Physics 361 - Machine Learning in Physics

Lecture 27 – PDEs, symbolic math, inverse problems

April 30th 2024

AI
∩
Universe
$\mathcal{L} = -\frac{1}{4} F_{\mu\nu} F^{\mu\nu}$
$+ i \bar{\psi} \not{D} \psi + h.c.$
$+ \bar{\psi}_i y_{ij} \psi_j \phi + h.c.$
$+ |D_\mu \phi|^2 - V(\phi)$

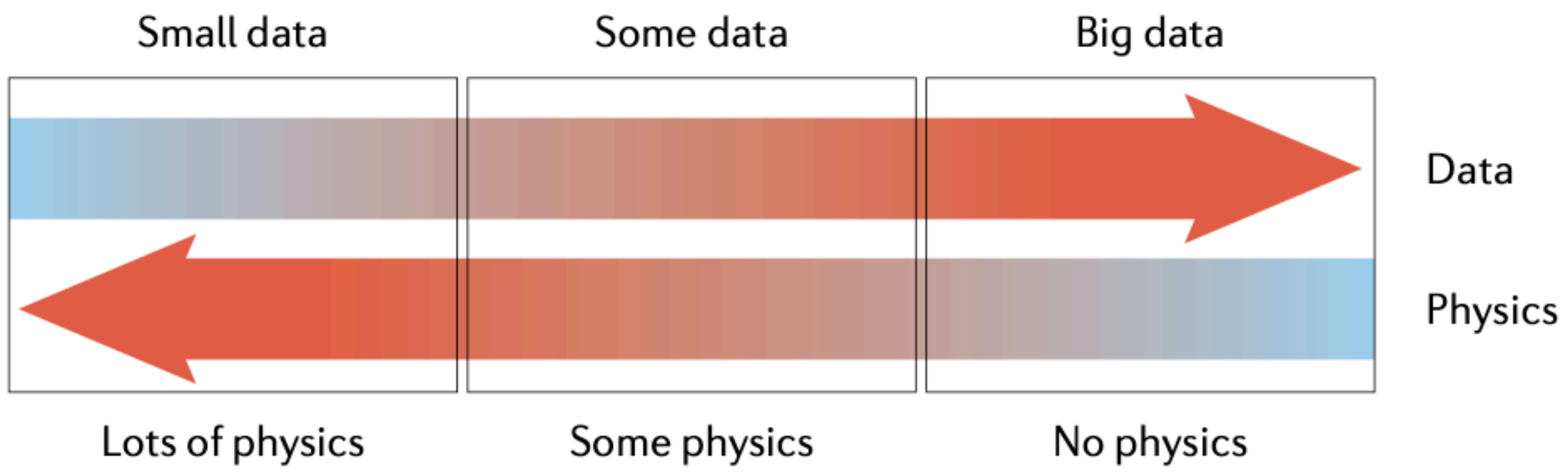
Moritz Münchmeyer

Some recent work at the interface between physics and machine learning

- For our last lecture I want to **briefly discuss some recent interesting research.**
- I will pick topics/papers that I am interested in myself, without being very systematic. Not all work that I present here is influential, though some is.
- I hope this will inspire you to work on ideas in this field.

“Physics-informed machine learning”

- Review: https://www.nature.com/articles/s42254-021-00314-5 (plots from this source)
- Simulating physics problems using the numerical discretization of partial differential equations (PDEs) remains difficult.
- Instead, machine learning based methods can be combined with physics (including physical laws or symmetries). This is called “physics-informed machine learning”.
- Generally speaking, the less we have training data the more physics knowledge can help the model to perform.



“Physics-informed machine learning”

- Physics knowledge can be included in machine learning in 3 general ways:
    - **Observational**: Physical properties (e.g. symmetries) can be learned directly from the observations.
    - **Inductive biases**: Physical properties can be exactly enforced in the model. A simple example are CNNs which are hard-coding translational invariance. E.g. Formally one can make a NN invariant under any symmetry group. **Physics can thus be encoded in the model architecture**. However, this often leads to complex implementations that are difficult to scale.
    - **Learning bias**: Physical constraints can be enforced in a soft way (i.e. not exact) by adding a term to the loss that penalizes violating the constraint (e.g. conservation of mass).
- **Hybrid methods** combine several of these.

Solving PDEs with PINNs

- PINNs: “Physics informed neural networks”
- Physics-informed neural networks (PINNs) integrate the information from both the measurements and partial differential equations (PDEs) by embedding the PDEs into the loss function of a neural network using automatic differentiation.
- Example: solving the viscous Burgers’ equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

- We also have some data (boundary condition) so that the solution is uniquely defined.
- Idea: We put the residual of the PDE into the loss function and solve the problem as an optimization problem with auto-differentiation.

Solving PDEs with PINNs

- The Loss thus is

$$\mathcal{L} = w_{\text{data}}\mathcal{L}_{\text{data}} + w_{\text{PDE}}\mathcal{L}_{\text{PDE}},$$

where

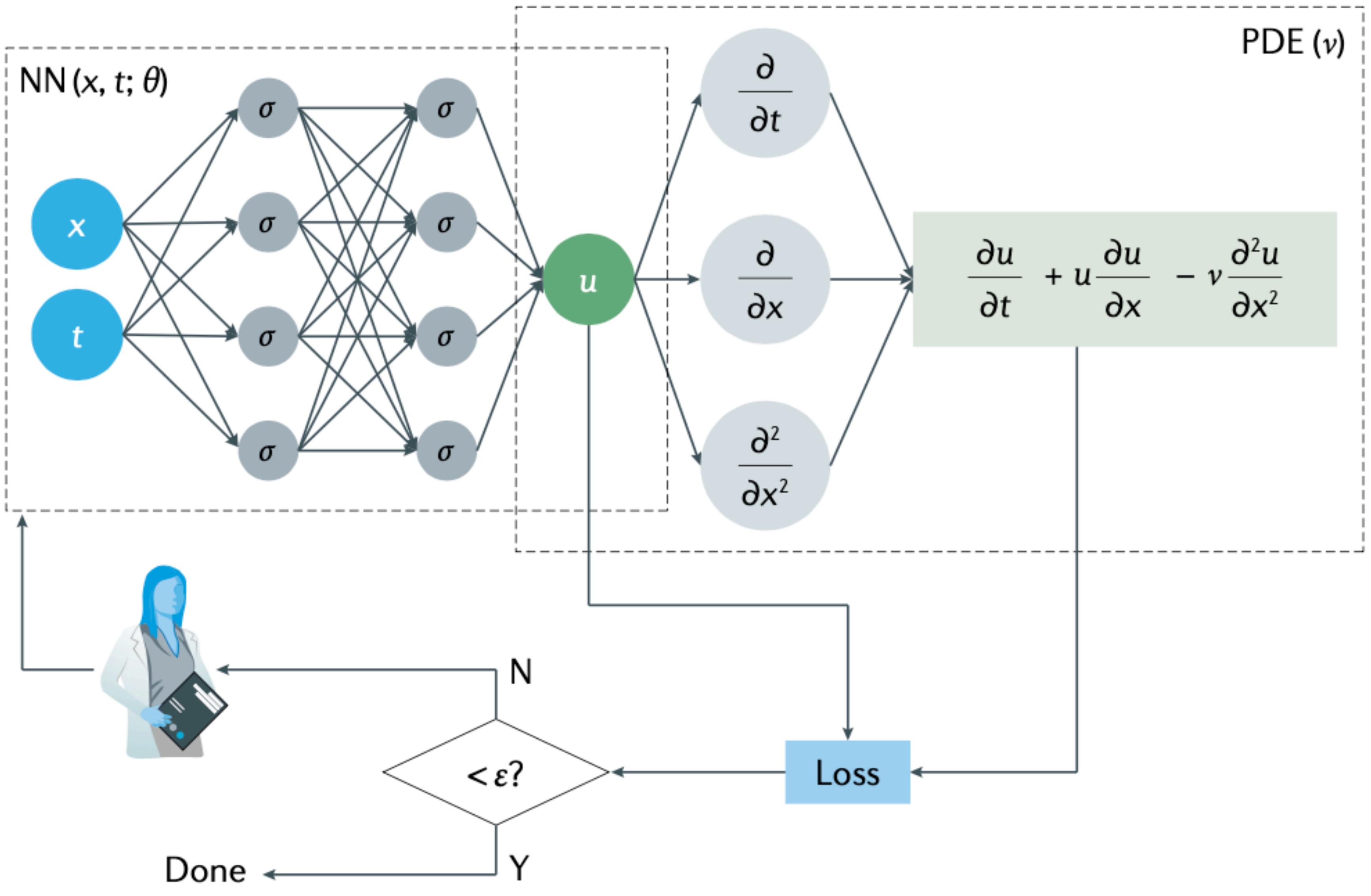
$$\mathcal{L}_{\text{data}} = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} (u(x_i, t_i) - u_i)^2 \quad \text{and}$$

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_{j=1}^{N_{\text{PDE}}} \left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} \right)^2 \Big|_{(x_j, t_j)}$$

Here $\{(x_i, t_i)\}$ and $\{(x_j, t_j)\}$ are two sets of points sampled at the initial/boundary locations and in the entire domain, respectively, and u_i are values of u at (x_i, t_i); w_{data} and w_{PDE} are the weights used to balance the interplay between the two loss terms. These weights can be user-defined or tuned automatically, and play an important role in improving the trainability of PINNs[76,173].

