Machine Learning based Turbulence Prediction in Urban Areas

CIMPA Summer School 2026 | Course Part 2

Maximilian Dauner
maximilian.dauner0@hm.edu
Munich University of Applied Sciences
Munich Center for Digital Sciences and AI (MUC.DAI)

HM
MUC.DAI
Munich Center for
Digital Sciences and AI

CIMPA

Agenda – 22. May

Uses cases & my research topic

Introduction to AI

AI-Topic: Dataset

Agenda – 24. May

Short recap

Lab 1
From PDE Simulations to Training Data

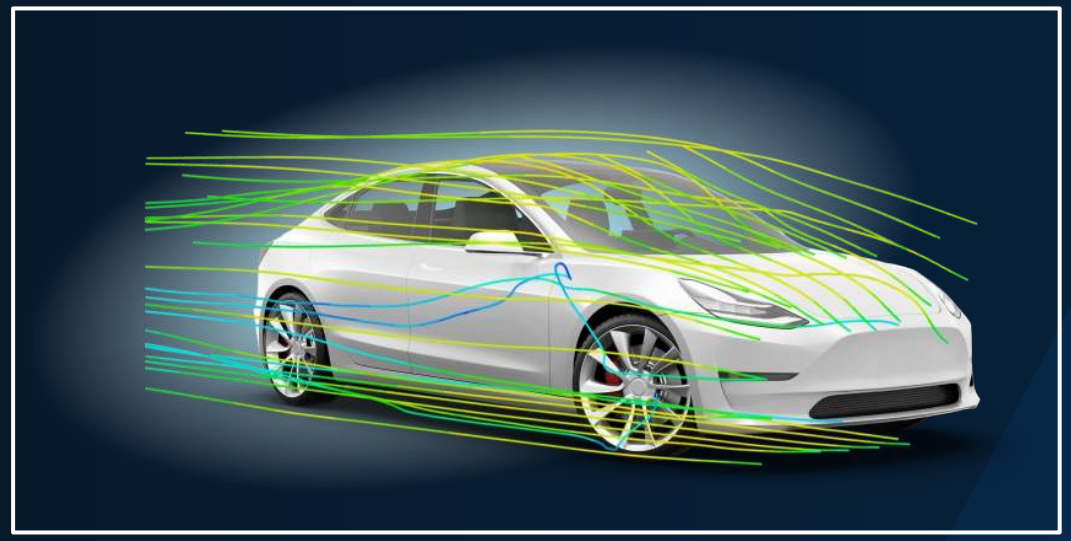
Introduction to neural network and training

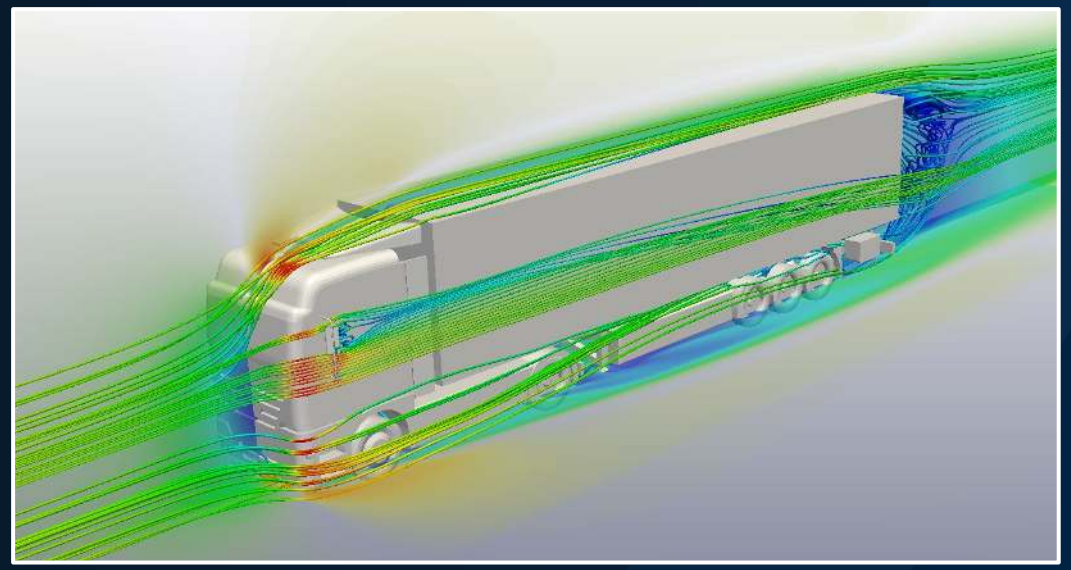
AI-Topic: (Fourier) Neural Operator Setup

