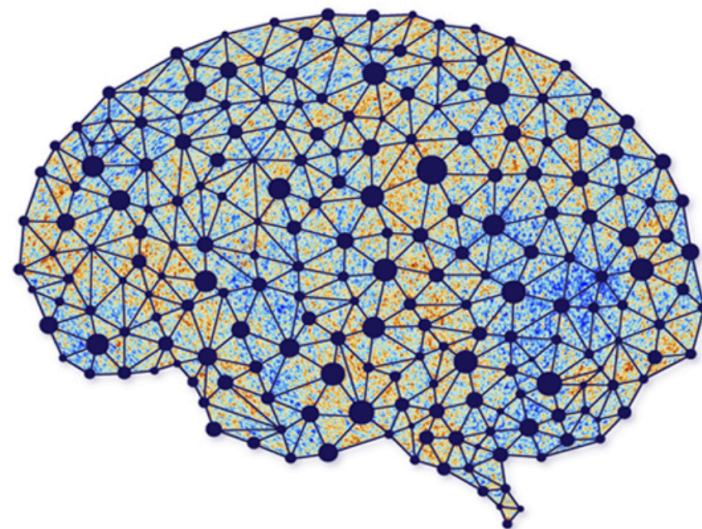


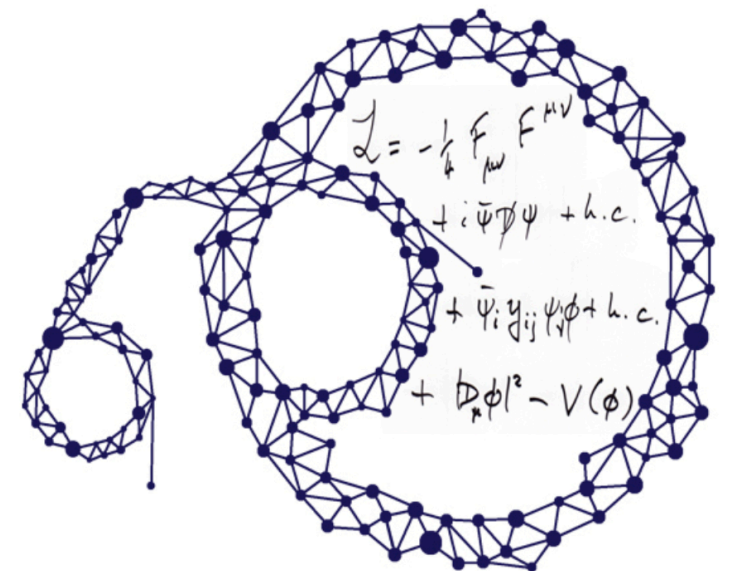
Physics 361 - Machine Learning in Physics

Lecture 24 – PDE solving, Inverse problems, Anomaly detection

April 22nd 2024



AI
∩
Universe

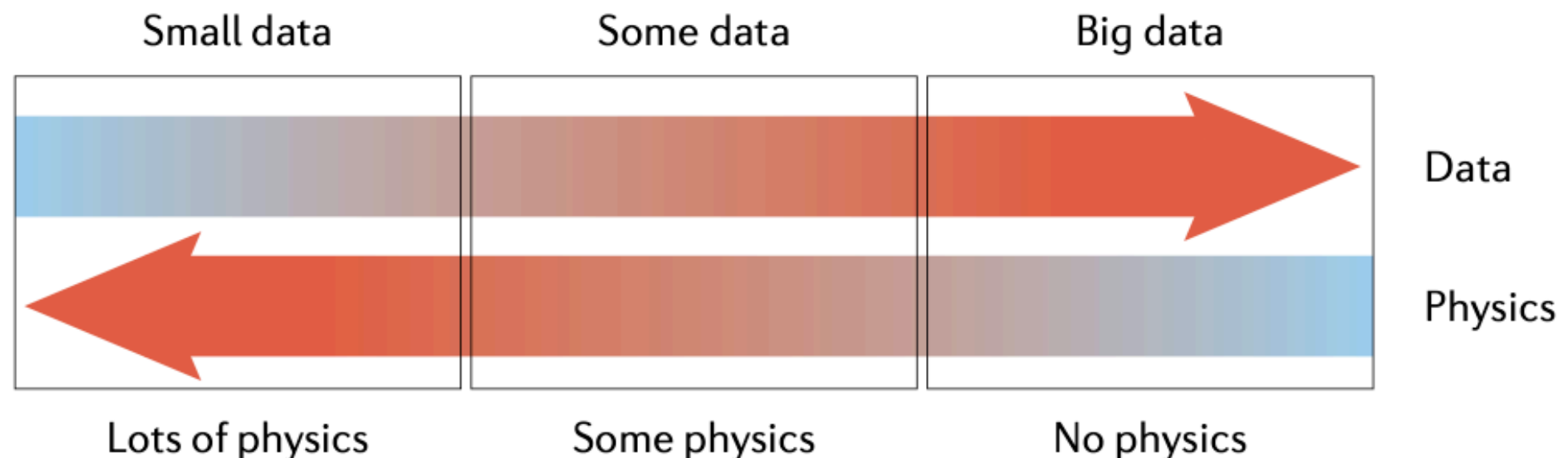


Moritz Münchmeyer

PDE solving

“Physics-informed machine learning”

- Review: <https://www.nature.com/articles/s42254-021-00314-5> (plots from this source)
 - “Despite great progress in simulating multiphysics problems using the numerical discretization of partial differential equations (PDEs), one still cannot seamlessly incorporate noisy data into existing algorithms, mesh generation remains complex, and high-dimensional problems governed by parameterized PDEs cannot be tackled. “
- 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