Solving PDEs with PINNs

- The PDE solution is specified by a neural network (rather than some pixelated discretization of space), so we get a continuous function.



NN (x, t; θ)
x
t
σ
u
PDE (ν)
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2}$
Loss
$< \varepsilon ?$
N
Y
Done

Algorithm 1: The PINN algorithm.
Construct a neural network (NN) $u(x, t; \boldsymbol{\theta})$ with $\boldsymbol{\theta}$ the set of trainable weights w and biases b, and σ denotes a nonlinear activation function. Specify the measurement data $\{x_i, t_i, u_i\}$ for u and the residual points $\{x_j, t_j\}$ for the PDE. Specify the loss $\mathcal{L}$ in Eq. (3) by summing the weighted losses of the data and PDE. Train the NN to find the best parameters $\boldsymbol{\theta}^*$ by minimizing the loss $\mathcal{L}$.

Neural Operators

- https://arxiv.org/abs/2309.15325 Neural Operators for Accelerating Scientific Simulations and Design
- The core idea behind Fourier Neural Operator is to **perform neural network operations in the Fourier space** (frequency domain), where convolution operations become multiplications. This approach efficiently captures the global interactions in the data, which are crucial for accurately solving PDEs.

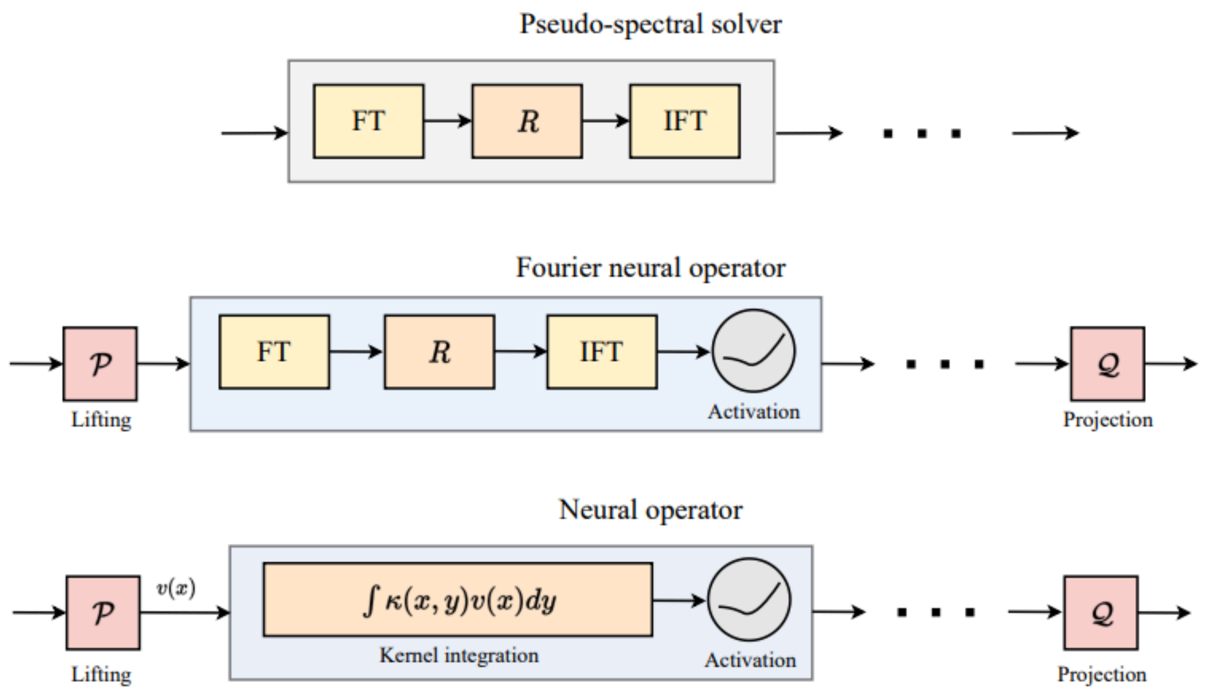


Fig. 3: Diagram comparing pseudo-spectral solver, Fourier Neural Operator (FNO), and the general Neural Operator architecture. FT and IFT refer to Fourier and Inverse Fourier Transforms. In general, lifting and projection operators $\mathcal{P}$, $\mathcal{Q}$ can be non-linear. Pseudo-spectral solvers are popular numerical solvers for fluid dynamics where the Fourier basis is utilized, and operations are iteratively carried out, as shown. The Fourier Neural Operator (FNO) is inspired by the pseudo-spectral solver, but has a non-linear representation that is learned. FNO is a special case of the Neural-Operator framework, shown in the last row, where the kernel integration can be carried out through different methods, e.g., direct discretization or through Fourier transform.

Comparing PDE solving methods

- https://arxiv.org/abs/2210.07182 PDEBENCH: An Extensive Benchmark for Scientific Machine Learning

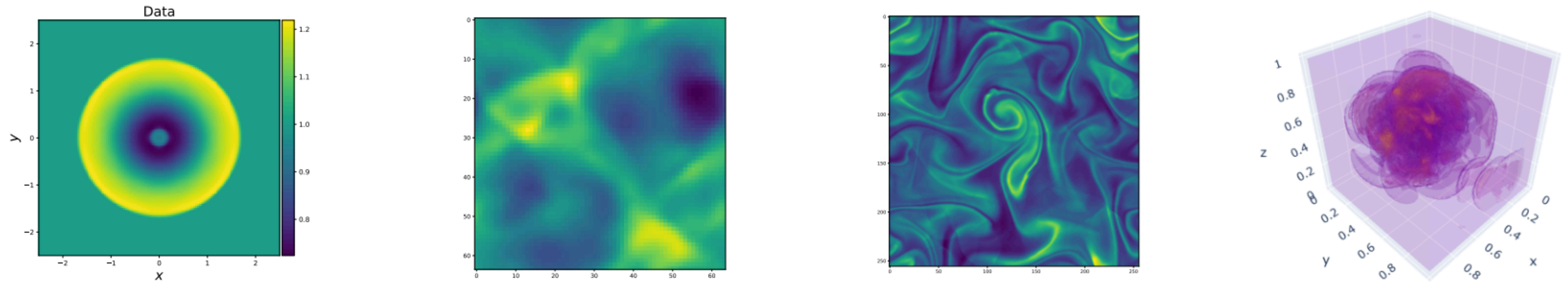


Figure 1: PDEBENCH provides multiple non-trivial challenges from the Sciences to benchmark current and future ML methods, including wave propagation and turbulent flow in 2D and 3D

Table 1: Summary of PDEBENCH's datasets with their respective number of spatial dimensions N_d, time dependency, spatial resolution N_s, temporal resolution N_t, and number of samples generated.

PDE	N_d	Time	N_s	N_t	Number of samples
advection	1	yes	1 024	200	10 000
Burgers'	1	yes	1 024	200	10 000
diffusion-reaction	1	yes	1 024	200	10 000
diffusion-reaction	2	yes	128×128	100	1000
diffusion-sorption	1	yes	1 024	100	10 000
compressible Navier-Stokes	1	yes	1 024	100	10 000
compressible Navier-Stokes	2	yes	512×512	21	1000
compressible Navier-Stokes	3	yes	$128 \times 128 \times 128$	21	100
incompressible Navier-Stokes	2	yes	256×256	1000	1000
Darcy flow	2	no	128×128	–	10 000
shallow-water	2	yes	128×128	100	1000

Comparing PDE methods

- The objective is to find some ML-based surrogate, sometimes referred to as an emulator, of the forward propagator (i.e. the next time step).
- Baseline ML models for PDE solving:

U-Net U-Net [48] is an auto-encoding neural network architecture used for processing images using multi-resolution convolutional networks with skip layers. U-Net is a black-box machine learning model that propagates information efficiently at different scales. Here, we extended the original implementation, which uses 2D-CNN, to the spatial dimension of the PDEs (i.e. 1D,3D).

Fourier neural operator (FNO) FNO [32] belongs to the family of Neural Operators (NOs), designed to approximate the forward propagator of PDEs. FNO learns a resolution-invariant NO by working in the Fourier space and has shown success in learning challenging PDEs.

Physics-Informed Neural Networks (PINNs) Physics-informed neural networks [47] are methods for solving differential equations using a neural network $u_\theta(t, x)$ to approximate the solution by turning it into a multi-objective optimization problem. The neural network is trained to minimize the PDE residual as well as the error with regard to the boundary and initial conditions. PINNs naturally integrate observational data [30], but require retraining for each new condition.

