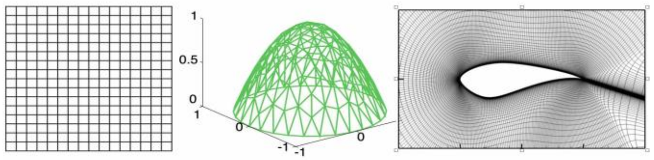
$$u(x) = 0, \quad x \in \partial D$$

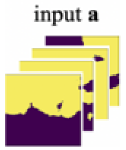
Given $f(x)$ and $a(x)$ find $u(x)$

Implementation

Discretization of f and a

Cause computers take discrete inputs





Example

Overall Algorithm

$$v_0(x) = NN_1(x, a(x))$$

$$v_{t+1}(x) = \sigma\left(Wv_t(x) + \int_{B(x,r)} \kappa_\phi(x, y, a(x), a(y))v_t(y)\,dy\right) \quad \text{for } t = 1, \ldots, T$$

$$u(x) = NN_2(v_T(x))$$

Reformulation of the PDE

Cause why not? Here basically $L_a u$ is the differential operator which depends on a and denotes the LHS like in our example.

$$(\mathcal{L}_a u)(x) = f(x), \quad x \in D$$

$$u(x) = 0, \quad x \in \partial D$$

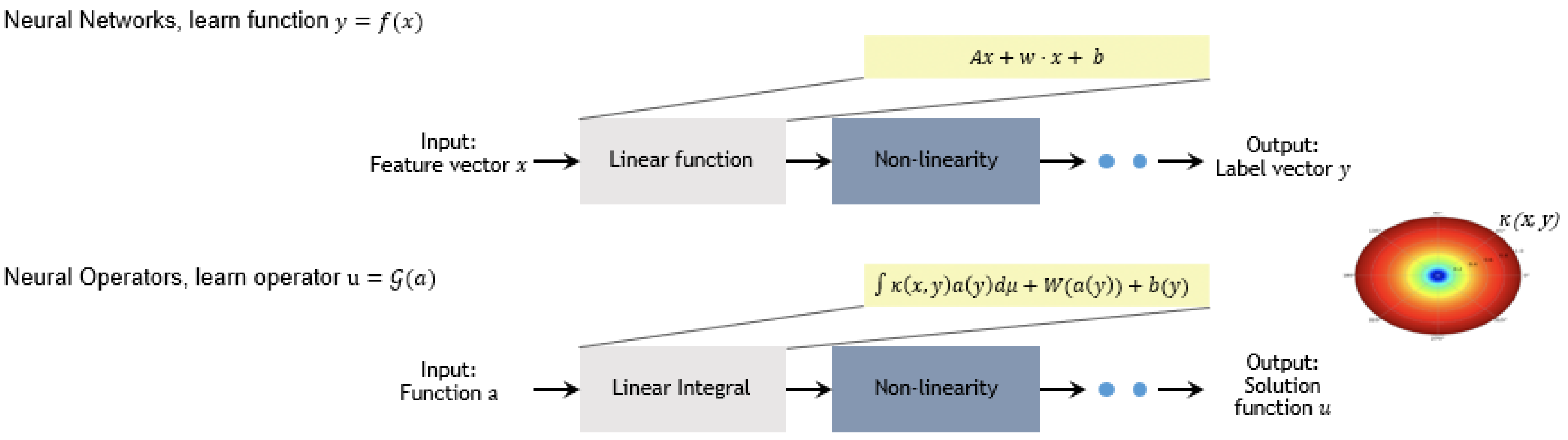
Green's Function

Surprisingly in mathematics there is this notion of a green's function $G_a(x,\cdot)$ which can give us a unique solution to the problem! (For more details check the paper). **Very cool right?**

$$u(x) = \int_D G_a(x, y)f(y)\,dy$$

Formula the green's function as a kernel via a neural network κ which depends on a.

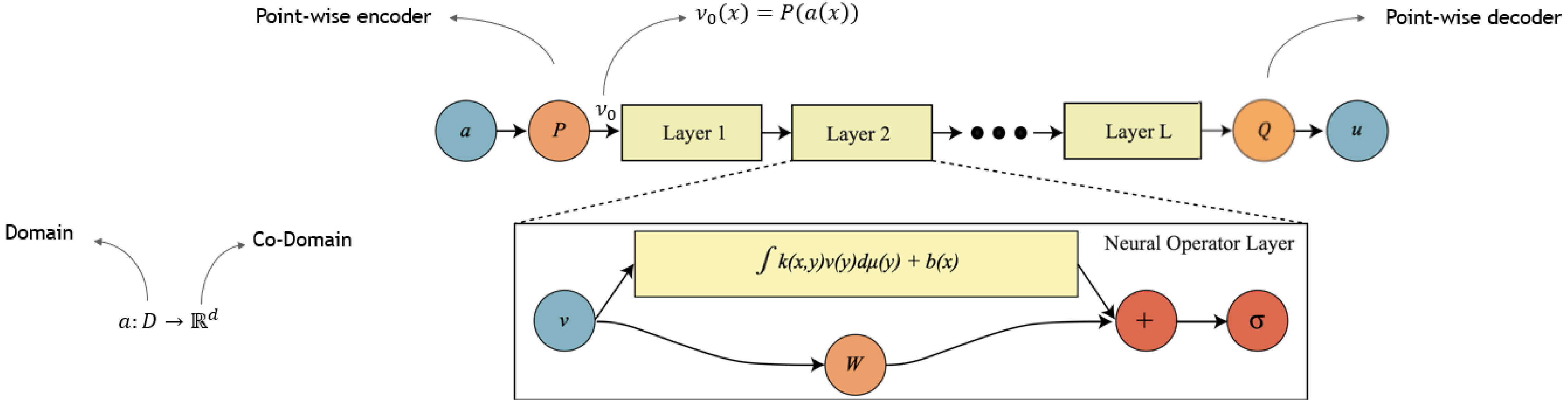
$$u(x) = \int_D \kappa(x, y, a(x), a(y))f(y)\,dy$$



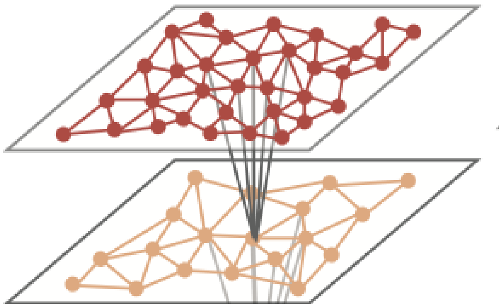
From Neural networks to Neural Operators

- Integral operator outputs functions (not just finite-dimensional vectors).
- Integral operator is discretization agnostic and discretization convergent.
- Neural Operators are universal approximator of operators.

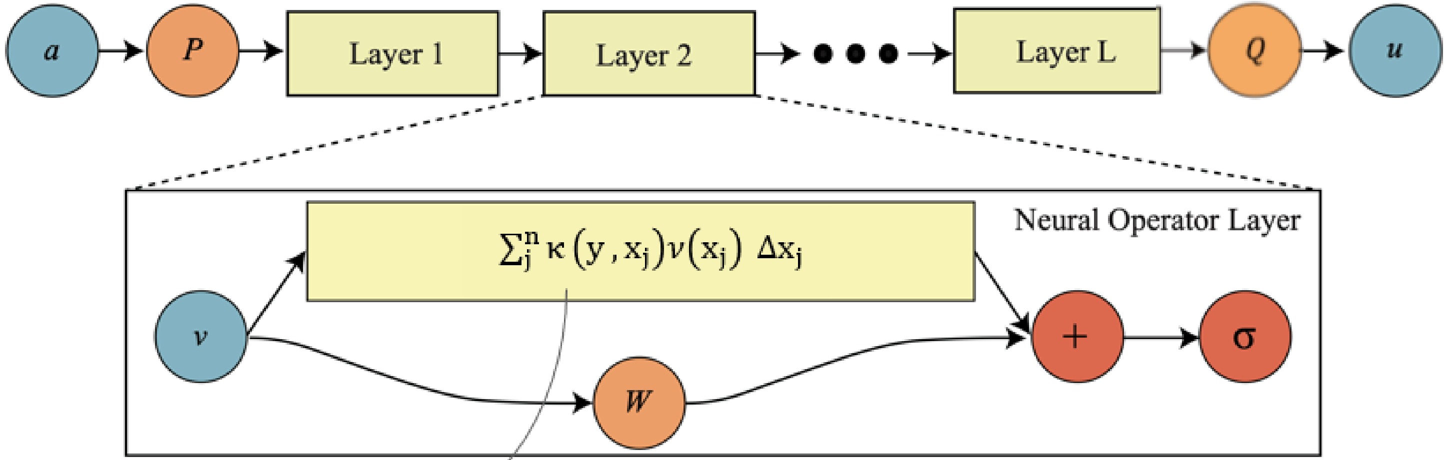
Architectures



Graph Neural Operator (GNO)