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

$u(x, t)$ is the solution

- 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

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

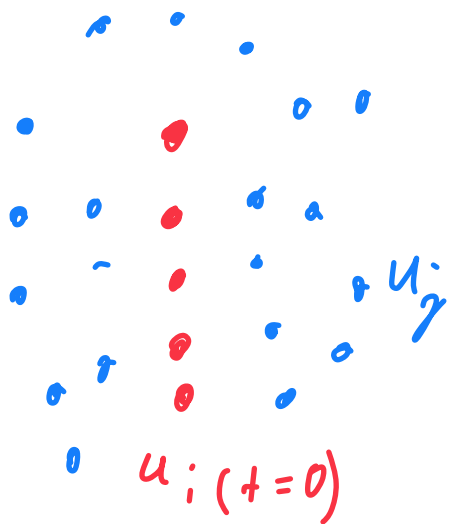
where

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

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

enforce boundary condition

enforce the PDE

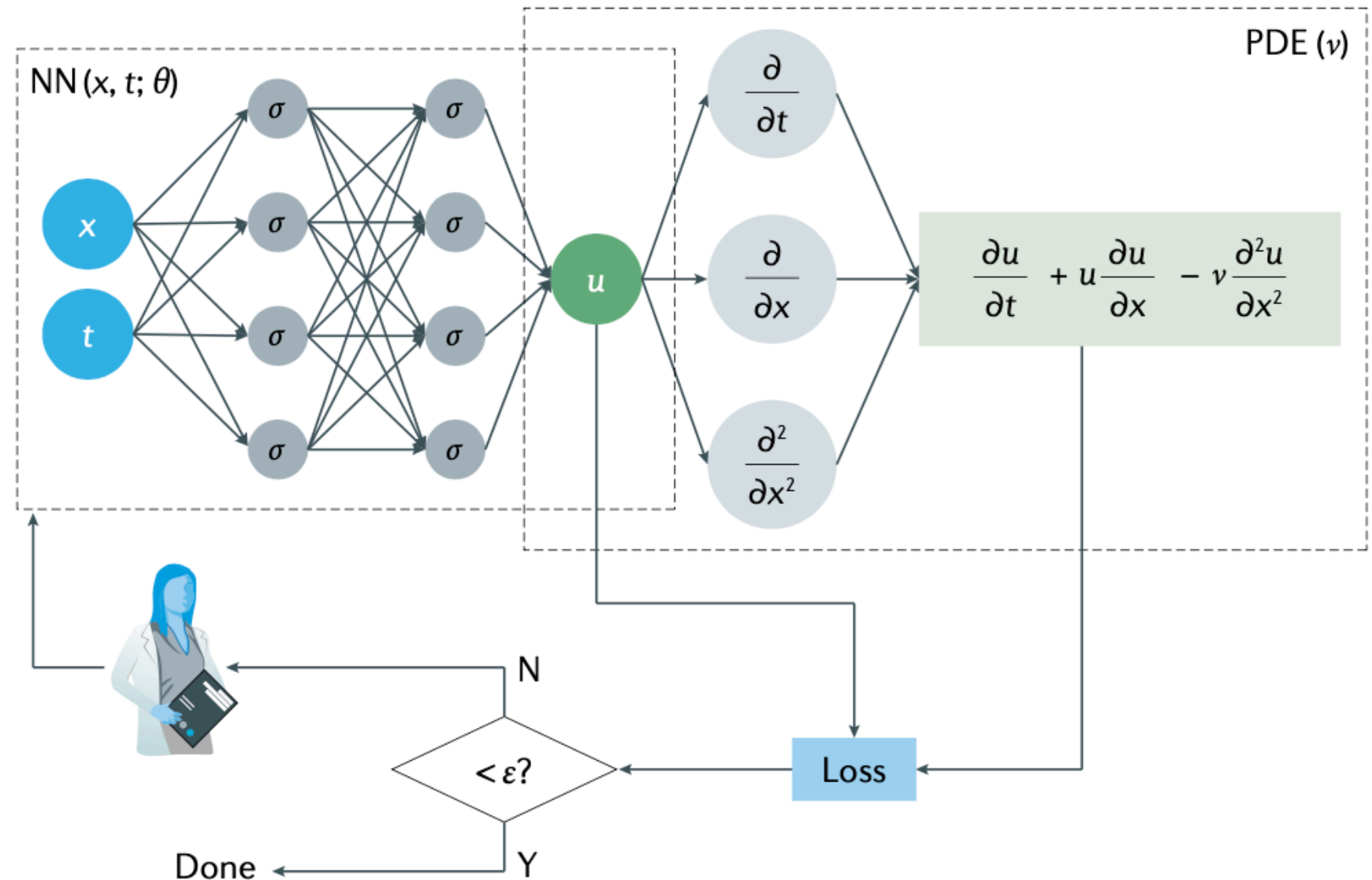


Here $\{(x_i, t_i)\}$ and $\{(x_j, t_j)\}$ are two sets of points sampled at the initial/boundary locations and in the entire domain, respectively, and u_i are values of u at (x_i, t_i) ; w_{data} and w_{PDE} are the weights used to balance the interplay between the two loss terms. These weights can 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.

We want to represent the solution $u(x,t)$ by a neural network which takes x and t as input and outputs $u(x,t)$.



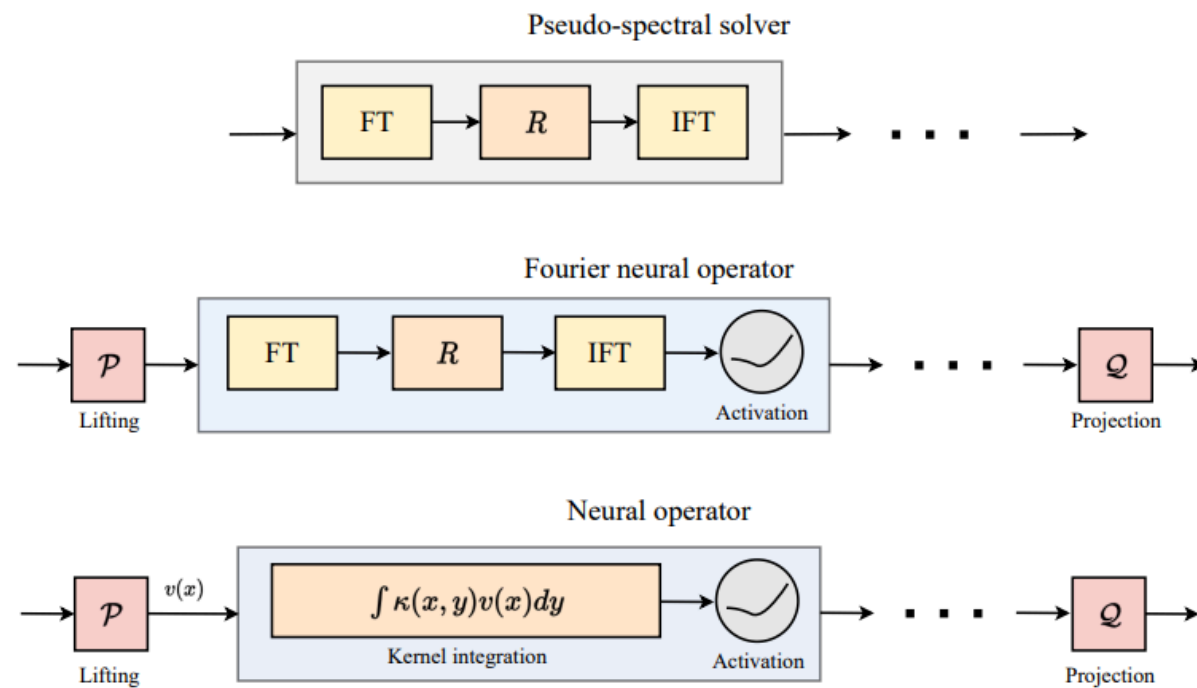
Algorithm 1: The PINN algorithm.

Construct a neural network (NN) $u(x, t; \theta)$ with θ 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 θ^* 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.

Neural Operators



R is a learned operator in Fourier space (e.g., a neural network modifying Fourier coefficients).

Lifting Operator \mathcal{P} :

Purpose: Transforms the input function $u(x)$ (which might be scalar- or low-dimensional valued) into a **higher-dimensional representation** or feature space where richer patterns and dependencies can be learned.

Projection Operator \mathcal{Q} :

Purpose: Maps the high-dimensional output features $\mathbf{v}(x)$ back down to the **target output space** (e.g., scalar field, vector field).

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

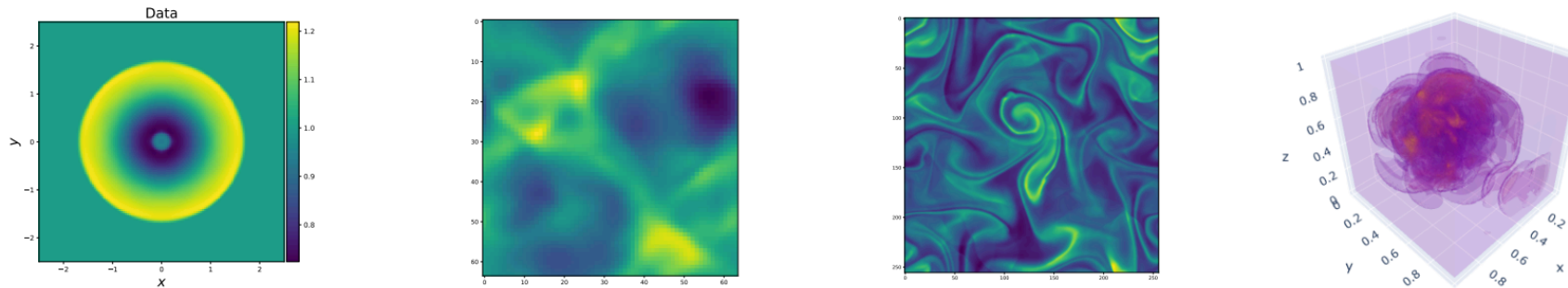


Figure 1: PDEBENCH provides multiple non-trivial challenges from the Sciences to benchmark current and future ML methods, including wave propagation and turbulent flow in 2D and 3D

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.