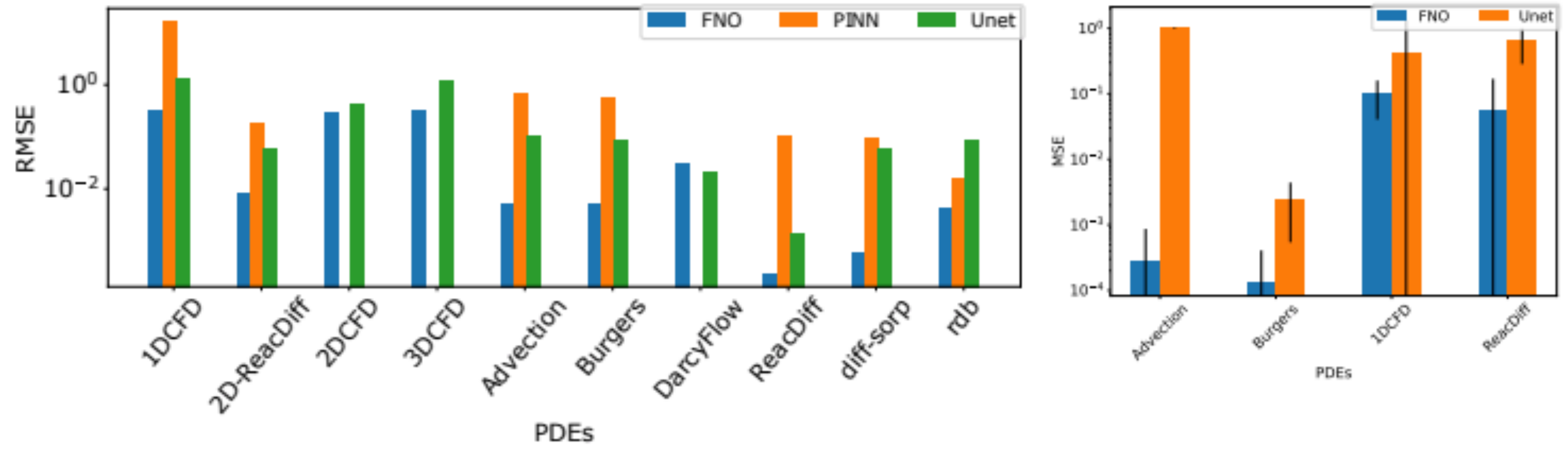
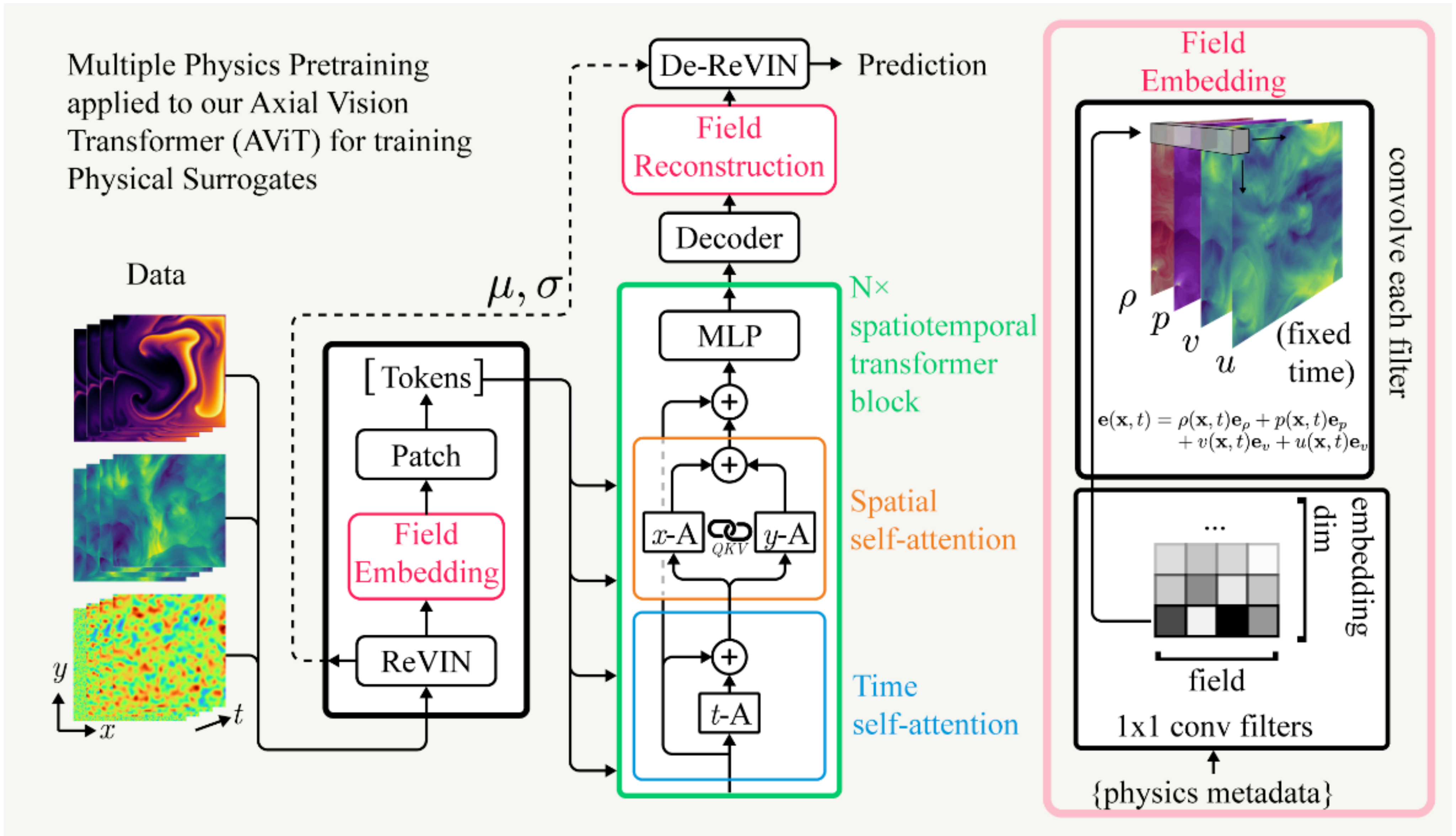


Figure 2: Comparisons of baseline models' performance for different problems for (a) the forward problem and (b) the inverse problem.

- Currently neural solvers often don't outperform classical methods but they are more flexible.

Foundation model for PDEs?

- https://arxiv.org/abs/2310.02994
    - At a fundamental level, many physical systems share underlying principles. Many of the equations describing physical behavior are derived from universal properties like conservation laws or invariances which persist across diverse disciplines like fluids, climate science, astrophysics, and chemistry. Can we learn these shared features ahead of time through pretraining and accelerate the development of models for new physical systems?
    - Results are somewhat encouraging.