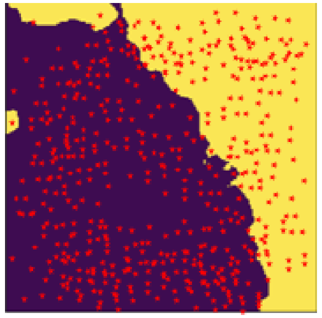


Suppose we parameterize the kernel as a Neural Network.



Learnable neural network

Local or Global



Generalization of GNNs to neural operators

Fourier Neural Operator (FNO)

Again note that map $K: v_t \to v_{t+1}$ is parameterized as

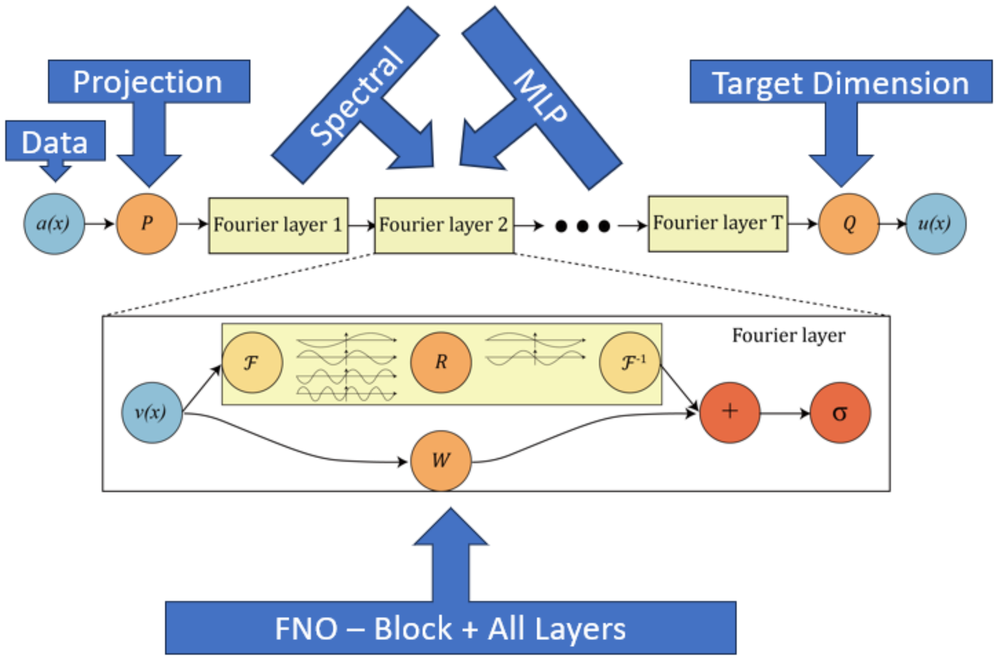
$$v'(x) = \int k(x, y) v(y) dy + W v(x)$$

Where k is a kernel function and W is the bias term. Now if we *restrict* $k(x, y) = k(x - y)$ then we get that our integral is indeed a convolution operator, which is a natural choice from the perspective of fundamental solutions. We can exploit the Fast Fourier Transform to do a convolution in Fourier space in quasilinear time.

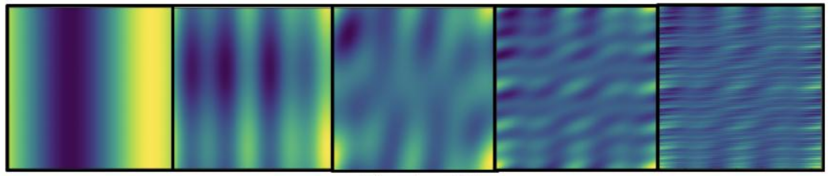
The Fourier layer consists of three steps:

1. Fourier transform F
2. Linear transform on the lower Fourier modes $R - K_{max}$
3. Inverse Fourier transform F^{-1}

Note: *When the input function is given on a regular grid*

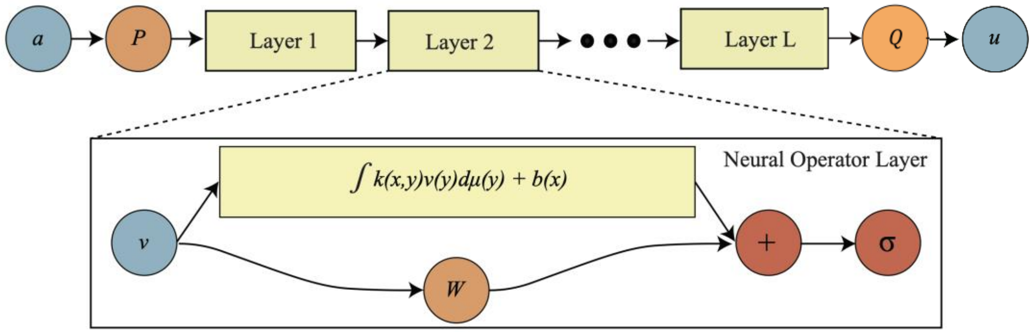



Filters in CNN

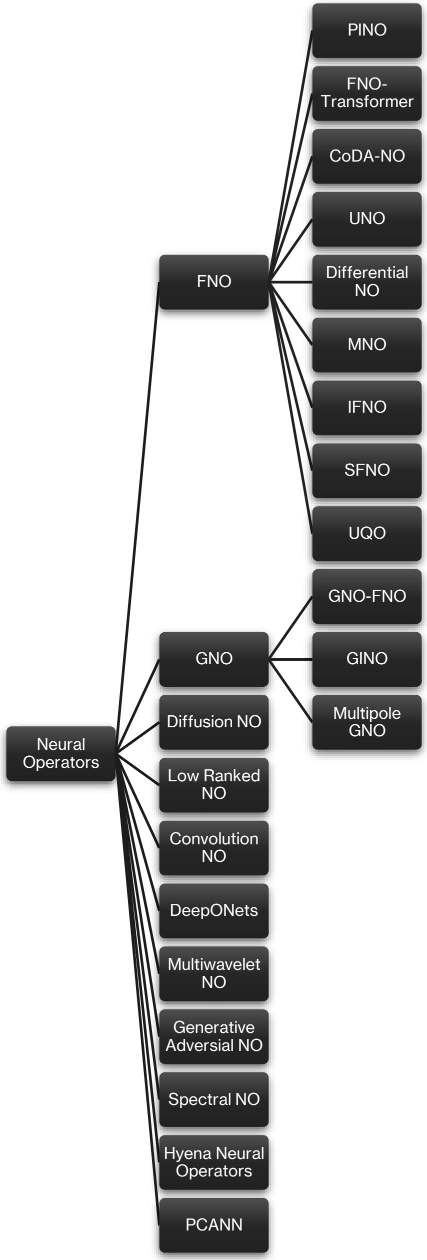

Fourier Filters

Filters in convolution neural networks are usually local. They are good to capture local patterns such as edges and shapes. Fourier filters are global sinusoidal functions. They are better for representing continuous functions.