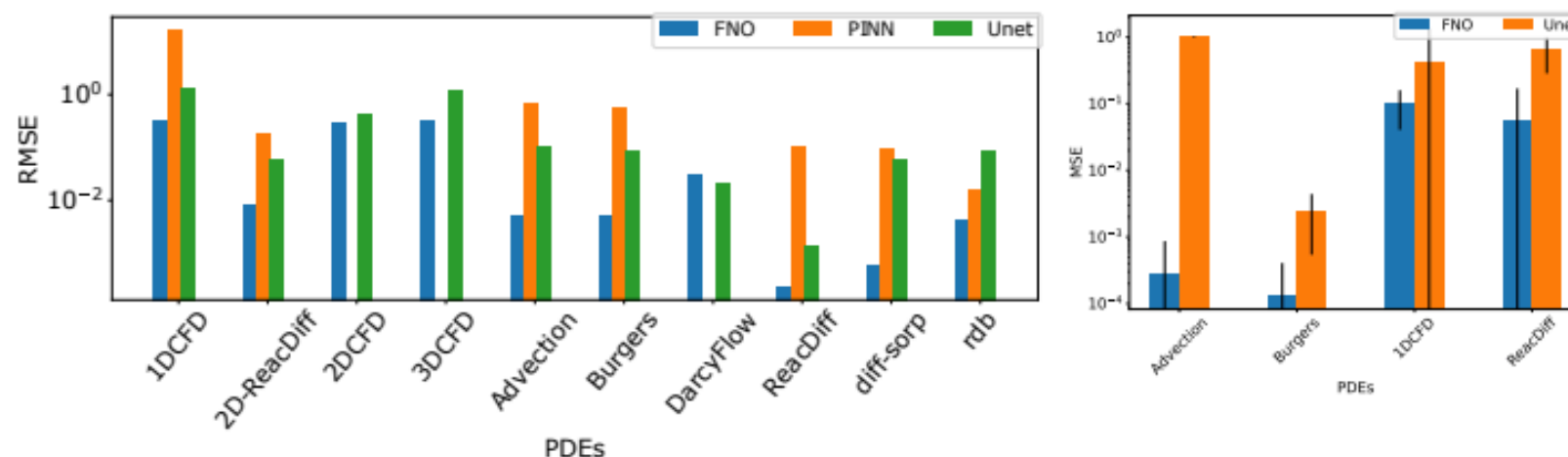
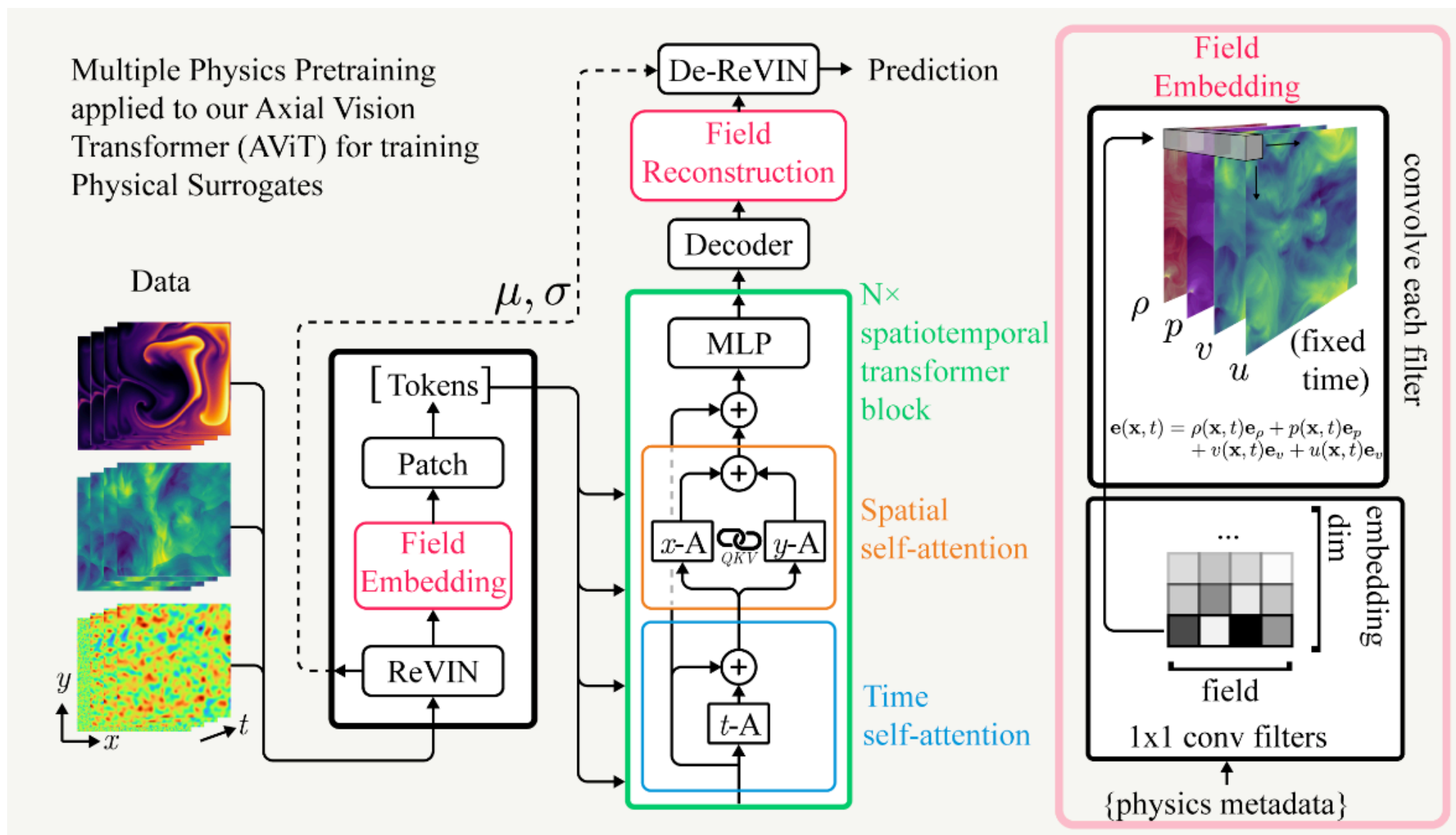


Figure 2: Comparisons of baseline models' performance for different problems for (a) the forward problem and (b) the inverse problem.

- 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.