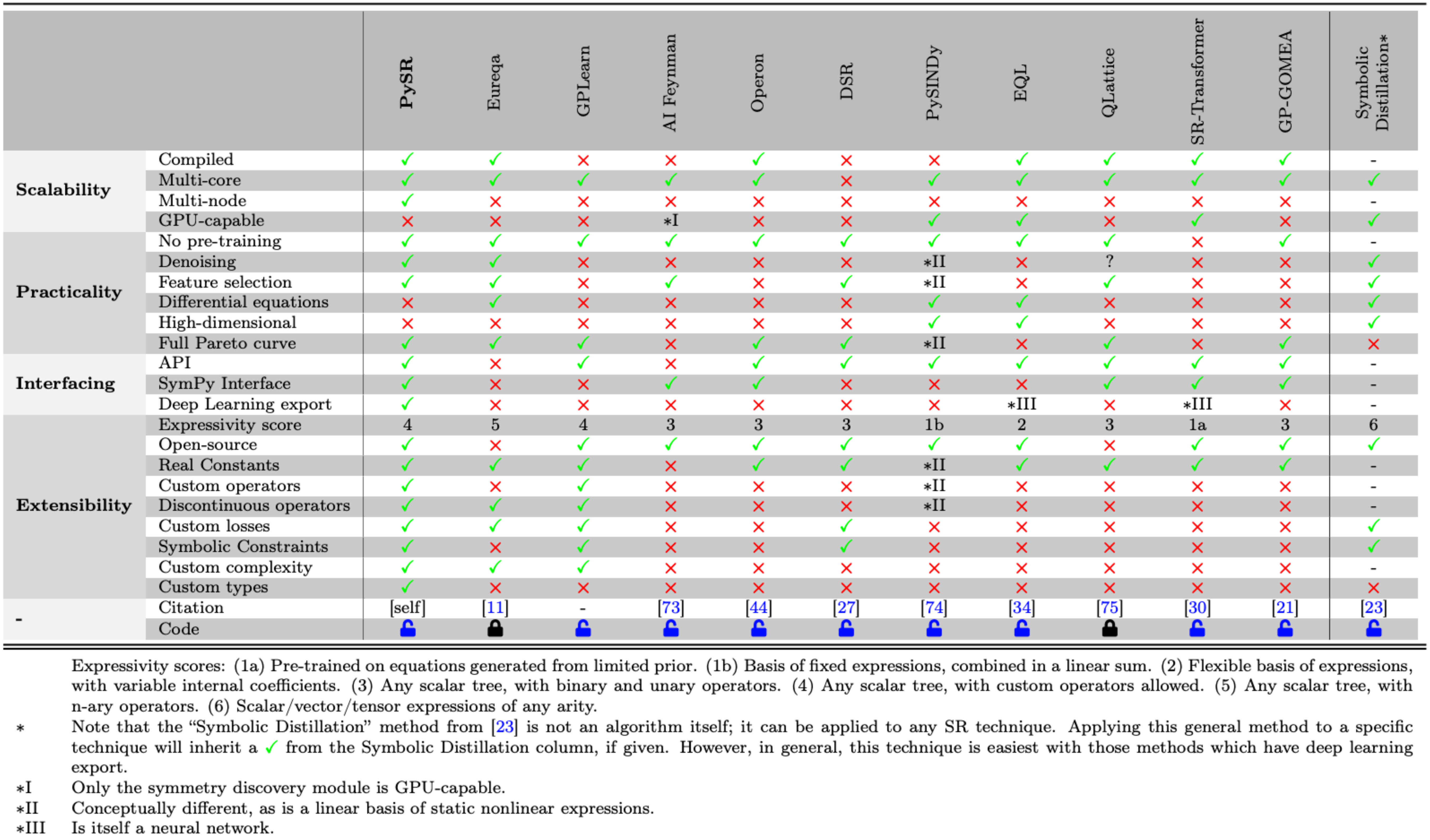

Multiple Physics Pretraining applied to our Axial Vision Transformer (AViT) for training Physical Surrogates
Data
y
x
t
μ, σ
[Tokens]
Patch
Field Embedding
ReVIN
De-ReVIN
Prediction
Field Reconstruction
Decoder
MLP
N× spatiotemporal transformer block
x-A
y-A
QKV
Spatial self-attention
t-A
Time self-attention
Field Embedding
ρ
p
v
u
(fixed time)
convolve each filter
$\mathbf{e}(\mathbf{x},t) = \rho(\mathbf{x},t)\mathbf{e}_\rho + p(\mathbf{x},t)\mathbf{e}_p + v(\mathbf{x},t)\mathbf{e}_v + u(\mathbf{x},t)\mathbf{e}_v$
...
embedding dim
field
1x1 conv filters
{physics metadata}

Discovering symbolic laws from data

- **Discovering symbolic laws = Symbolic regression**
- Symbolic Regression is usually done with Genetic algorithms, building a population of possible expressions that evolve (i.e. are modified). See for example the PySR paper https://arxiv.org/abs/2305.01582 .
- Some examples of combining symbolic regression with machine learning:
  - https://arxiv.org/abs/2006.11287 Discovering Symbolic Models from Deep Learning with Inductive Biases, 2202.02306
    - We develop a general approach to distill symbolic representations of a learned deep model by introducing strong inductive biases. We focus on Graph Neural Networks (GNNs). The technique works as follows: we first encourage sparse latent representations when we train a GNN in a supervised setting, then we apply symbolic regression to components of the learned model to extract explicit physical relations. We find the correct known equations, including force laws and Hamiltonians, can be extracted from the neural network
  - https://www.science.org/doi/10.1126/sciadv.aay2631 AI Feynman: A physics-inspired method for symbolic regression
    - We develop a recursive multidimensional symbolic regression algorithm that combines neural network fitting with a suite of physics-inspired techniques. We apply it to 100 equations from the Feynman Lectures on Physics, and it discovers all of them.

Symbolic Regression methods

Comparison from https://arxiv.org/abs/2305.01582



		PySR	Eureqa	GPLearn	AI Feynman	Operon	DSR	PySINDy	EQL	QLattice	SR-Transformer	GP-GOMEA	Symbolic Distillation*
Scalability	Compiled	✓	✓	✗	✗	✓	✗	✗	✓	✓	✓	✓	-
	Multi-core	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
	Multi-node	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	-
	GPU-capable	✗	✗	✗	∗I	✗	✗	✓	✓	✗	✓	✗	✓
Practicality	No pre-training	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	-
	Denoising	✓	✓	✗	✗	✗	✗	∗II	✗	?	✗	✗	✓
	Feature selection	✓	✓	✗	✓	✗	✓	∗II	✗	✓	✗	✗	✓
	Differential equations	✗	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓
	High-dimensional	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓
	Full Pareto curve	✓	✓	✓	✗	✓	✓	∗II	✗	✓	✗	✓	✗
Interfacing	API	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓	-
	SymPy Interface	✓	✗	✗	✓	✓	✗	✗	✗	✓	✓	✓	-
	Deep Learning export	✓	✗	✗	✗	✗	✗	✗	∗III	✗	∗III	✗	-
Extensibility	Expressivity score	4	5	4	3	3	3	1b	2	3	1a	3	6
	Open-source	✓	✗	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
	Real Constants	✓	✓	✓	✗	✓	✓	∗II	✓	✓	✓	✓	-
	Custom operators	✓	✗	✓	✗	✗	✗	∗II	✗	✗	✗	✗	-
	Discontinuous operators	✓	✓	✓	✗	✗	✗	∗II	✗	✗	✗	✗	-
	Custom losses	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✓
	Symbolic Constraints	✓	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✓
	Custom complexity	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	-
	Custom types	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
-	Citation	[self]	[11]	-	[73]	[44]	[27]	[74]	[34]	[75]	[30]	[21]	[23]
	Code	🔓	🔒	🔓	🔓	🔓	🔓	🔓	🔓	🔒	🔓	🔓	🔓

Expressivity scores: (1a) Pre-trained on equations generated from limited prior. (1b) Basis of fixed expressions, combined in a linear sum. (2) Flexible basis of expressions, with variable internal coefficients. (3) Any scalar tree, with binary and unary operators. (4) Any scalar tree, with custom operators allowed. (5) Any scalar tree, with n-ary operators. (6) Scalar/vector/tensor expressions of any arity.

∗ Note that the "Symbolic Distillation" method from [23] is not an algorithm itself; it can be applied to any SR technique. Applying this general method to a specific technique will inherit a ✓ from the Symbolic Distillation column, if given. However, in general, this technique is easiest with those methods which have deep learning export.

∗I Only the symmetry discovery module is GPU-capable.

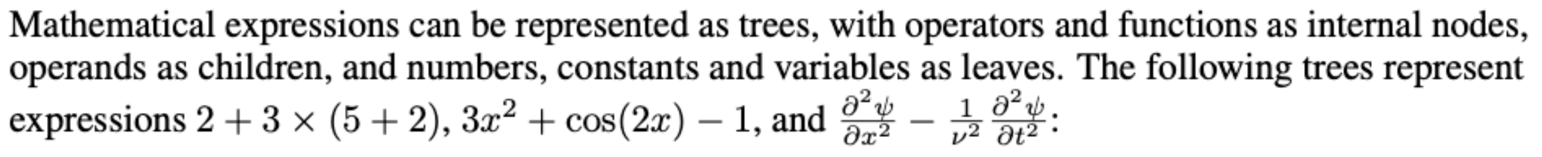
∗II Conceptually different, as is a linear basis of static nonlinear expressions.

∗III Is itself a neural network.

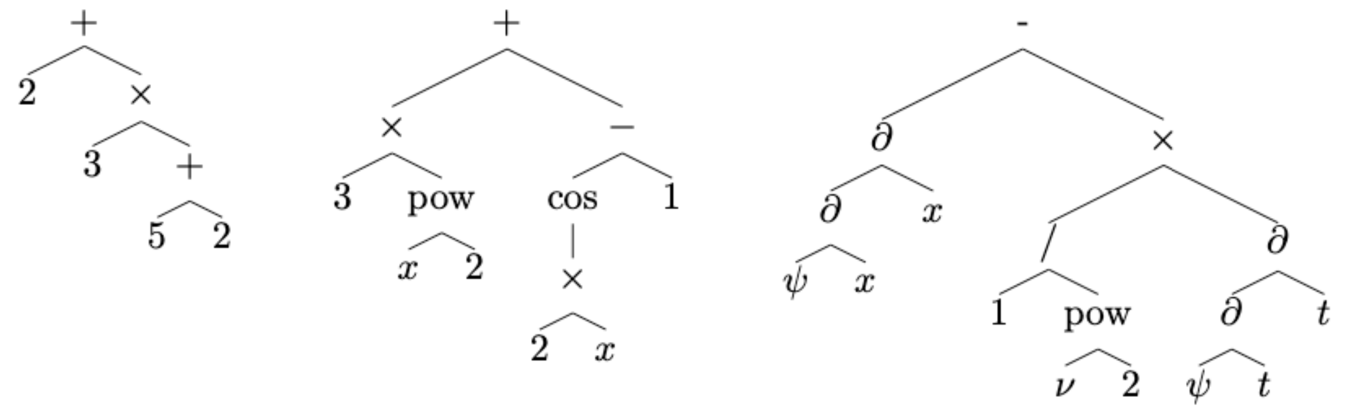
Symbolic math with neural networks

- https://arxiv.org/abs/1912.01412 Deep Learning for Symbolic Mathematics

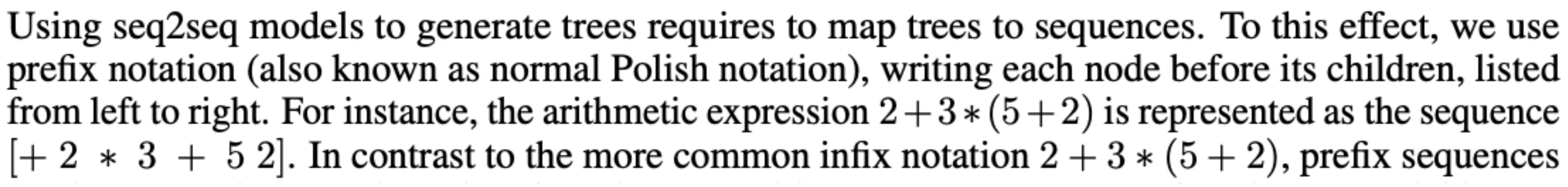
- We use **sequence-to-sequence models (e.g. transformer)** on two problems of symbolic mathematics: function integration and ordinary differential equations (ODEs)
- Representing mathematical expressions as trees