Other Architectures

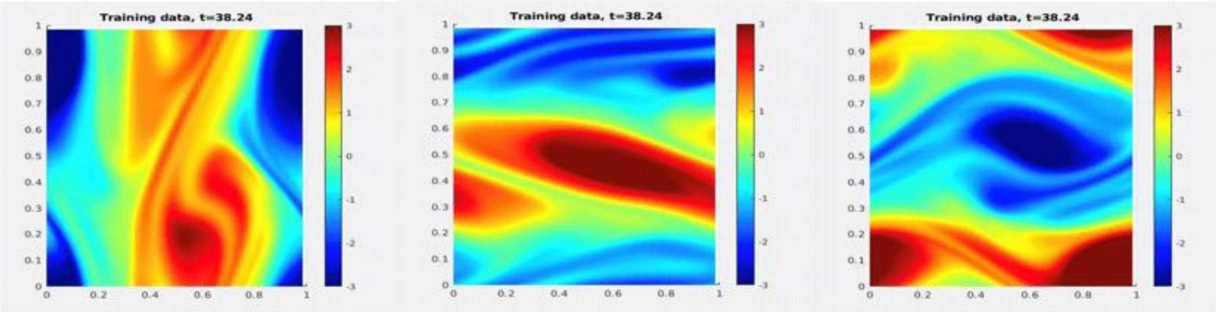


Lots of variants of Neural Operators depending on how you parameterize the kernel.

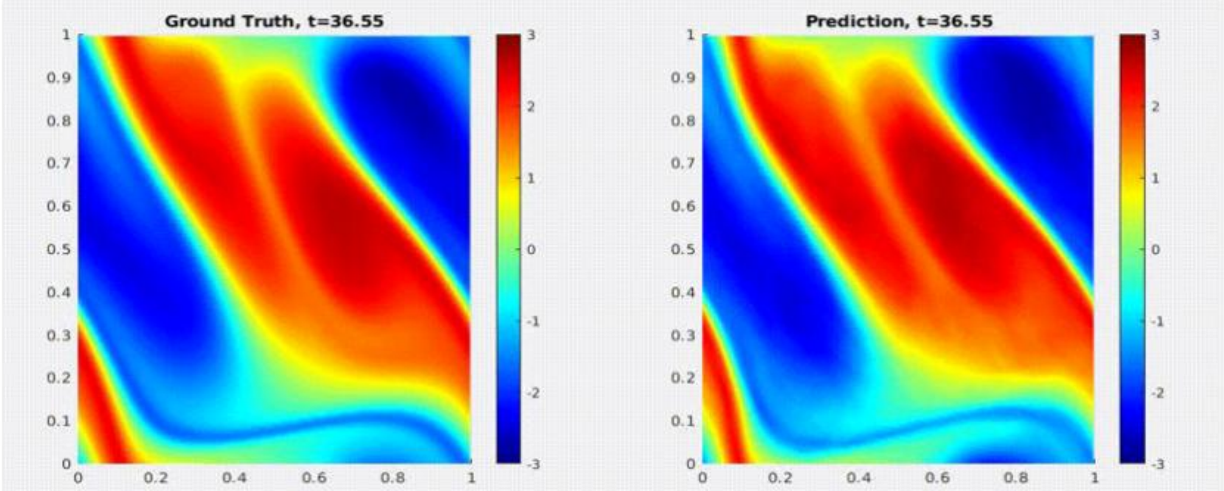


Zero shot super resolution

Train using coarse resolution data



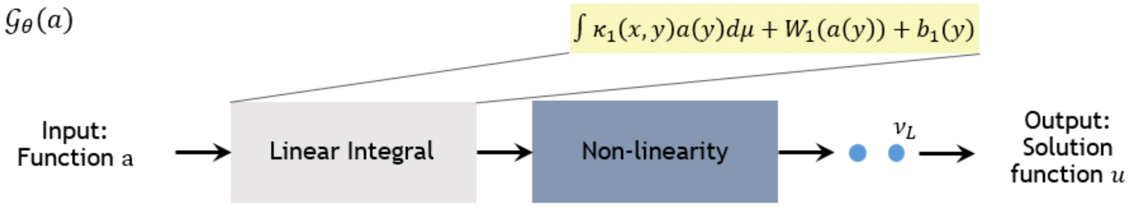
Directly evaluate on higher resolution (no re-training)



Note: The data contains the effect of high-resolution components of the physics because the data is assumed from real world (very high resolution solver).

Universal Approximators of Operators

Neural Operators, learn operator $u = \mathcal{G}_\theta(a)$



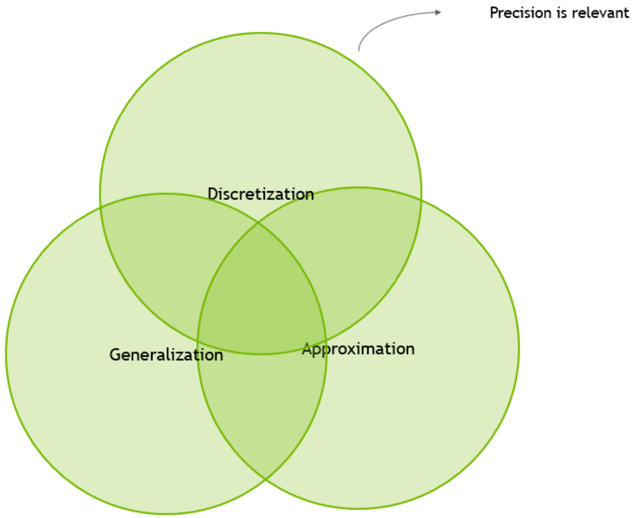
- Neural Operators are universal approximator of operators.

$$\|\hat{\mathcal{G}}_\theta(D_L, a|_{D_L}) - \mathcal{G}^\dagger(a)\|_{\mathcal{U}} \le \underbrace{\|\hat{\mathcal{G}}_\theta(D_L, a|_{D_L}) - \mathcal{G}_\theta(a)\|_{\mathcal{U}}}_{\text{discretization error}} + \underbrace{\|\mathcal{G}_\theta(a) - \mathcal{G}^\dagger(a)\|_{\mathcal{U}}}_{\text{approximation error}}.$$

Theorem (Universal approximation theorem of neural operators) :

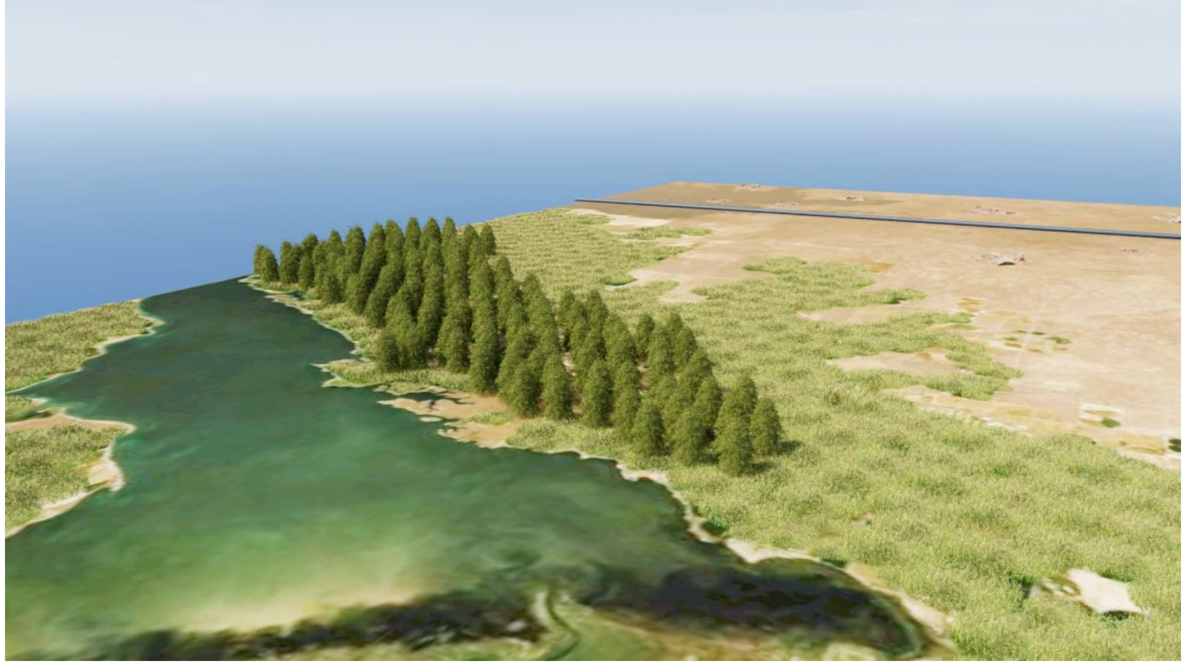
Under a mild regularity condition, for any given arbitrary operator between general function spaces $\mathcal{G}^\dagger$, and any $\epsilon > 0$, there exist a neural operator $\mathcal{G}_\theta$, such that,