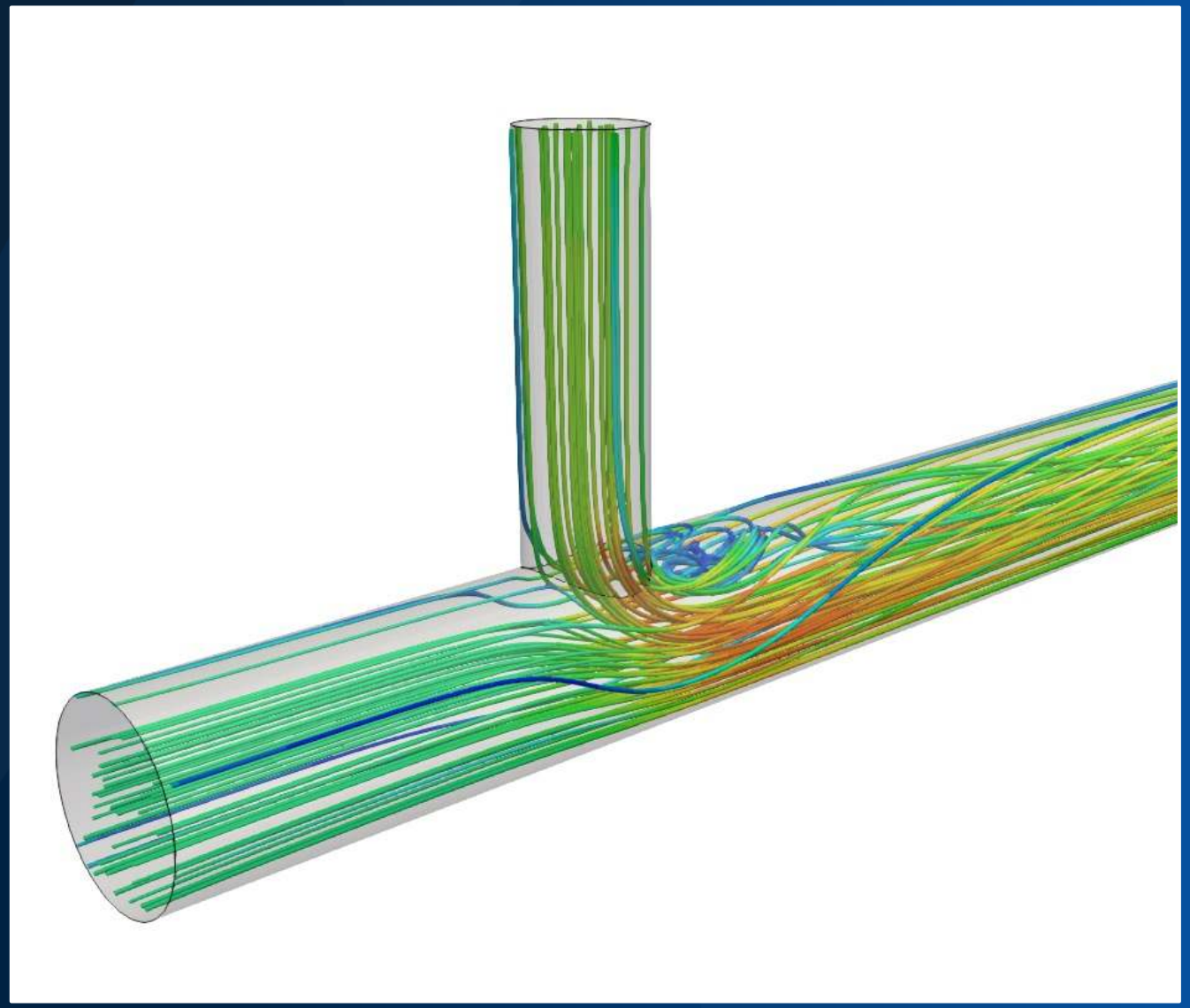
Lab 2
From Neural Networks to PDE Prediction