Mathematical expressions can be represented as trees, with operators and functions as internal nodes, operands as children, and numbers, constants and variables as leaves. The following trees represent expressions $2 + 3 \times (5 + 2)$, $3x^2 + \cos(2x) - 1$, and $\frac{\partial^2 \psi}{\partial x^2} - \frac{1}{\nu^2} \frac{\partial^2 \psi}{\partial t^2}$:



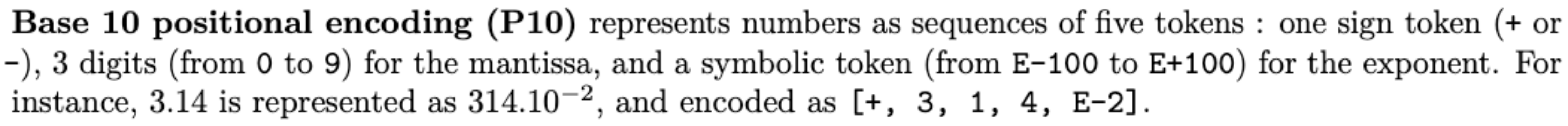
- The tree can then be represented as a sequence of tokens.



Using seq2seq models to generate trees requires to map trees to sequences. To this effect, we use prefix notation (also known as normal Polish notation), writing each node before its children, listed from left to right. For instance, the arithmetic expression $2+3*(5+2)$ is represented as the sequence $[+\ 2\ *\ 3\ +\ 5\ 2]$. In contrast to the more common infix notation $2 + 3 * (5 + 2)$, prefix sequences

Symbolic math with neural networks

- Real numbers can also be discretized as tokens, for example as follows (2112.01898):



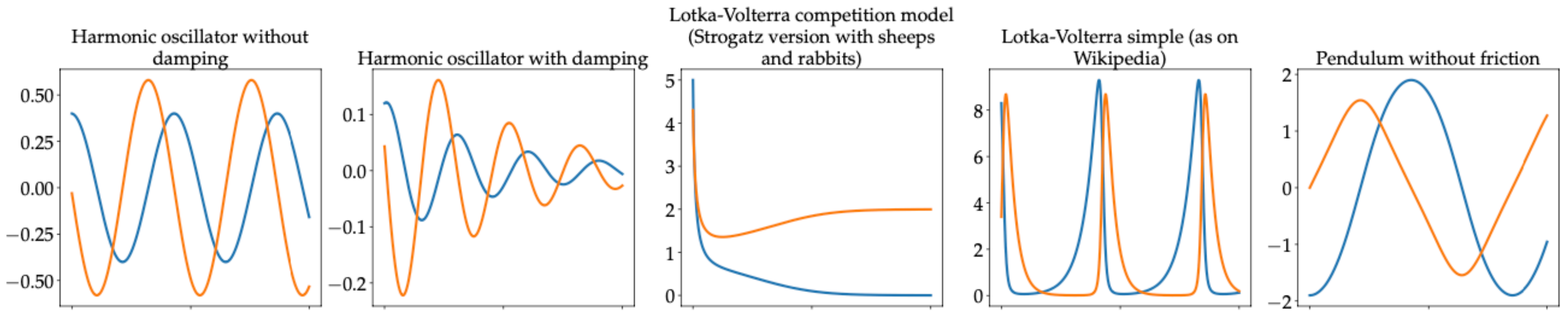
Base 10 positional encoding (P10) represents numbers as sequences of five tokens : one sign token (`+` or `-`), 3 digits (from `0` to `9`) for the mantissa, and a symbolic token (from `E-100` to `E+100`) for the exponent. For instance, 3.14 is represented as 314.10^{-2}, and encoded as `[+, 3, 1, 4, E-2]`.

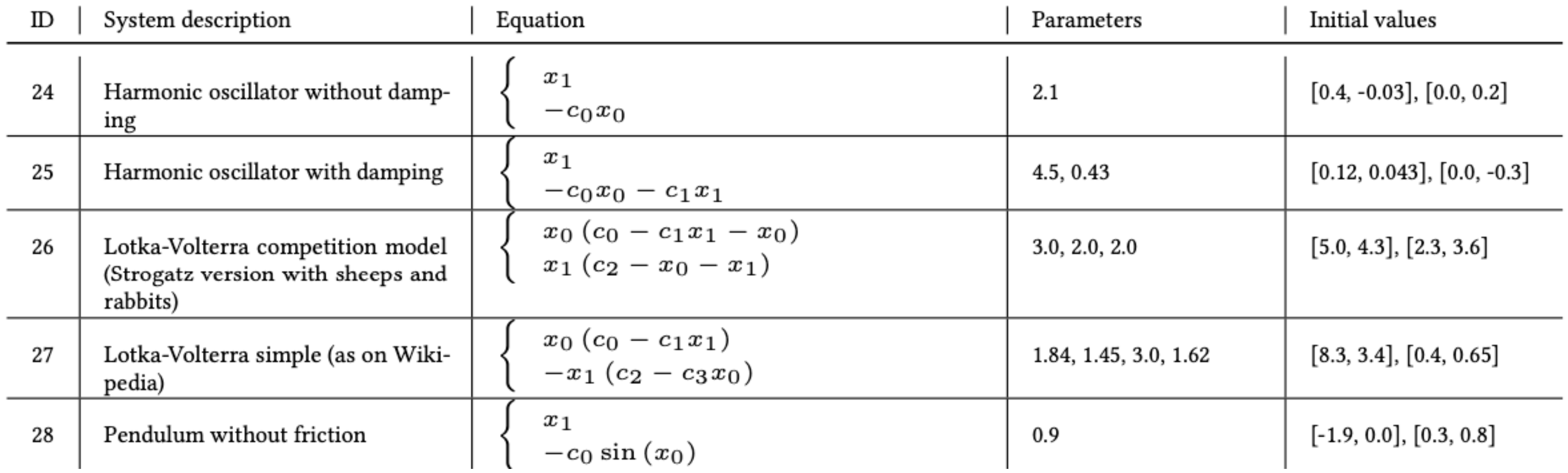
- Next we need to generate a large training set, e.g. to learn integration. This can be done by randomly generating expressions, and then using conventional methods (such as Mathematica) to find solutions.
  - Random expressions can be generated forward or backward (e.g. differentiation is easier than integration).
- The model in the paper 1912.01412 outperforms Mathematica at integration and ODE solution finding in the range where it was trained (note however that this is a rather restricted range, see e.g. 1912.05752).
- The strategy of tokenizing equations has been widely adopted.

ODEFormer

- Can we give a neural network observed data (solutions to differential equations) and ask it to find the underlying equation of motion (differential equations)?
- We are currently interested in this topic in my group so let me show you a recent paper by other authors in this direction: https://arxiv.org/abs/2310.05573 **ODEFormer: Symbolic Regression of Dynamical Systems with Transformers**
- Example: 2-dimensional ODE

$$\dot{x} = f(x)$$

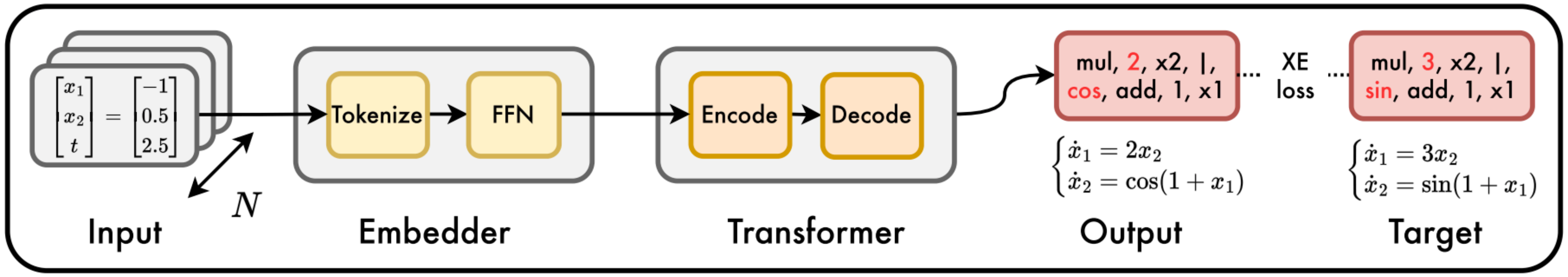




ID	System description	Equation	Parameters	Initial values
24	Harmonic oscillator without damping	$\begin{cases} x_1 \\ -c_0 x_0 \end{cases}$	2.1	[0.4, -0.03], [0.0, 0.2]
25	Harmonic oscillator with damping	$\begin{cases} x_1 \\ -c_0 x_0 - c_1 x_1 \end{cases}$	4.5, 0.43	[0.12, 0.043], [0.0, -0.3]
26	Lotka-Volterra competition model (Strogatz version with sheeps and rabbits)	$\begin{cases} x_0 (c_0 - c_1 x_1 - x_0) \\ x_1 (c_2 - x_0 - x_1) \end{cases}$	3.0, 2.0, 2.0	[5.0, 4.3], [2.3, 3.6]
27	Lotka-Volterra simple (as on Wikipedia)	$\begin{cases} x_0 (c_0 - c_1 x_1) \\ -x_1 (c_2 - c_3 x_0) \end{cases}$	1.84, 1.45, 3.0, 1.62	[8.3, 3.4], [0.4, 0.65]
28	Pendulum without friction	$\begin{cases} x_1 \\ -c_0 \sin(x_0) \end{cases}$	0.9	[-1.9, 0.0], [0.3, 0.8]