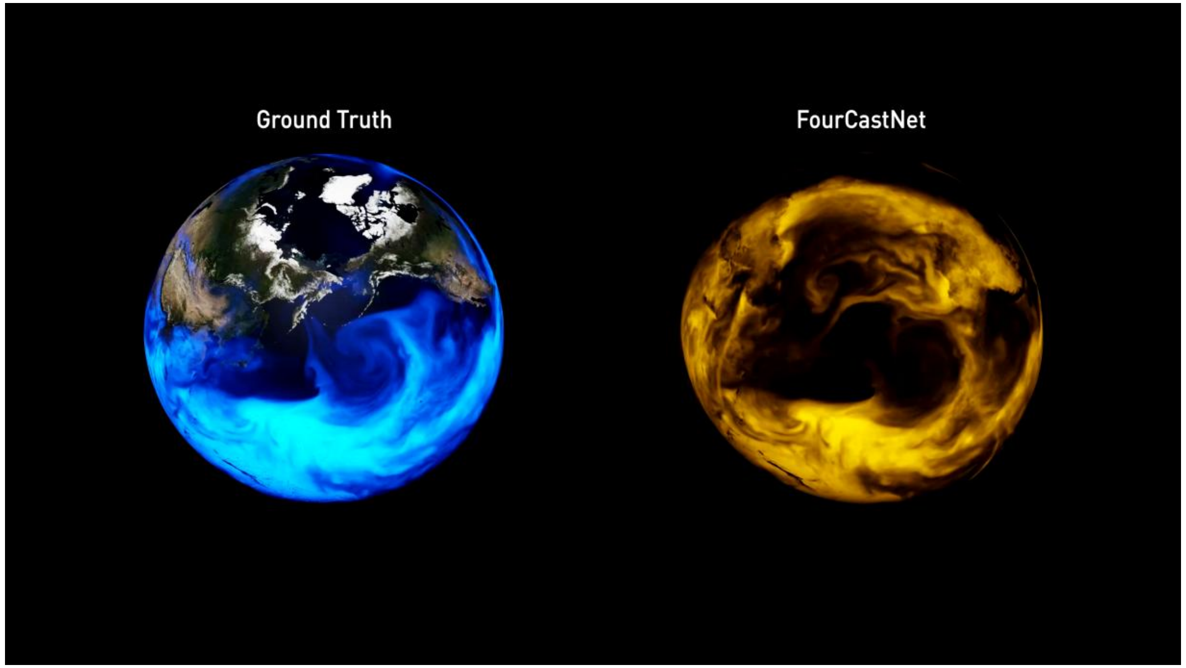
$$\sup_a \|\mathcal{G}^\dagger(a) - \mathcal{G}(a)\|_{\mathcal{U}} \le \epsilon.$$



Learning happens on discretized data

Big Impact Applications

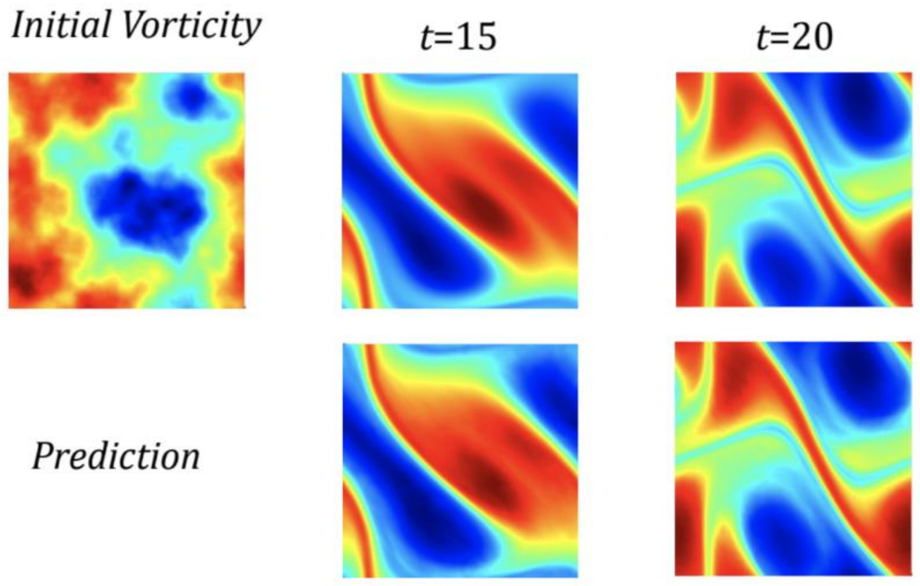


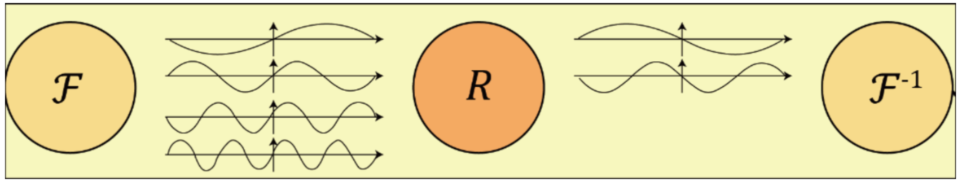


Optimization difficulties in FNO

At the core of FNO is a spectral layer that leverages a discretization-convergent representation in the Fourier domain, which learns weights over a fixed set of frequencies. However, there are optimization difficulties in the training of FNO. However, training FNO presents two significant challenges, particularly in large-scale, high-resolution applications

1. Computing Fourier transform on ***high-resolution inputs*** is computationally intensive but necessary since fine-scale details are needed for solving many PDEs, such as fluid flows.
2. Selecting the ***relevant set of frequencies*** in the spectral layers is challenging, and too many modes can lead to overfitting, while too few can lead to underfitting.





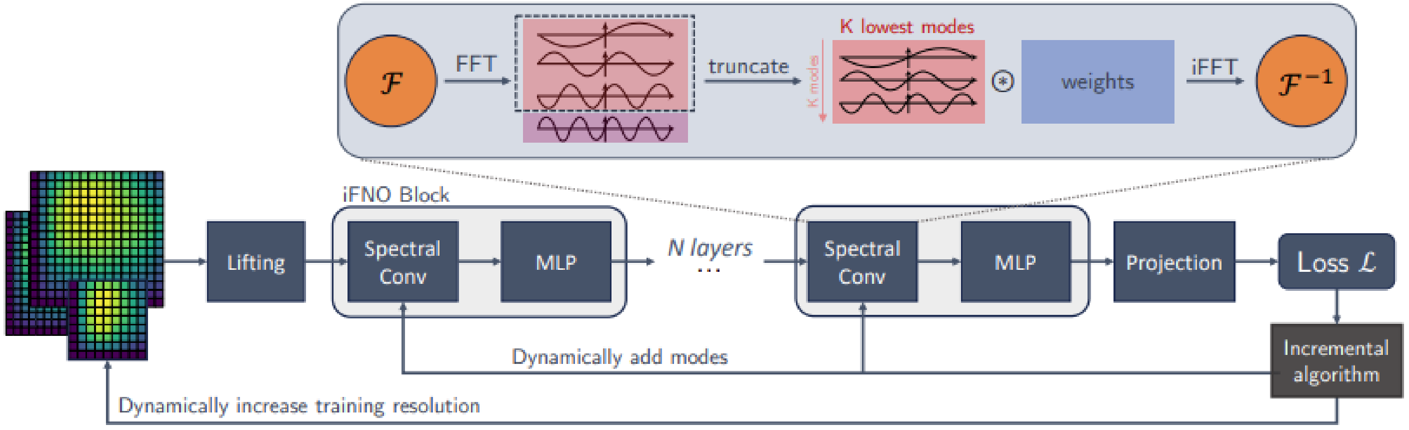
Incremental FNO

Instead of fixing the frequency modes and data resolution, we propose iFNO that *progressively augments both frequency modes and training resolution.*

- Start from **minimal** frequency modes and **lowest** training resolution.
- When the optimization quality is **not improved**, increase both frequency modes and training resolution.
- Repeat the process multiple times until the network **converges.**

Why does it work?