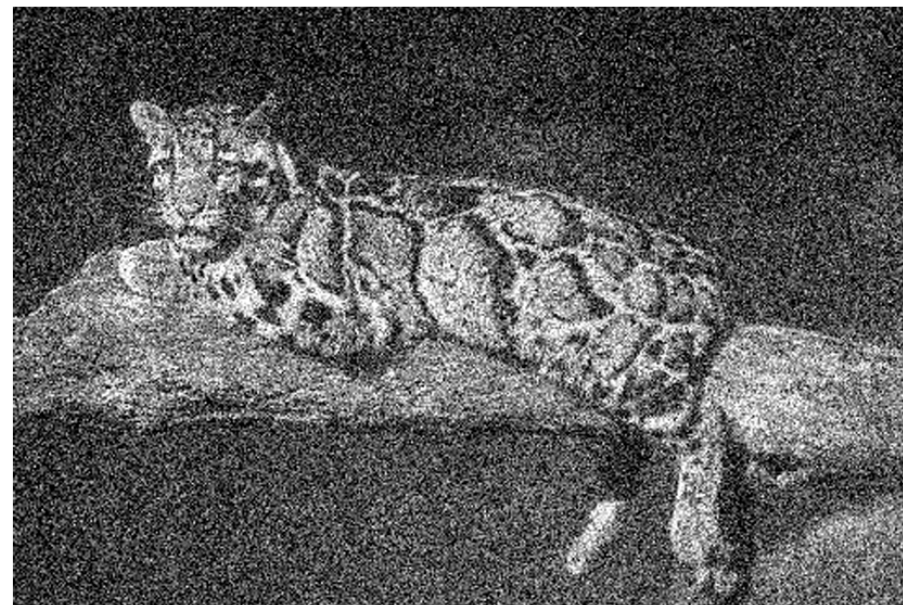
Inverse problems

Review of inverse problems in the 2d image domain: <https://arxiv.org/abs/2005.06001>

(a) Ground truth



(b) Noisy image, 14.88dB



(c) EPLL, 25.68dB



(d) RIM, 25.91dB

Figure 5: *Denoising performance on example image used in Zoran and Weiss [5]. $\sigma = 50$. Noisy image was 8-bit quantized before reconstruction.*

Example from <https://arxiv.org/pdf/1706.04008>. c and d are different inverse problem solutions.

Inverse problems

- Inverse problems are **ubiquitous in science and engineering**, involving the recovery of underlying causes (inputs) from observed effects (outputs).
- **Examples** include
 - medical imaging
 - deblurring in vision
 - source reconstruction in physics
- Solving inverse problems is challenging because they are **often ill-posed** (i.e., solutions may not exist, be unique, or depend continuously on the data).
- **Many data analysis problems in research** can be formulated as Inverse Problems, and we have already encountered some in previous lectures.
- **Machine learning offers powerful tools to tackle these problems**, especially when traditional methods struggle.

Mathematical formulation

Formally, an inverse problem seeks to find input x from observations y , given a forward operator \mathcal{F} :

$$y = \mathcal{F}(x) + \epsilon$$

where \mathcal{F} models the physical process, and ϵ is noise. The inverse problem is to find x given y .

- The “**forward operator**” (or “**forward model**”) can be either linear or non-linear. Usually it is not invertible (i.e ill-defined).
- Often one needs to **add a regularizer R** to make the problem well-defined:

Solve using optimization:

$$\hat{x} = \arg \min_x \|\mathcal{F}(x) - y\|^2 + \lambda R(x)$$

- We often also need an **uncertainty or even a full Bayesian answer** (distribution of solutions) to the inverse problem.

Overview of methods

- There are many different machine learning methods to solve such problems, **with and without machine learning**.
- They differ in what forward model they address (e.g. **linear vs non-linear**) and what **assumptions** they make.
- Some approaches require **knowledge of the forward model**, others don't.
- We will look through **some common techniques** without being systematic, focussing on machine learning techniques.

Direct supervised approach

A simple approach: Use a neural network and learn the operation from training data.

- Learn a direct mapping from observations to solutions:

$$x = \mathcal{G}_\theta(y)$$

- Trained on paired data $\{(x_i, y_i)\}$.
- Common architectures: CNNs (e.g., U-Nets), transformers for 3D data.

Pros:

- Fast inference.
- No need for knowledge of \mathcal{F} .

Cons:

- Requires large training datasets.
- Generalization can be limited.

Bayesian Inverse Problems

- Bayesian approaches to inverse problems aim to model uncertainty and provide posterior distributions over solutions instead of single point estimates.

$$p(x | y) \propto p(y | x) p(x)$$

- Often one can assume a Gaussian likelihood defined by the forward model F :

- Given input x and observation y : $p(y | x) = \mathcal{N}(y; \mathcal{F}(x), \sigma^2 I)$

$$y = \mathcal{F}(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

- In this case, we can then **sample from the posterior** using **MCMC**, or methods that work for higher dimensionality such as **Hamiltonian Monte Carlo** or **Variational Inference**.

Example: Reconstructing the initial conditions of the universe

- Let's look at an example where the previous technique is used in cosmology: reconstructing the initial conditions from observed data.

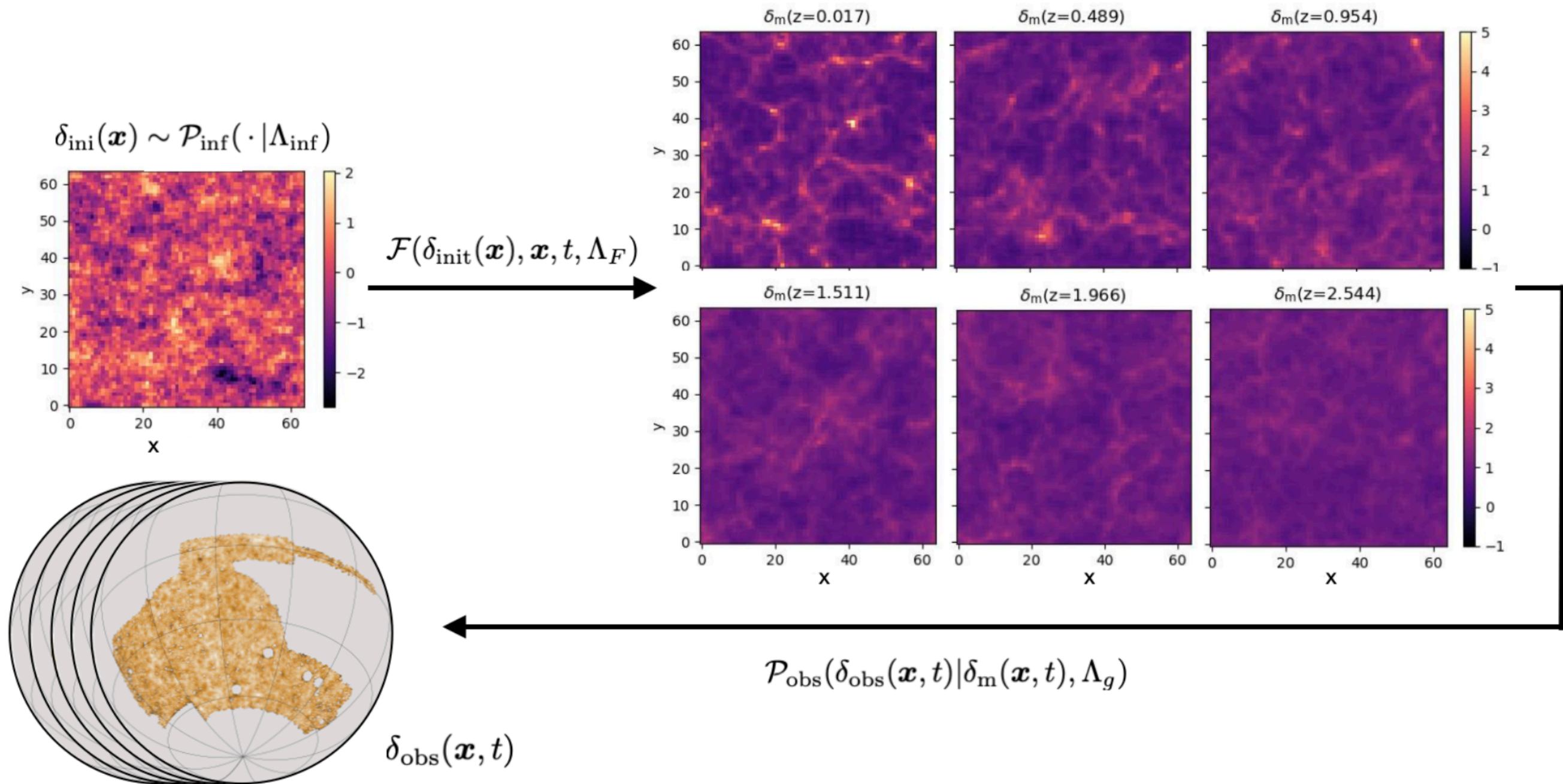
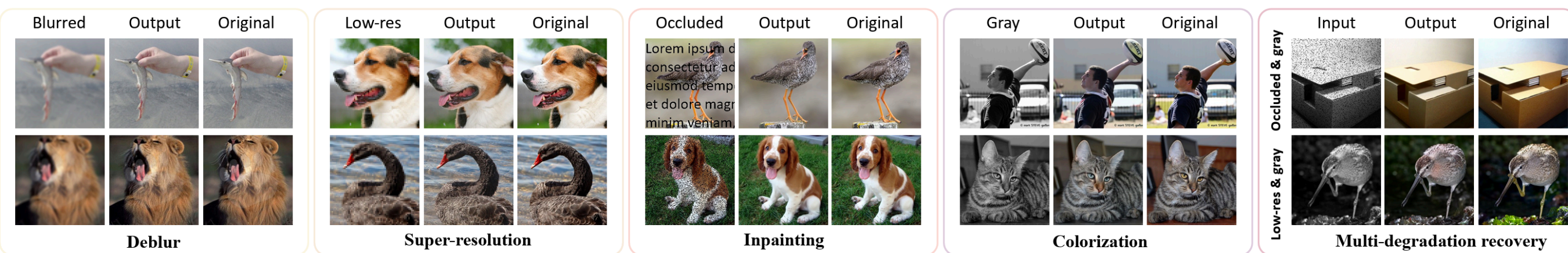