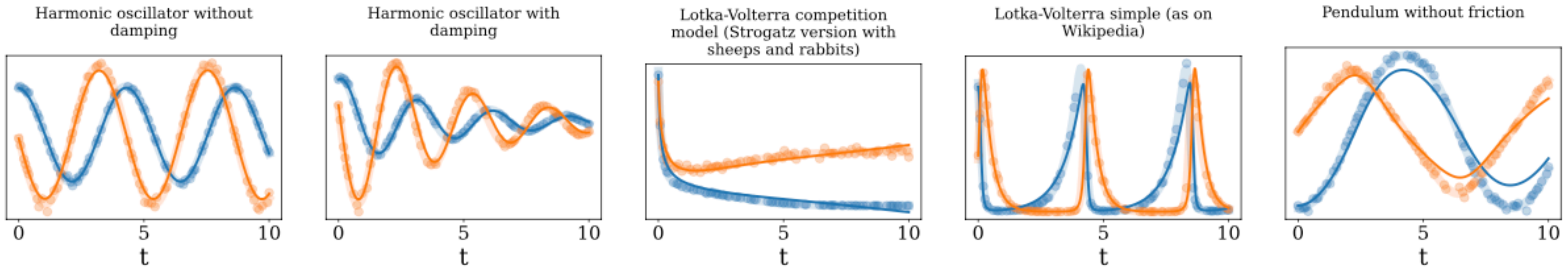
ODEFormer

- Use ideas we discussed: Tokenizing equations, transformer encoder-decoder



$\begin{bmatrix} x_1 \\ x_2 \\ t \end{bmatrix} = \begin{bmatrix} -1 \\ 0.5 \\ 2.5 \end{bmatrix}$
N
Input
Tokenize
FFN
Embedder
Encode
Decode
Transformer
mul, 2, x2, |, cos, add, 1, x1
$\dot{x}_1 = 2x_2$, $\dot{x}_2 = \cos(1+x_1)$
Output
XE loss
mul, 3, x2, |, sin, add, 1, x1
$\dot{x}_1 = 3x_2$, $\dot{x}_2 = \sin(1+x_1)$
Target

Figure 2: **Sketch of our method to train ODEFormer**. We generate random ODE systems, integrate a solution trajectory on a grid of N points $x \in \mathbb{R}^D$, and train ODEFormer to directly output the ODE system in symbolic form, supervising the predicted expression via cross-entropy loss.

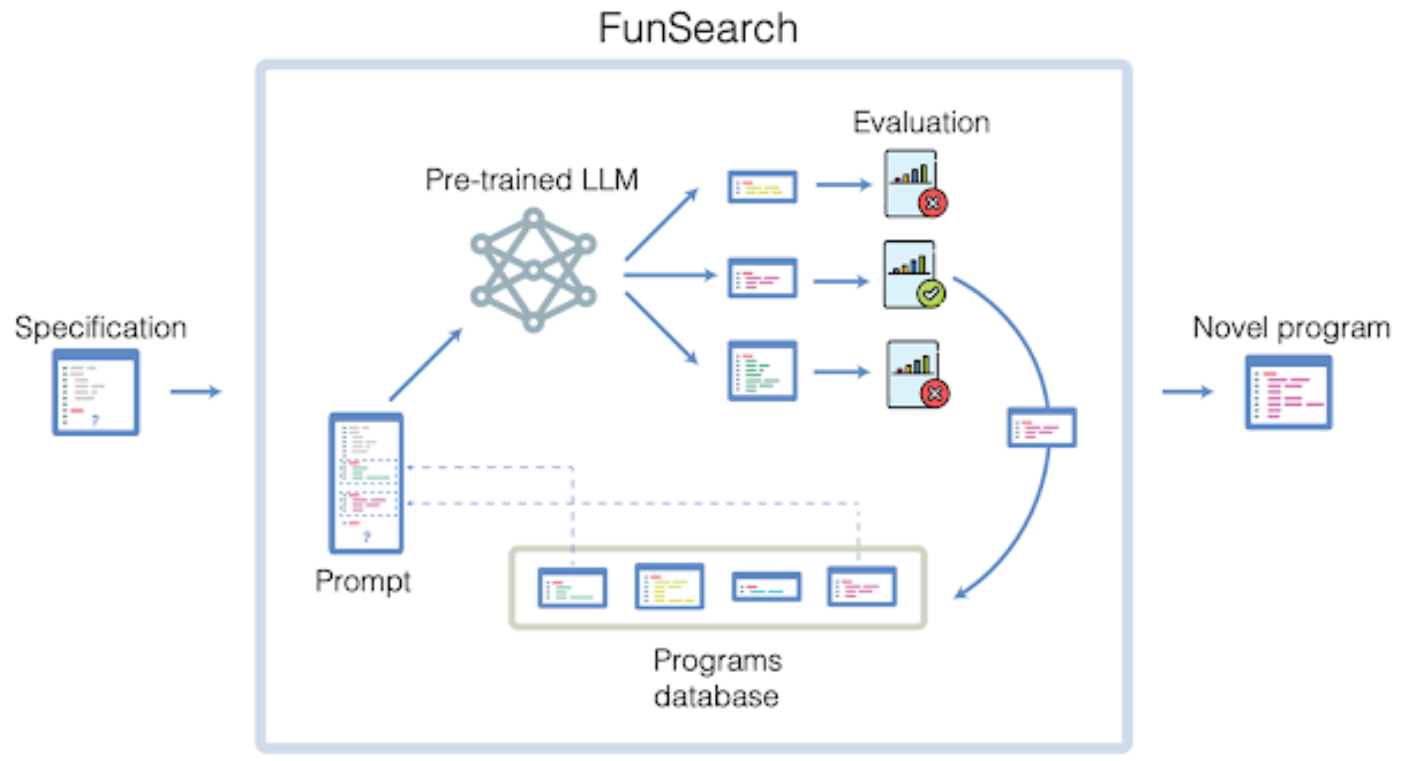


- Could we discover the equations of motion of an observed system for which we don't know the answer?

Theorem proving with LLMs

- https://arxiv.org/abs/2009.03393 Generative Language Modeling for Automated Theorem Proving
    - Come up with proposals for proofs (sequence of mathematical operations) that can then be checked .
- https://deepmind.google/discover/blog/funsearch-making-new-discoveries-in-mathematical-sciences-using-large-language-models/ FunSearch: Making new discoveries in mathematical sciences using Large Language Models
    - Searching for programs

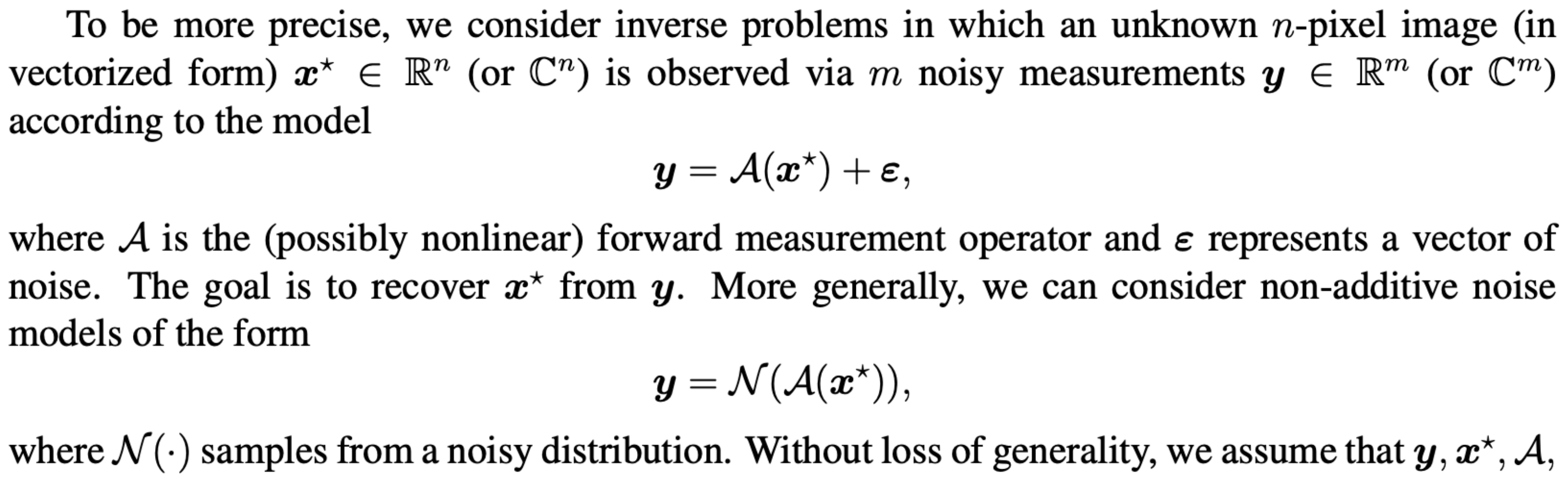
This work represents the first time a new discovery has been made for challenging open problems in science or mathematics using LLMs. FunSearch discovered new solutions for the cap set problem, a longstanding open problem in mathematics.