- **iFNO follows spectral bias in deep neural networks**
  Spectral bias suggests that neural networks prioritize the learning of low-frequency components of the target function.
- **iFNO adds explicit constraints over frequency modes and training resolution**
  Additional constraints further regularize the training of FNO.



Advantages:

1. **iFNO improves generalization performance** by regularizing frequency evolution and training resolution in particular a 10% lower testing and using 20% fewer frequency modes compared to the existing FNO.
2. **iFNO reduces training cost** as few frequency modes require less parameters, and low training resolution requires less dimensionality achieving a 30% faster performance, enabling larger scale simulations.

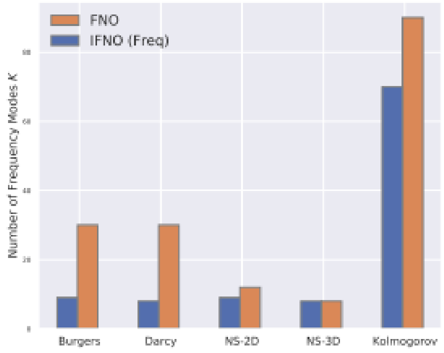
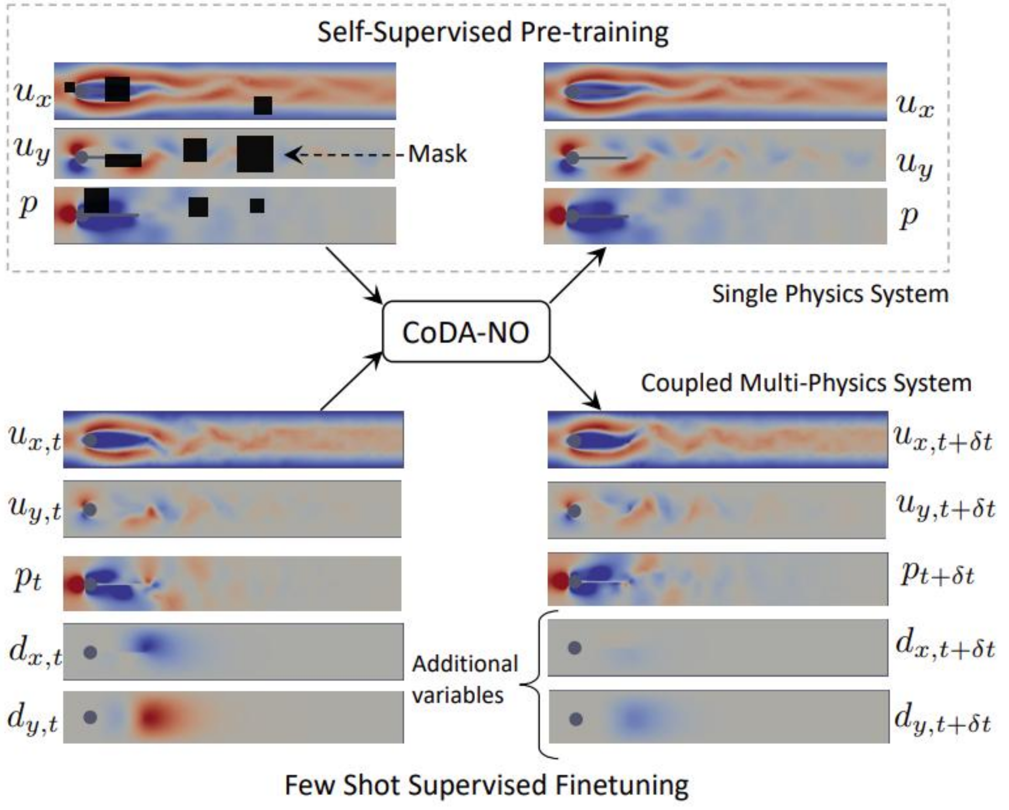


Figure 6: Number of frequency modes K in the converged FNO and iFNO models across datasets. We report K in the first Fourier convolution operator. NS denotes Navier-Stokes equations.

Moving towards a *foundational model*

Existing neural operator architectures face challenges when solving Multiphysics problems with coupled PDEs, due to complex geometries, interactions between physical variables, and the lack of large amounts of high-resolution training data.

- The architecture ***should not be restricted to a fixed number of physical variables***, allowing it to handle PDEs with varying numbers of variables.
- It should be able to learn and predict different PDE systems, even when the number of physical variables differs between the training and target systems.
- When the training and target PDE systems have ***overlapping physical variables***, the architecture should allow for transfer of learned knowledge between the systems.



Codomain Attention Neural Operator - Architecture

1. ***Permutation Equivariant Neural Operator:*** This allows CoDA-NO to process vector-valued input functions $a = [a_1, a_2, \dots, a_d]$, where each a_i represents a different physical variable like velocity, pressure etc. It applies the same integral or pointwise operator to each component a_i sharing weights across variables.

2. ***CoDA-NO Layer:*** This is the core innovation. It extends the ***self-attention mechanism from standard transformers to operate on functions instead of finite-dimensional vectors***. Specifically, for an input vector-valued function $w = [w_1, w_2, \dots, w_d]$:

    1. It tokenizes w along the codomain/channel dimension into separate ***token functions*** w_i treating each physical variable w_i as a token.
    2. For each w_i , it computes query q_i , key k_i , and value v_i functions using learnable operators Q, K, V.
    3. It computes weighted sums of the value functions v_i , using weights from dot products between q_i and k_i in function space.
    4. This gives output token functions o_i for each variable.
    5. Finally, it concatenates these o_i back into the output vector-valued function o.

CoDA-NO Architecture (Cont'd)

1. ***Variable Specific Positional Encoding (VSPE):*** It learns positional encodings e_j for each input variable a_j, concatenating e_j with a_j to obtain extended input functions. Then we just applied a shared point wise lifting operator to all of these extended input functions.
2. ***Function Space Normalization:*** It extends normalization layers like BatchNorm to operate on functions instead of vectors.

Lastly, to effectively handle non-uniform complex geometries, we follow the GINO architecture where a GNO is used as an encoding and decoding module. We note that all of the integral operators are FNO. Finally, by tokenizing along the codomain and applying self-attention there, CoDA-NO can explicitly model interactions between different physical variables of multiphysics PDEs within a single model.

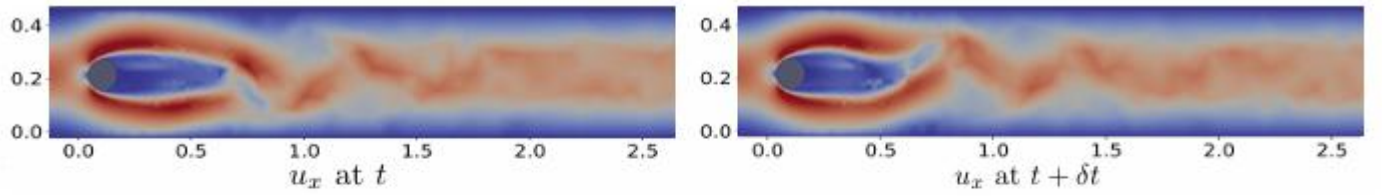


Figure 3: Visualization of **horizontal velocity** u_x at t and $t + \delta t$ time step.