Figure credit: Chihway Chang, U Chicago.

Generative priors

- If we know the likelihood, such as the Gaussian case in the previous slide, we still need to **define a prior** to evaluate the posterior.
- One can use a **Generative Prior**, to improve the reconstruction. If we can learn the prior distribution of the uncorrupted signal x (from simulations thereof), we can use this information to de-noise the corrupted signal.
- We briefly discussed using a normalizing flow as a generative prior in the normalizing flows section.
- **Generative priors can also be learned with diffusion models.**



Examples from: <https://arxiv.org/pdf/2304.01247>

Briefly: Sampling with Diffusion models / Score-Based generative models

- Often we can sample from the forward model, but it is non-differentiable and we do not have a closed form expression for the likelihood.
- An interesting recent technique for such cases is to **use de-noising score-matching to learn the posterior**.
- The idea is to perturb the data with noise and to learn to approximate the posterior by de-noising it.
- Original idea:
 - <https://arxiv.org/abs/2011.13456> Score-Based Generative Modeling through Stochastic Differential Equations
 - <https://arxiv.org/abs/2111.08005> Solving Inverse Problems in Medical Imaging with Score-Based Generative Models
- Application to the cosmology initials conditions problem:
 - <https://arxiv.org/abs/2304.03788> Posterior Sampling of the Initial Conditions of the Universe from Non-linear Large Scale Structures using Score-Based Generative Models

Example: library for inverse problems in imaging

- <https://deepinv.github.io/deepinv/>

DeepInverse: a PyTorch library for imaging with deep learning

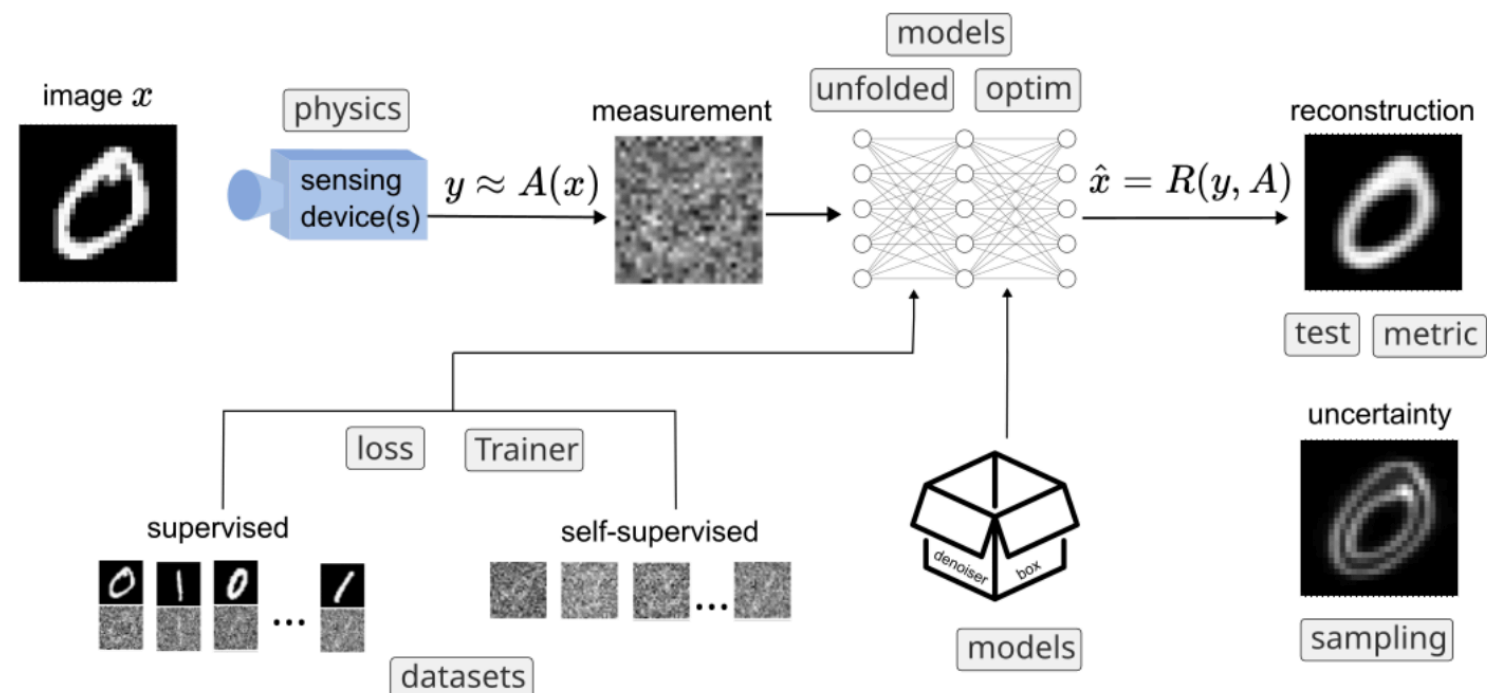
Test passing Build docs passing python 3.9+ code style black codecov 78% deepinv library 160 members Open in Colab

DeepInverse is a PyTorch-based library for solving imaging inverse problems with deep learning.

Github repository: [deepinv/deepinv](https://github.com/deepinv/deepinv).

Featuring

- Large collection of [predefined imaging operators](#) (MRI, CT, deblurring, inpainting, etc.).
- [Training losses](#) for inverse problems (self-supervised learning, regularization, etc.).
- Many [pretrained deep denoisers](#) which can be used for [plug-and-play restoration](#).
- Framework for [building datasets](#) for inverse problems.
- Easy-to-build [unfolded architectures](#) (ADMM, forward-backward, deep equilibrium, etc.).
- [Diffusion algorithms](#) for image restoration and uncertainty quantification (Langevin, diffusion, etc.).
- A large number of well-explained [examples](#), from basics to state-of-the-art methods.