The FunSearch process. The LLM is shown a selection of the best programs it has generated so far (retrieved from the programs database), and asked to generate an even better one. The programs proposed by the LLM are automatically executed, and evaluated. The best programs are added to the database, for selection in subsequent cycles. The user can at any point retrieve the highest-scoring programs discovered so far.

Inverse problems

- Most data analysis problems in research can be formulated as Inverse Problems, and we have already encountered some in previous lectures.
- Review of inverse problems in the 2d image domain: https://arxiv.org/abs/2005.06001



To be more precise, we consider inverse problems in which an unknown n-pixel image (in vectorized form) $\boldsymbol{x}^\star \in \mathbb{R}^n$ (or $\mathbb{C}^n$) is observed via m noisy measurements $\boldsymbol{y} \in \mathbb{R}^m$ (or $\mathbb{C}^m$) according to the model

$$\boldsymbol{y} = \mathcal{A}(\boldsymbol{x}^\star) + \boldsymbol{\varepsilon},$$

where $\mathcal{A}$ is the (possibly nonlinear) forward measurement operator and $\boldsymbol{\varepsilon}$ represents a vector of noise. The goal is to recover $\boldsymbol{x}^\star$ from $\boldsymbol{y}$. More generally, we can consider non-additive noise models of the form

$$\boldsymbol{y} = \mathcal{N}(\mathcal{A}(\boldsymbol{x}^\star)),$$

where $\mathcal{N}(\cdot)$ samples from a noisy distribution. Without loss of generality, we assume that $\boldsymbol{y}, \boldsymbol{x}^\star, \mathcal{A},$

- There are many different machine learning methods to solve such problems.
  - A naive one is to simply train a Neural Network to map from y to x, on simulated training data pairs.
  - A different method is to define a likelihood for the problem, with a generative prior (score matching, normalizing flows etc) and find the posterior of x given y. This results in a probabilistic solution (i.e. with error bars).
  - There are also specialized algorithms, for example 1706.04008 (“Recurrent Inference Machines for Solving Inverse Problems”)

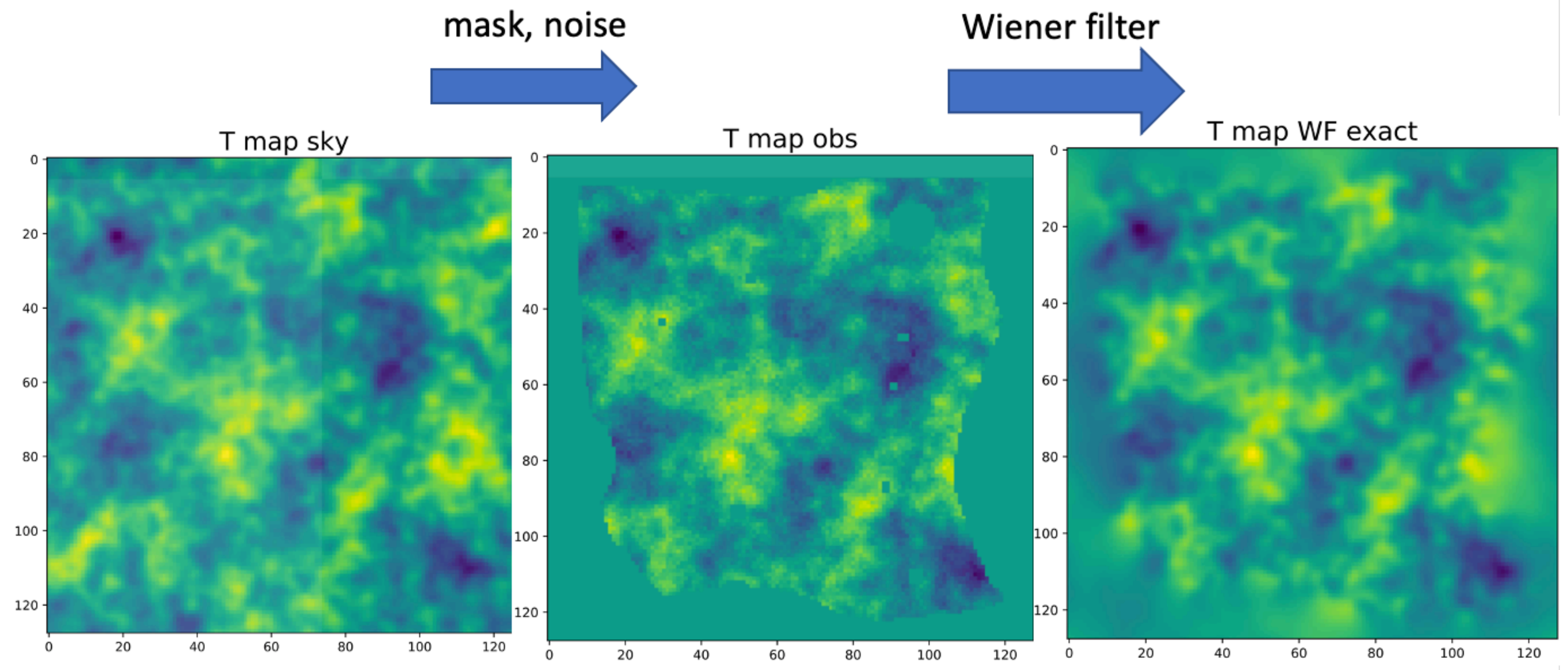
Example of solving a Inverse Problem from my research: Wiener filtering

https://arxiv.org/abs/1905.05846

Learning a mathematical operator

- Sometimes it is possible to design a neural network that is specifically designed to enforce a mathematical relation
- In cosmology, one often wants to first reconstruct the data from a noisy operation using a linear operation called “Wiener filtering”, which is solving a linear inverse problem.
- The problem is that Wiener filtering is too computationally expensive for large data sets.
- We wanted to know if this task can be done better with a Neural network, but under several constraints:
    - We did not want to use Wiener-filtered training data. Instead we train on the likelihood.
    - We wanted to enforce the property that the Filtering is linear, but with a filter that depends non-linearity on the noise.
- I want to show you here that this can be achieved by a non-trivial neural network architecture.

Example from my research: Wiener filtering



true CMB

measurement

Best possible reconstruction of the true CMB

Very important method! First step for any optimal statistical analysis.

Wiener filtering

- **Common situation:**

$$\underset{\substack{\uparrow\\\text{data}}}{d} = \underset{\substack{\uparrow\\\text{signal}}}{s} + \underset{\substack{\uparrow\\\text{noise}}}{n}$$

- **Wiener filter:**

$$d_{WF} = \underbrace{S(S}_{\substack{\uparrow\ \uparrow\\\text{Signal covariance matrix.}}} + \underset{\substack{\uparrow\\\text{Noise covariance matrix.}}}{N})^{-1}d$$

- **Optimal reconstruction** of s given d.
- Data d can have 10^8 elements. Direct matrix inversion impossible.
- Standard approach: **conjugate gradient method**. But too slow! **Most Planck CMB analysis is suboptimal for this reason.**