CoDA-NO – Diagram

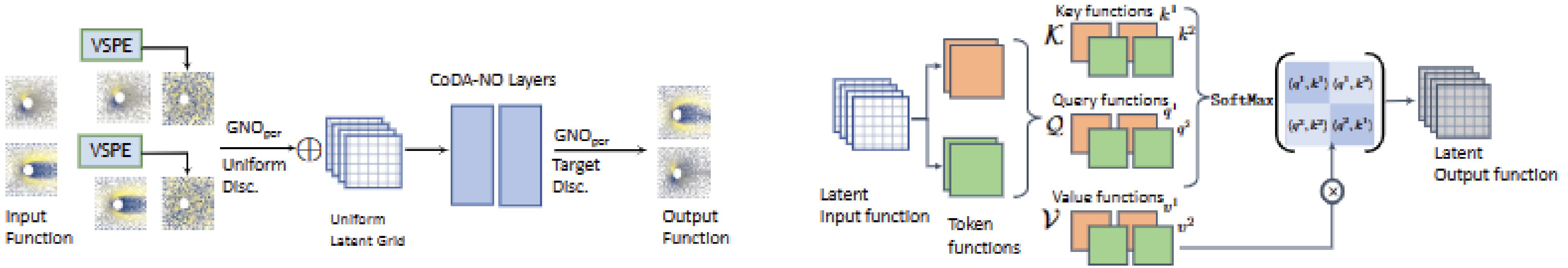


Figure 2. On the **left**, we illustrate the architecture of the Codomain Attention Neural Operator. Each physical variable (or co-domain) of the input function is concatenated with *variable specific positional encoding* (VSPE). Each variable, along with the VSPE, is passed through a GNO layer, which maps from the given non-uniform geometry to a latent regular grid. Then, the output on a uniform grid is passed through a series of CoDA-NO layers. Lastly, the output of the stacked CoDA-NO layers is mapped onto the domain of the output geometry for each query point using another GNO layer. On the **right**, we illustrate the mechanism of codomain attention. At each CoDA-NO layer, the input function is tokenized codomain-wise, and each token function is passed through the $\mathcal{K}$, $\mathcal{Q}$, and $\mathcal{V}$ operators to get key, query, and value functions $\{k^1, k^2\}$, $\{q^1, q^2\}$, and $\{v^1, v^2\}$ respectively. The output function is calculated via an extension of the self-attention mechanism to the function space.

Codomain Attention Neural Operator (Training)

Self-supervised Pretraining:

- The objective is to train the model to reconstruct the original input function from its masked version.
- The input function is masked by setting values of a percentage of mesh points to zero for some variables, or by completely masking certain variables.
- The model's encoding component acts as the Encoder, while the decoding component is the Reconstructor during this phase

Supervised Fine-tuning:

- The Reconstructor from the pretraining phase is replaced by a randomly initialized Predictor module in the decoding component.
- The parameters of the Encoder and variable-specific positional encodings (VSPEs) are initialized from the pretrained weights.
- If the fine-tuning (target) PDE introduces new variables not present during pretraining, additional VSPEs are trained for these new variables to adapt to the expanded set of variables.

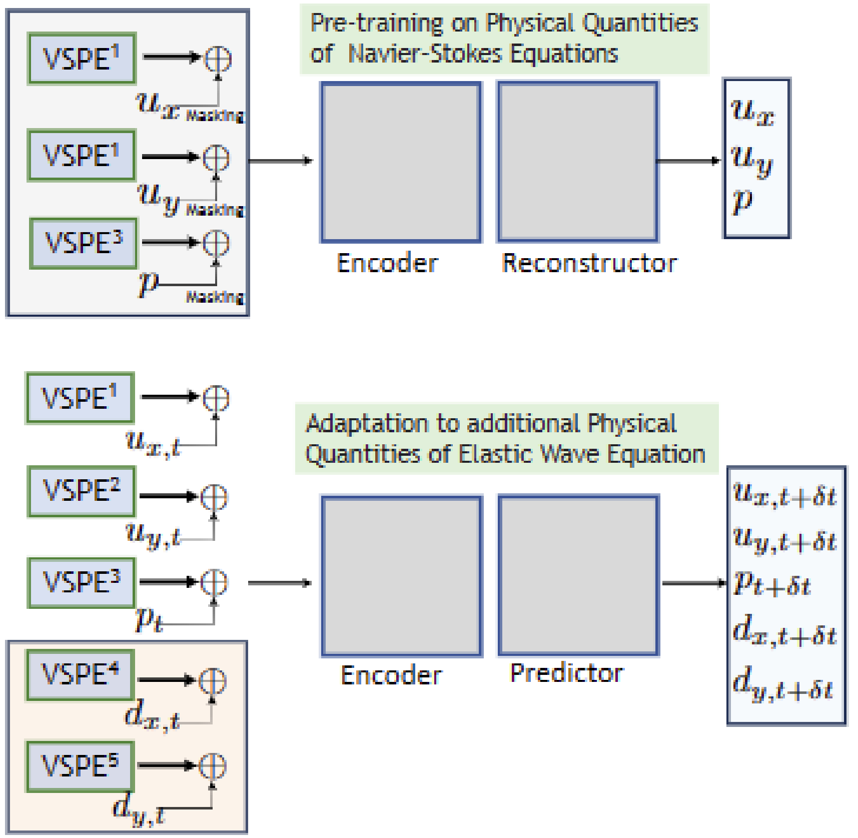
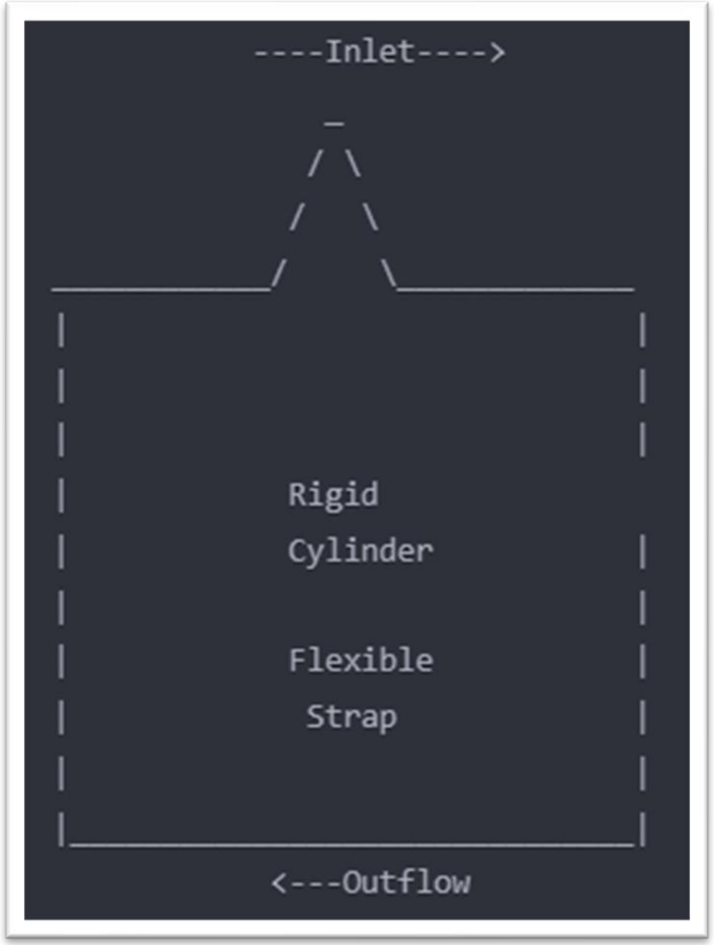


Figure 3. **Test time adaptation to new physical variables.** The model is pre-trained on the Navier-Stokes equation dataset, which comprises physical variables such as velocities u_x, u_y, and pressure p. To adapt this pre-trained model on a fluid-solid interaction dataset containing an additional Elastic wave equation with new displacement variables d_x and d_y, it is only necessary to add two additional VSPEs to the whole pipeline.