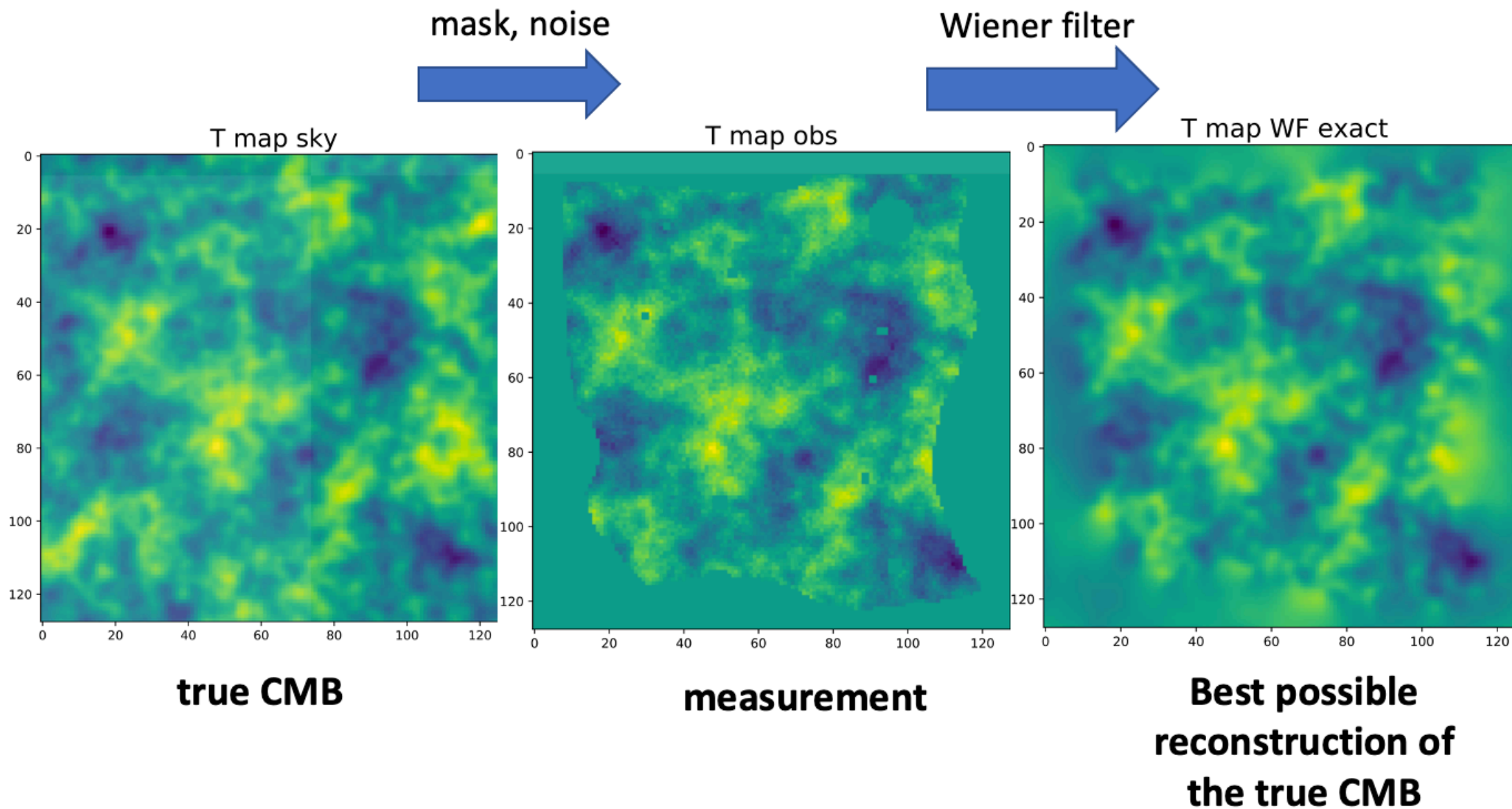
Example of solving a Linear Inverse Problem from my research: Wiener filtering

<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.

Example from my research: Wiener filtering



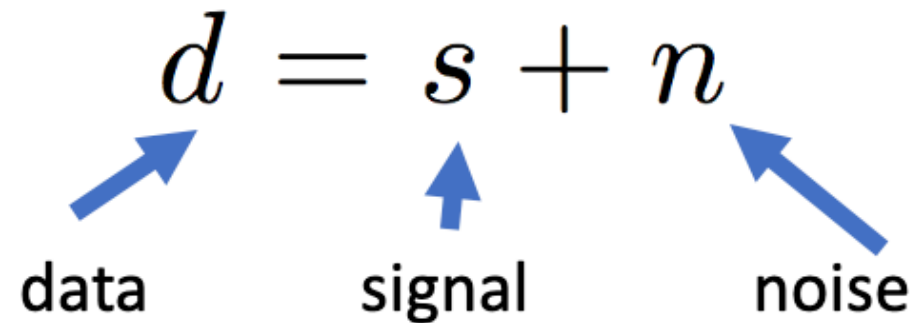
Very important method! First step for any optimal statistical analysis.

Wiener filtering

- **Common situation:**

$$d = s + n$$

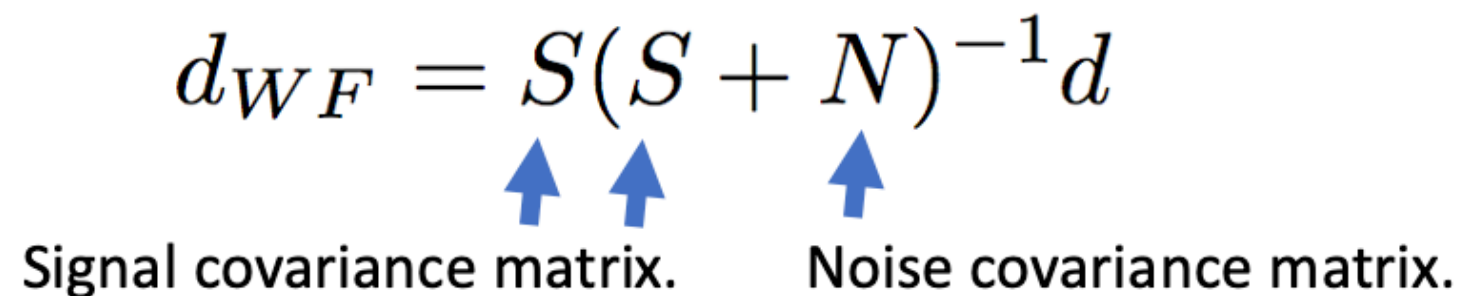
data signal noise



- **Wiener filter:**

$$d_{WF} = S(S + N)^{-1}d$$

Signal covariance matrix. Noise covariance matrix.