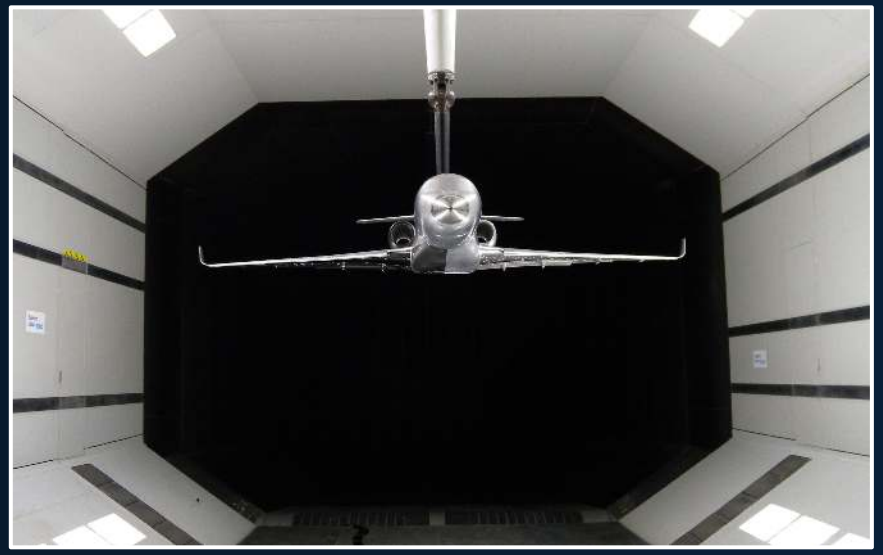
HM MUC.DAI
Munich Center for
Digital Sciences and AI