Datasets

1. **PDEBench** features a much wider range of PDEs than existing benchmarks. Datasets employed in our study encompass diverse PDE types and parameters like Navier-Stokes equations, diffusion-reaction equations, and shallow-water equations.
2. **Fluid Dynamics Problem (NS):** Governed by the Navier-Stokes equation
    1. Involves a Newtonian, incompressible fluid impinging on a rigid cylinder with an attached rigid strap. The physical variables are the fluid velocity (u) and pressure (p)
3. **Fluid-Structure Interaction Problem (NS+EW):** Coupled system governed by both the Navier-Stokes equation for fluid and the Elastic wave equation for the solid
    1. Involves a Newtonian, incompressible fluid interacting with an elastic, compressible solid object (cylinder with an attached deformable elastic strap).The physical variables are the fluid velocity (u), pressure (p), and the solid displacement field (d).



Results

•**Model**: CoDA-NO outperforms all baselines

•**Pre-training**:

- •NS+EW pre-training performs best overall
- •NS pre-training also effective, especially for fluid dynamics (NS) tasks

•**Few-shot learning**:

- •CoDA-NO shows significant improvement with limited data (5-25 samples)
- •Performance gap narrows but remains as sample size increases to 100

•**Viscosity/Reynolds number**:

- •Consistent performance across Re = 400 and Re = 4000
- •Adapts well to more turbulent flows (Re = 4000)

•**Task generalization**:

- •Effectively transfers from fluid dynamics (NS) to fluid-structure interaction (NS+EW)

Table 1: Test L_2 loss for fluid dynamics (NS) and fluid-solid interaction (NS+EW) datasets with viscosity $Re = 400$ and $Re = 4000$ for different numbers of few-shot training samples.

Model	Pretrain Dataset	$Re = 400$						$Re = 4000$		
		# Few Shot Training Samples								
		5		25		100		5	25	100
		Evaluation Dataset								
		NS	NS+EW	NS	NS+EW	NS	NS+EW	NS+EW	NS+EW	NS+EW
GINO	-	0.200	0.122	0.047	0.053	0.022	0.043	0.717	0.292	0.136
DeepO	-	0.686	0.482	0.259	0.198	0.107	0.107	0.889	0.545	0.259
GNN	-	0.038	<u>0.045</u>	0.008	0.009	0.008	0.009	0.374	0.310	0.132
ViT	-	0.271	0.211	0.061	0.113	0.017	0.021	0.878	0.409	0.164
U-Net	-	13.33	3.579	0.565	0.842	0.141	0.203	3.256	0.563	0.292
Ours	-	0.182	0.051	0.008	0.084	0.006	<u>0.004</u>	0.326	0.264	0.070
	NS	<u>0.025</u>	0.071	<u>0.007</u>	<u>0.008</u>	**0.004**	0.005	0.366	0.161	0.079
	NS+EW	**0.024**	**0.040**	**0.006**	**0.005**	<u>0.005</u>	**0.003**	**0.308**	**0.143**	**0.069**

Results

- **Full CoDA-NO**:
  - Best performance across all scenarios
  - Especially effective with few-shot learning (5-25 samples)
- **Impact of Components**:
  - CoDA-NO without VSPE & Norm: Significant performance drop
  - Adding CoDA-NO alone: Major improvement, especially for NS+EW
  - VSPE: Critical for model convergence with limited data
  - Normalization: Essential for effective training
- **Pre-training Effects**:
  - NS+EW pre-training: Best overall performance
  - NS pre-training: Effective, especially for NS tasks
- **Generalization**:
  - Full CoDA-NO shows best adaptation from NS to NS+EW tasks
- **Key Takeaway**: All components (CoDA-NO, VSPE, Normalization) are crucial for optimal performance and generalization

CoDA-NO	VSPE	Norm	Pretrain Dataset	# Few Shot Training Samples 5		25		100	
				NS	NS+EW	NS	NS+EW	NS	NS+EW
✗	✗	✗	✗	0.271	0.211	0.061	0.113	0.017	0.020
✓	✗	✗	✗	0.182	0.051	0.008	0.084	0.006	0.004
✓	✗	✓	NS	0.049	0.079	0.009	0.0132	0.004	0.009
✓	✗	✓	NS EW	0.045	0.057	0.010	0.011	0.008	0.004
✓	✓	✗	NS	*	*	0.023	*	0.008	0.006
✓	✓	✗	NS EW	0.057	0.232	0.012	0.052	0.006	0.006
✓	✓	✓	NS	0.025	0.071	0.007	0.008	**0.004**	0.005
✓	✓	✓	NS EW	**0.024**	**0.040**	**0.006**	**0.005**	0.005	**0.003**

Results

Zero-Shot Super Resolution Performance

- **Task**: Fluid-Solid (NS-EW) Interaction Problem
- **Setting**: Trained on 1317 mesh points, tested on 2193 points

Key Findings:

1. CoDA-NO outperforms all baselines significantly
2. Pre-training improves performance:
    1. NS pre-training slightly better than NS-ES pre-training
3. Performance across viscosities (μ):
    1. Best: $\mu = 5$
    2. Good: $\mu = 10$
    3. Challenging: $\mu = 1$
4. Baseline comparisons:
    1. ViT performs best among baselines
    2. U-Net and DeepO struggle most with this task

Conclusion: CoDA-NO demonstrates superior generalization to higher resolution meshes without specific training.

Model	Pretrain Dataset	Fluid Viscocities $\mu = 5$	$\mu = 1$	$\mu = 10$
U-Net	-	0.144	0.267	0.216
Vit	-	0.052	0.175	0.046
GINO	-	0.069	0.103	0.0711
DeepO	-	0.113	0.107	0.357
GNN	-	0.223	0.211	0.247
CoDA-NO	NS-ES	0.041	0.063	0.048
CoDA-NO	NS	**0.032**	**0.049**	**0.035**

Results

- **Single-Physics PDEs:**
    - Shallow Water Equations (SWE):
        - CoDA-NO: 0.04072 (12% improvement over FNO)
    - Diffusion Equation (DIFF):
        - CoDA-NO: 0.00810 (43% improvement over FNO)
- **Multi-Physics Dataset (NS+DIFF+SWE):**
    - CoDA-NO: 0.00302 (slightly higher than FNO's 0.00118)
- **Reconstruction Error:**
    - CoDA-NO consistently lower than FNO across all datasets
- **Model Size Comparison:**
    - CoDA-NO: 11 million parameters
    - FNO: 1.9 billion parameters (98% larger)
- **Performance vs. DPOT:**
    - Comparable performance to DPOT on DIFF dataset
    - CoDA-NO: 0.0081 vs. DPOT-L-500: 0.0073
    - Achieved with significantly fewer parameters and training epochs

Model	Dataset	Test Error	
		Prediction Error	Reconstruction Error
CoDA-NO	SWE	**0.04072**	**0.00460**
FNO		0.04631	0.03262
CoDA-NO	DIFF	**0.00810**	**0.00041**
FNO		0.01415	0.01894
CoDA-NO	NS+DIFF+SWE	0.00302	**0.00006**
FNO		**0.00118**	0.00287

Model	Model Parameters
CoDA-NO	11M
FNO	1.9B
DPOT-FT-T	7M
DPOT-FT-S	30M
DPOT-FT-M	100M
DPOT-FT-L	500M

Overall picture

1. **Superior Performance:** Outperforms baselines in few-shot learning and zero-shot super-resolution tasks
2. **Adaptability:** Seamlessly handles varying numbers of physical variables and complex geometries
3. **Generalization:** Effectively transfers knowledge between single and multiphysics problems
4. **Robustness:** Maintains performance across various Reynolds numbers, including turbulent flows

CoDA-NO demonstrates potential as a versatile foundation model for solving diverse multiphysics PDEs, opening new avenues for efficient scientific computing.