$\Rightarrow$ **Neural network approach**

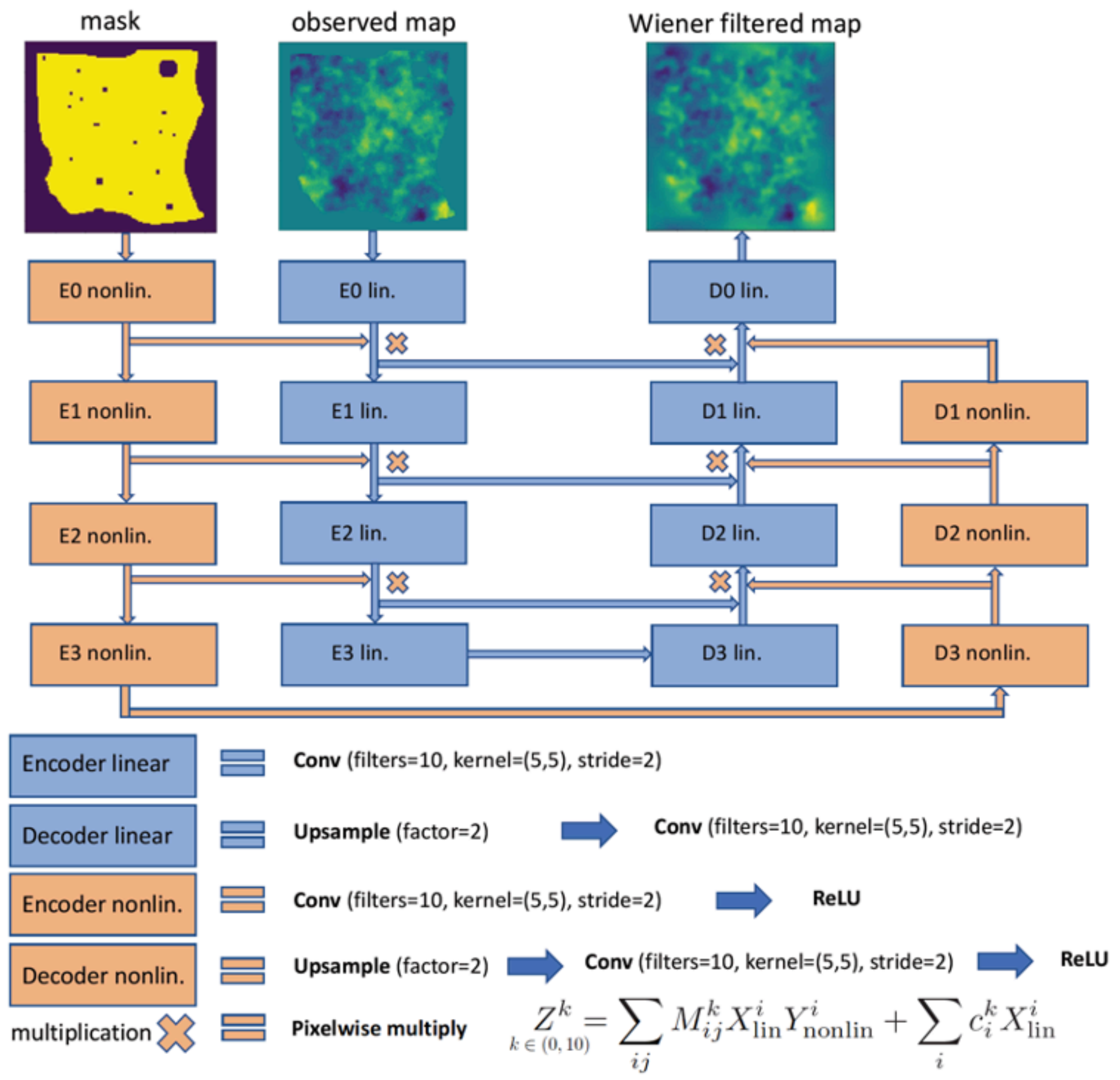
New neural network architecture

- Crucial: **must not induce non-linearities.**
- Construct a neural network that is **explicitly linear in the data!**

$$y = M(\text{mask})d$$

- **Nonlinear in mask/noise**

Machine learning does not need to be based on "generic functions"!


mask
observed map
Wiener filtered map
E0 nonlin.
E0 lin.
D0 lin.
E1 nonlin.
E1 lin.
D1 lin.
D1 nonlin.
E2 nonlin.
E2 lin.
D2 lin.
D2 nonlin.
E3 nonlin.
E3 lin.
D3 lin.
D3 nonlin.
Encoder linear = Conv (filters=10, kernel=(5,5), stride=2)
Decoder linear = Upsample (factor=2) → Conv (filters=10, kernel=(5,5), stride=2)
Encoder nonlin. = Conv (filters=10, kernel=(5,5), stride=2) → ReLU
Decoder nonlin. = Upsample (factor=2) → Conv (filters=10, kernel=(5,5), stride=2) → ReLU
multiplication = Pixelwise multiply $Z^k_{k\in(0,10)} = \sum_{ij} M^k_{ij} X^i_{\text{lin}} Y^i_{\text{nonlin}} + \sum_i c^k_i X^i_{\text{lin}}$

MM et. al., 1905.05846, NeurIPS 2019

Loss functions and training

- **3 possible loss functions** (training objectives) with very different properties:

"naïve loss" $$J_1(d,y) = \frac{1}{2}(y - y_{\rm WF})^T A (y - y_{\rm WF})$$ Not useful in practice.

"supervised loss" $$J_2(s,y) = \frac{1}{2}(y - s)^T A (y - s)$$ Works well in S/N>1 regime.

"physical loss" $$J_3(d,y) = \frac{1}{2}(y - d)^T N^{-1} (y - d) + \frac{1}{2} y^T S^{-1} y$$ Works well everywhere.

$$J_3(d,y) = -\log P(s|d)_{s=y} + \text{const.}$$

- All can be analytically shown to be minimized by WF solution, i.e.

$$\frac{\partial \langle J \rangle}{\partial M} \overset{!}{=} 0 \quad \Longrightarrow \quad M = S(S+N)^{-1}$$

Neural networks can be used in low signal-to-noise situations! ✓

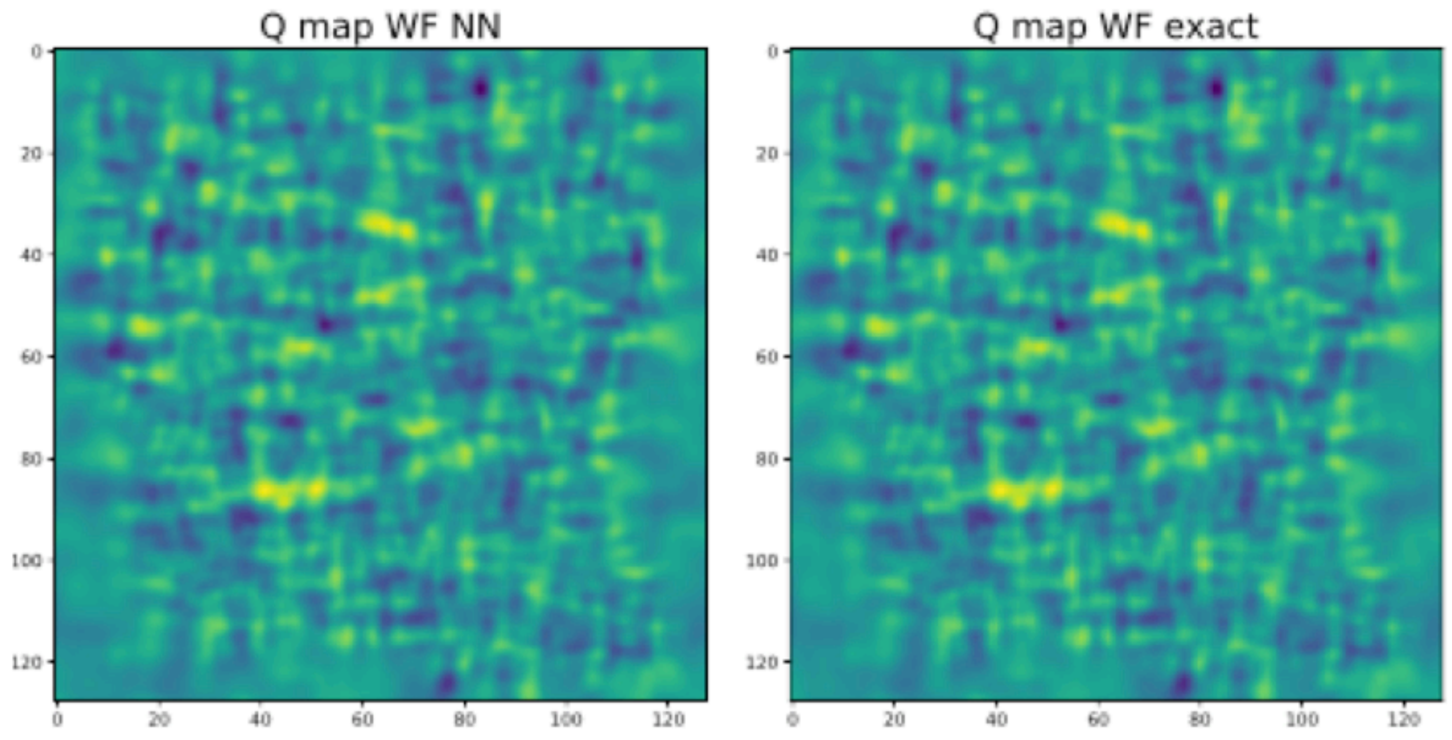
Results

Neural network output maps are at least 99% Wiener filtered.

Neural Network Wiener filtering is **1000 times** faster than the exact method!

- Works independent of mask and noise levels.
- Plug into standard analysis pipelines in cosmology.

CMB polarization example:


Q map WF NN
Q map WF exact