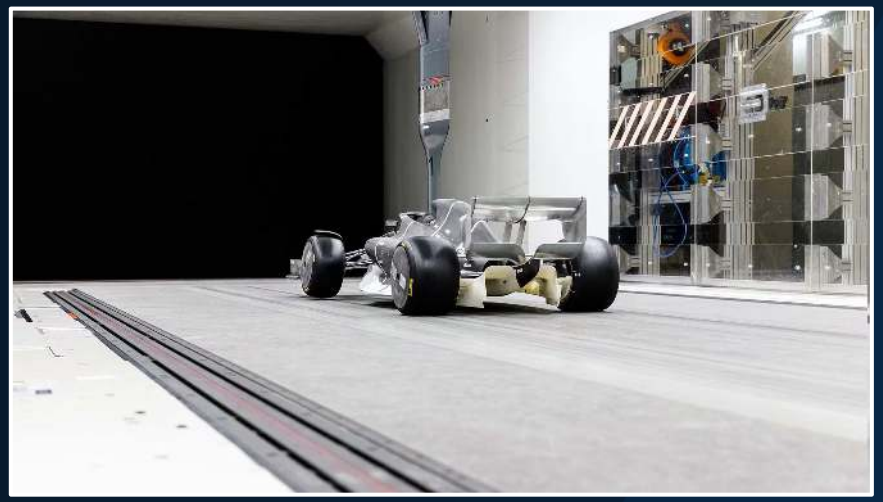

CIMPA







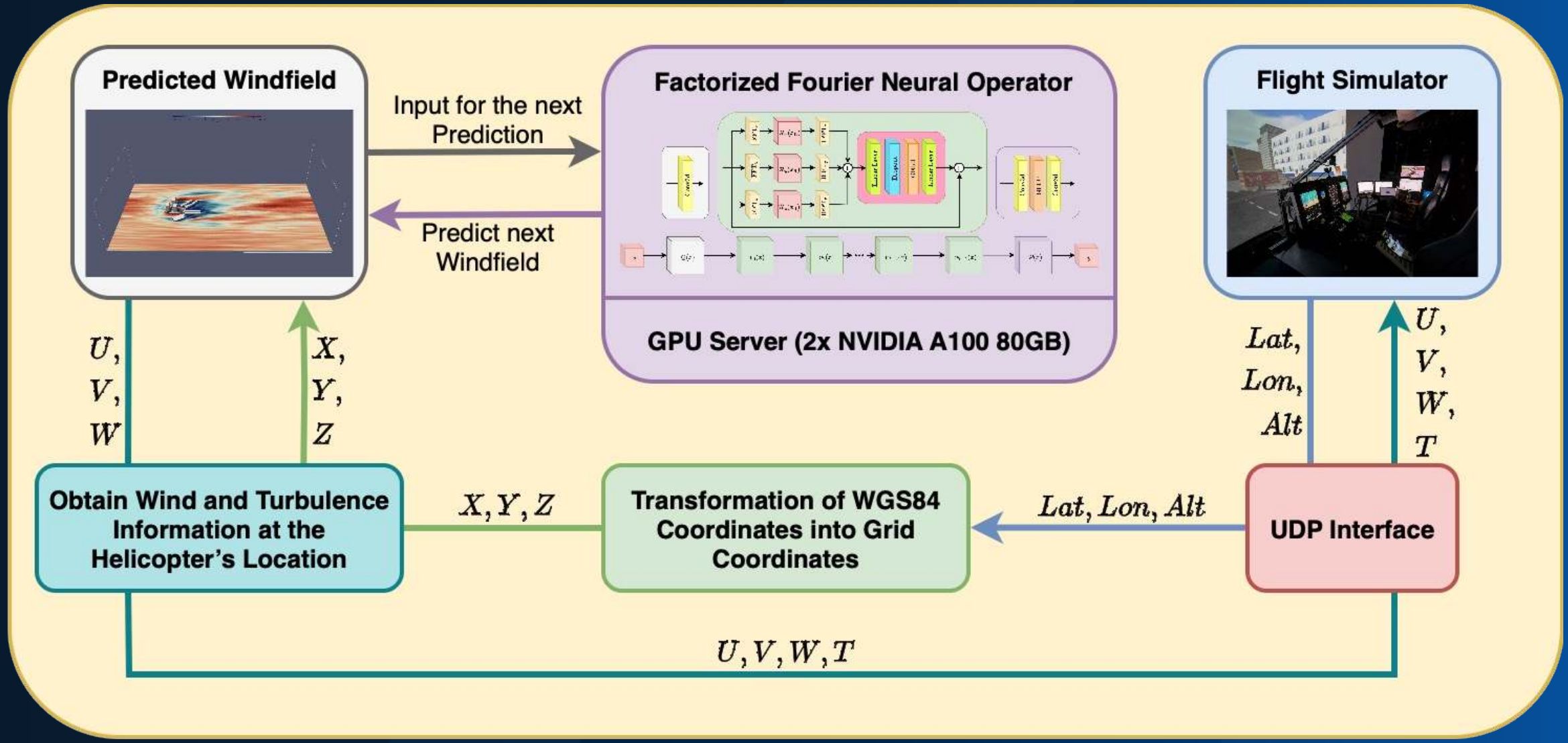




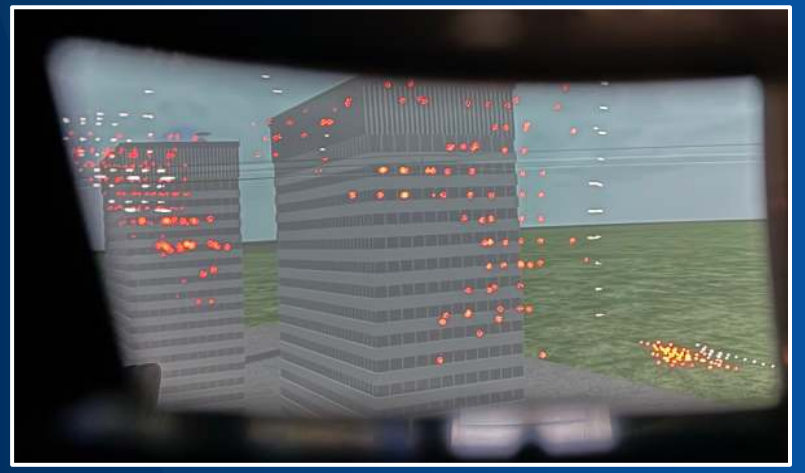

CAE7000

AI can learn from existing simulation and measurement data to create **fast surrogate models** that approximate parts of the flow prediction process much faster than running a full CFD simulation every time.