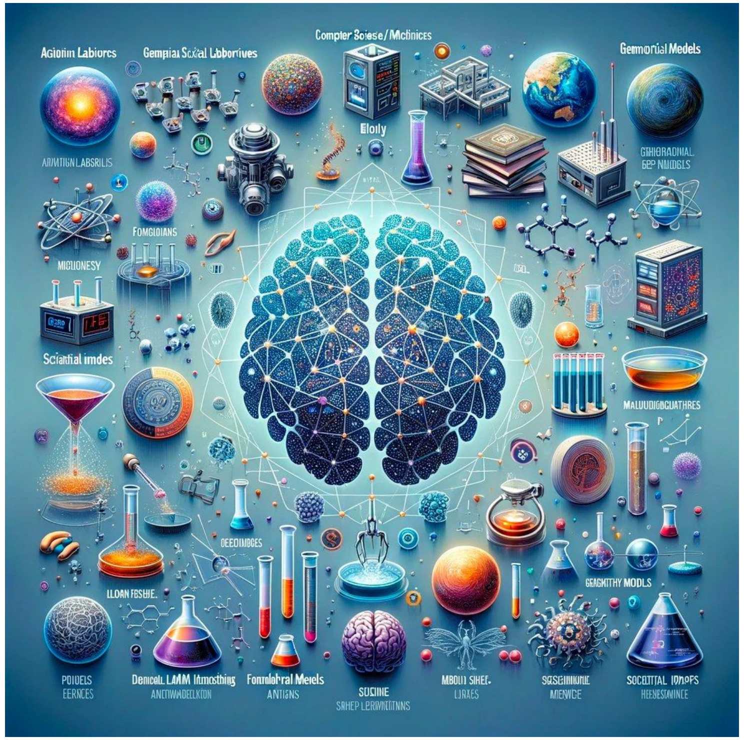
Some open problems in Neural Operator

- Scaling up is a big challenge
- The resolution in the intermediate layers is designer choice, how it should be done?
- Neural Operator architectures are still primitive.
- What can these architectures bring to CV?
- Uncertainty Quantification is essential, but how can it be done in function spaces?
- Reinforcement learning and control + Neural Operators = How can it be done?
- Unsupervised learning – representation learning in function spaces?
- What about meta learning, adversarial robustness, transfer learning etc?

Takeaway: Lots to discover in this field.

Collaboration

- **Be aware of challenges:**
  - Often domain experts are pessimistic about ML
  - Initial ML methods are often not as good as the existing paradigms
  - Domain experts don't know much ML as ML-ists don't know much about other fields.
  - Needs joint development and language bridge.
  - The data is not generated having ML in mind



Caltech + UChicago AI4Science Initiative

Codebase

Neural Operator is an open-source library with a permissive license for scientific ML. It provides a unified API for different neural architectures for operator learning. Our mission is to democratize state-of-the-art algorithms like yours through a unified codebase.

In particular, we have added a Physics-based Informed Neural Operator (PINO) as an extension of PINNs that overcomes the critical limitations by incorporating both data and physics losses at varying resolutions. This allows for better generalization and extrapolation to resolutions beyond the training data and is much more suitable for multi-scale dynamic PDE systems.

The New codebase for all Neural Operators are present here: [neuraloperator/neuraloperator: Learning in infinite dimension with neural operators]

Must watch Resources

- https://youtu.be/_j7bceE9AyA - ICML 2024 Tutorial"Machine Learning on Function spaces – By Kamyar Azizzadensheli
- https://youtu.be/6bl5XZ8kOzI - AI That Connects the Digital and Physical Worlds | Anima Anandkumar | TED
- https://youtu.be/PpTkY8IgV3c - Tutorial on Neural Operators by Zongyi Li
- https://youtu.be/y5EJr4ofGOc - ML for Solving PDEs: Neural Operators on Function Spaces by Anima Anandkumar
- ETH Zürich DLSC: Course Introduction – YouTube – Playlist by ETH on scientific computing
- https://youtu.be/W8PybqAk6Ik - Fourier Neural Operator (FNO) [Physics Informed Machine Learning] by Steve Brunton

Acknowledgements

Thank you to Umaa Rebbapragada for inviting me to speak at JPL.

All this work is mostly done by the team here at Caltech and Nvidia under Anima Anandkumar.

Special thanks to Zongyi Li, Kamyar Azzizadenesheli (for slides as well), Jean Kossaifi, Jiawei Zhao, Julius Berner, Ashiq Rahman, David Pitt. I thank the rest of the contributors and the other members of the teams for their help and contributions.

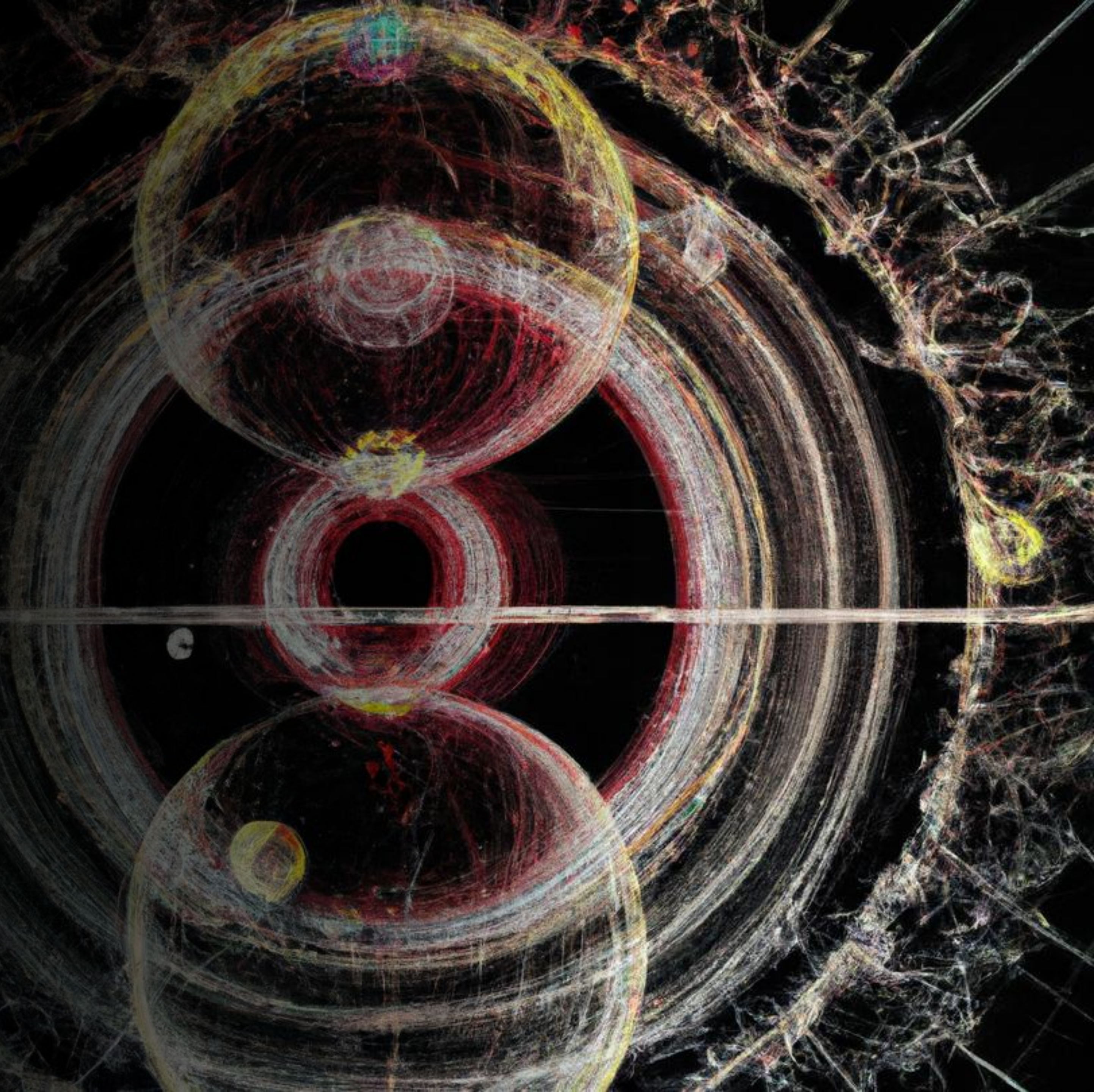
Conclusion

AI for science
Ai of Science
Ai + Science

Future of science

Two Nobel Prizes this year:)

Any Questions?



References

1. Neural operator: Graph kernel network for partial differential equations, Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar
2. Neural operator: Learning maps between function spaces, Nikola Kovachki, Zongyi Li, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar
3. Blog post by Zongyi Li, https://zongyi-li.github.io/blog/2020/graph-pde/

Caltech