- **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.**



Neural network approach

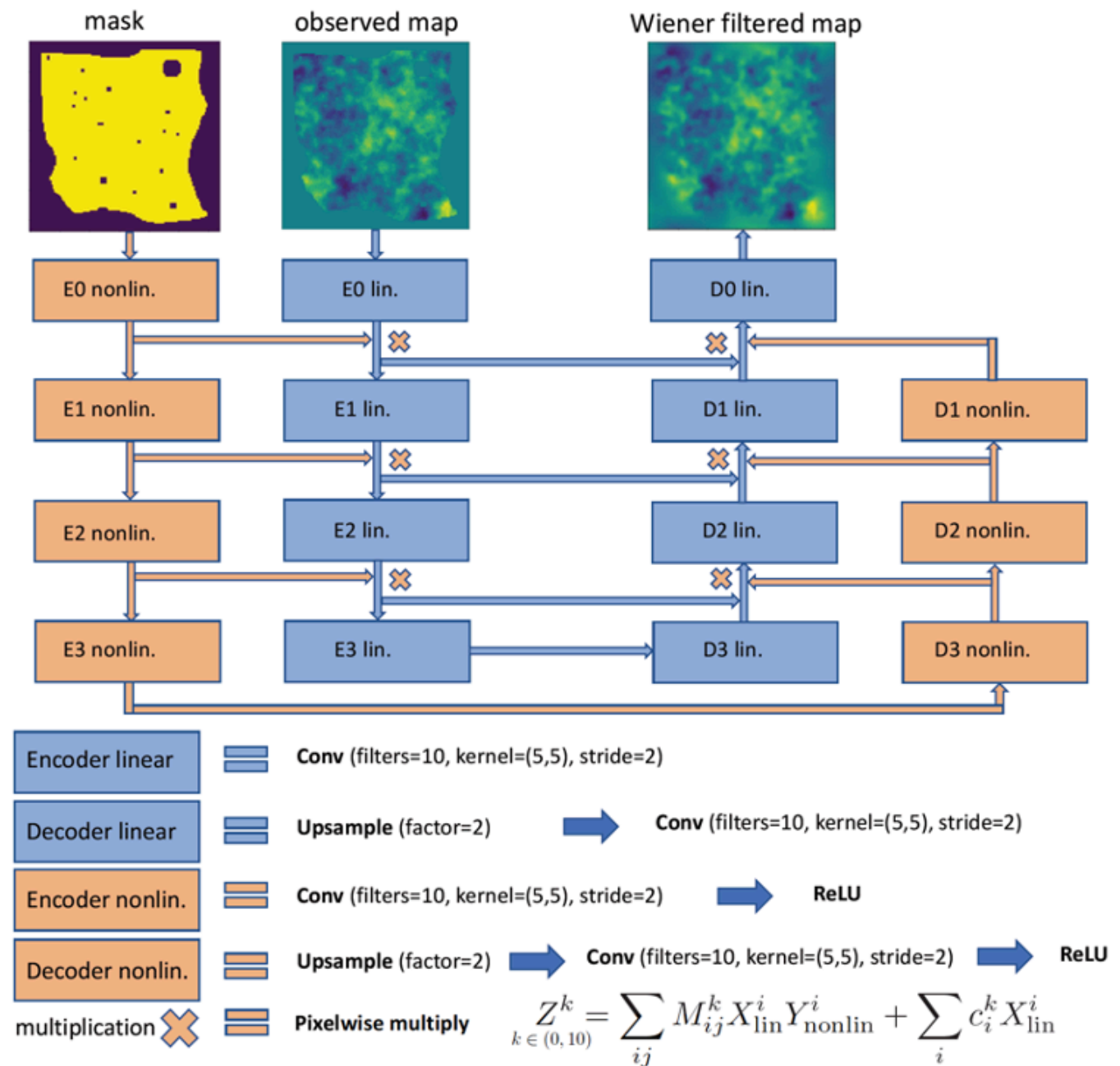
New neural network architecture

- 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”!



Loss functions and training

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

“naïve loss” $J_1(d, y) = \frac{1}{2}(y - y_{\text{WF}})^T A(y - y_{\text{WF}})$ 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.

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

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



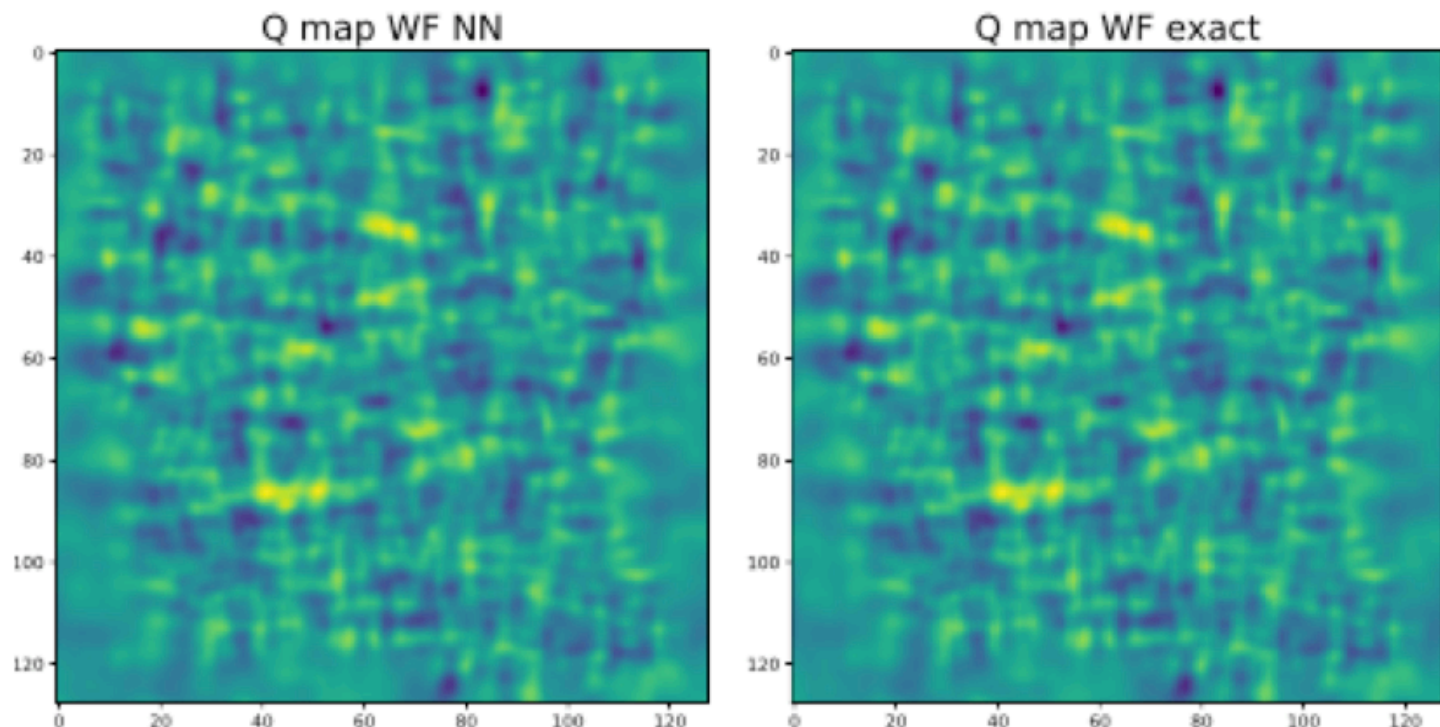
Results

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:



Anomaly detection

Anomaly detection (AD)

- Anomaly detection is the task of identifying rare or unusual observations in data that do not conform to expected patterns.
- **Outside of physics examples:**
 - fraud detection
 - network security
 - fault detection.
- **Physics examples:**
 - Large Hadron Collider (LHC) generates colossal amounts of data. How do we find something unusual without knowing what to look for?
 - In astrophysics, for example Fast Radio Bursts were discovered by accident long after the data was taken.

Unsupervised AD

- **Clustering-Based**

- Use clustering algorithms like k-means, DBSCAN.
- Points far from any cluster center or in small clusters are anomalies.



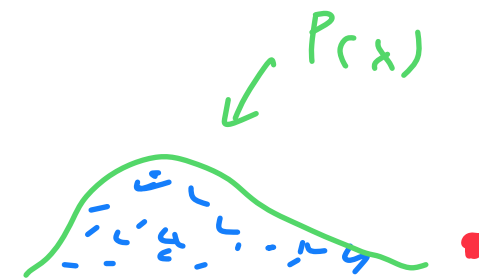
- **Distance-Based**

- Compute distances to nearest neighbors (e.g., k-NN).
- Points with high average distance are anomalies.



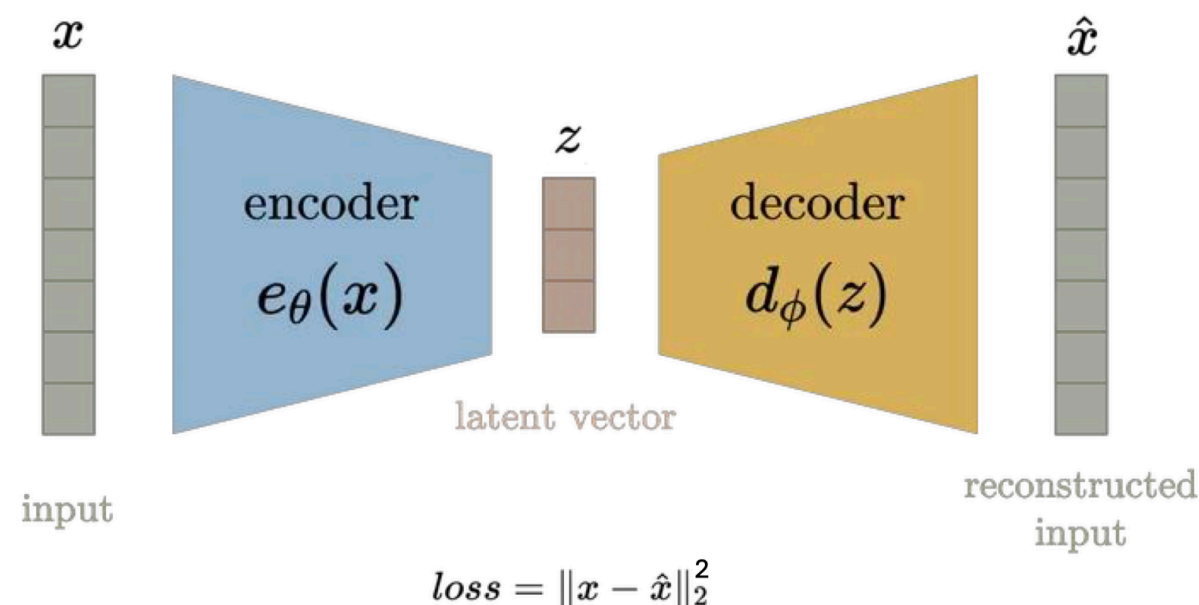
- **Density-Based**

- Use density estimation to find regions of high probability.
- Low-density points are considered anomalies.



AD with Dimensionality Reduction

- Dimensionality reduction techniques, such as Principal Component Analysis (PCA) and autoencoders, can be employed for unsupervised anomaly detection in high-dimensional data.
- Anomalies often exhibit high reconstruction errors when projected back to the original space.
- Autoencoders (Neural Networks)
 - Train a neural network to reconstruct inputs.
 - High reconstruction error at test time signals an anomaly.
 - Variants: Variational autoencoders, LSTM autoencoders for time series.



AD with density estimation

- Use case: Model the distribution of Standard Model background events in high-dimensional feature space.
- Method: Use a normalizing flow to estimate the likelihood of observed events. Events with low likelihood are potential anomalies.
- E.g.: <https://arxiv.org/abs/2001.04990> Anomaly Detection with Density Estimation

Anomaly Detection with Density Estimation

Benjamin Nachman, David Shih

We leverage recent breakthroughs in neural density estimation to propose a new unsupervised anomaly detection technique (ANODE). By estimating the probability density of the data in a signal region and in sidebands, and interpolating the latter into the signal region, a likelihood ratio of data vs. background can be constructed. This likelihood ratio is broadly sensitive to overdensities in the data that could be due to localized anomalies. In addition, a unique potential benefit of the ANODE method is that the background can be directly estimated using the learned densities. Finally, ANODE is robust against systematic differences between signal region and sidebands, giving it broader applicability than other methods. We demonstrate the power of this new approach using the LHC Olympics 2020 R\&D Dataset. We show how ANODE can enhance the significance of a dijet bump hunt by up to a factor of 7 with a 10\% accuracy on the background prediction. While the LHC is used as the recurring example, the methods developed here have a much broader applicability to anomaly detection in physics and beyond.

Has any anomaly been found?

- Anomaly detection is **used to find unusual objects, eg weird looking galaxies**. For example, a galaxy that recently collided with another one.
- As far as I know, **no confirmed discovery of “new physics”** has been made via anomaly detection methods using machine learning.
- Anomalies **often = detector issues**: Many flagged "anomalies" are **noise, miscalibrations, or rare but known effects**.
- Not every anomaly can be examined by experts (just like every “UFO sighting”). Many experiments have some unexplained data, but it is believed to have mundane explanations.

Examples

- CERN's LHC:
 - **"LHC Olympics" challenge (2020):** invited ML groups to find hidden signals in simulated collider data without knowing the new physics models.
 - But: no real data yet yielded a confirmed anomaly → discovery.
- Astrophysics:
 - ML has found rare events (e.g., supernova outliers, odd galaxy morphologies), but again, no "new physics" in a fundamental sense.
- IceCube and LIGO:
 - ML has helped clean data and detect outliers in real-time,
 - but findings have supported existing models, not contradicted them.

Example: SNAD project

- <https://arxiv.org/abs/2410.18875> Exploring the Universe with SNAD: Anomaly Detection in Astronomy

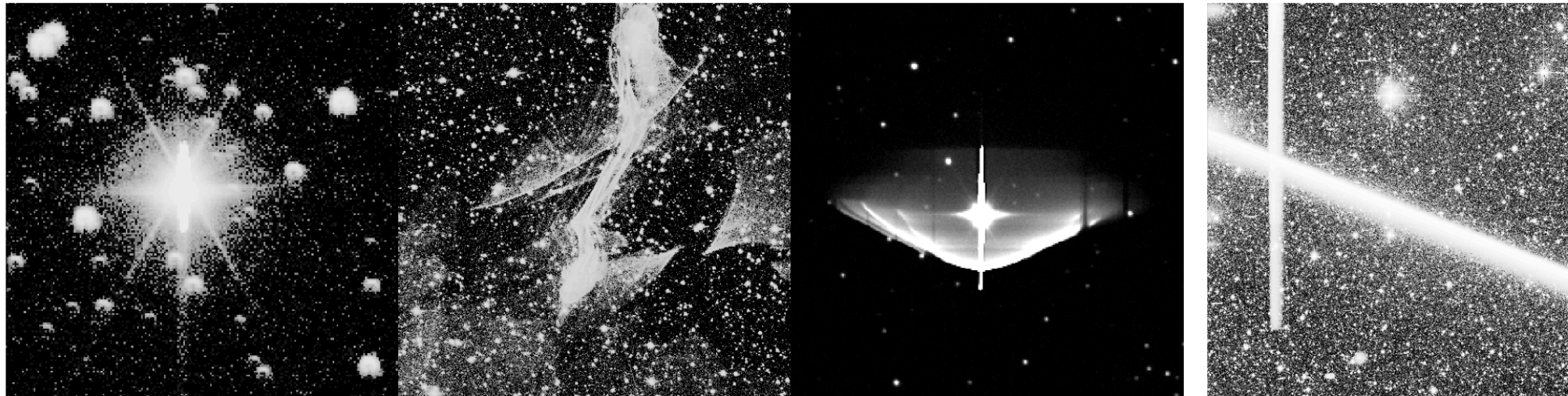


Fig. 3: Some examples from the SNAD catalog of artefacts.

Abstract. SNAD is an international project with a primary focus on detecting astronomical anomalies within large-scale surveys, using active learning and other machine learning algorithms. The work carried out by SNAD not only contributes to the discovery and classification of various astronomical phenomena but also enhances our understanding and implementation of machine learning techniques within the field of astrophysics. This paper provides a review of the SNAD project and summarizes the advancements and achievements made by the team over several years.

The LHC Olympics 2020

- <https://arxiv.org/abs/2101.08320> The LHC Olympics 2020: A Community Challenge for Anomaly Detection in High Energy Physics
- Groups used autoencoders, likelihood-free methods, and density estimation to propose promising regions for follow-up.

The LHC Olympics 2020

A Community Challenge for Anomaly Detection in High Energy Physics



Course logistics

- **Reading for this lecture:**
 - See references on the slides.