The main advantage is not that AI replaces physics or CFD simulations, but that it can **accelerate repeated predictions, support design exploration, and make complex flow information available faster**.


Predicted Windfield
Input for the next Prediction
Predict next Windfield
Factorized Fourier Neural Operator
GPU Server (2x NVIDIA A100 80GB)
Flight Simulator
U, V, W
X, Y, Z
Lat, Lon, Alt
U, V, W, T
Obtain Wind and Turbulence Information at the Helicopter's Location
X, Y, Z
Transformation of WGS84 Coordinates into Grid Coordinates
Lat, Lon, Alt
UDP Interface
U, V, W, T


REISER



Artificial Intelligence (AI)

Systems that simulate aspects of human intelligence, such as reasoning, learning, problem-solving, and decision-making.

Machine Learning (ML)

A subset of AI in which systems learn from data and improve their performance through training without being explicitly programmed for every task.

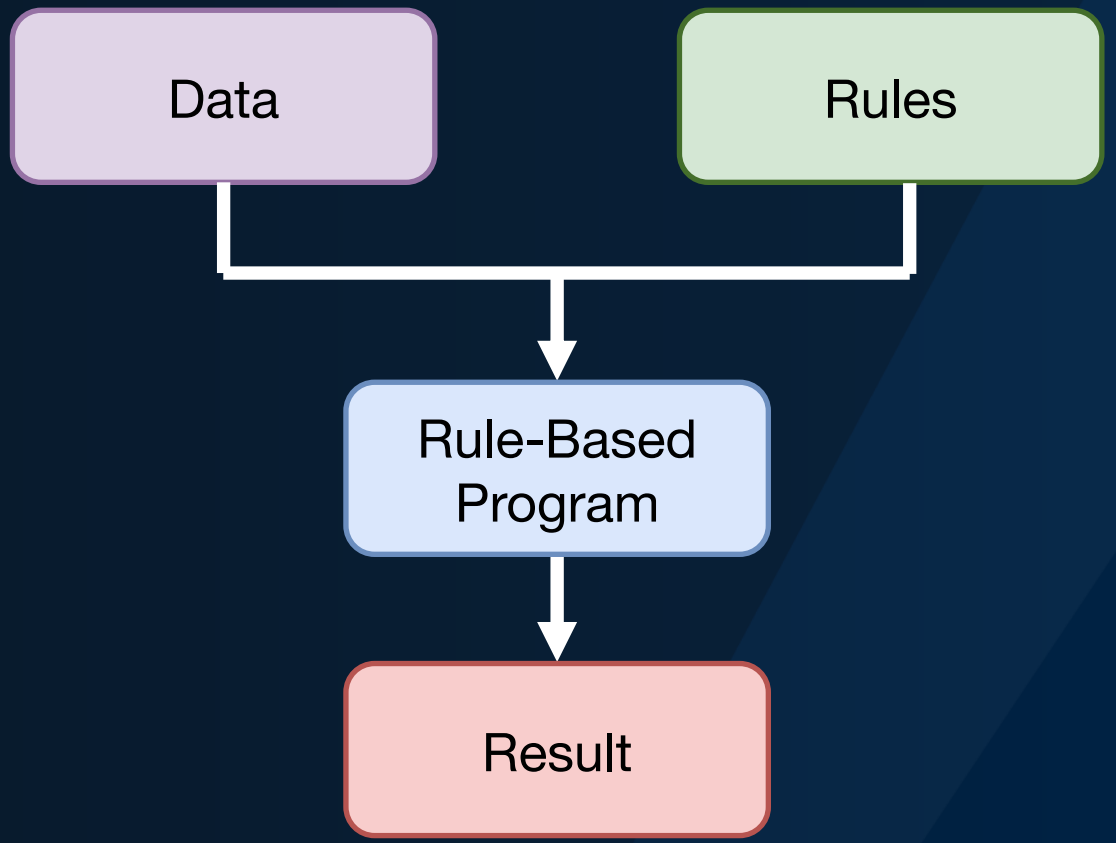
Neural Network (NN)

Computational models inspired by the human brain that recognize patterns and relationships in data through interconnected layers of artificial neurons.

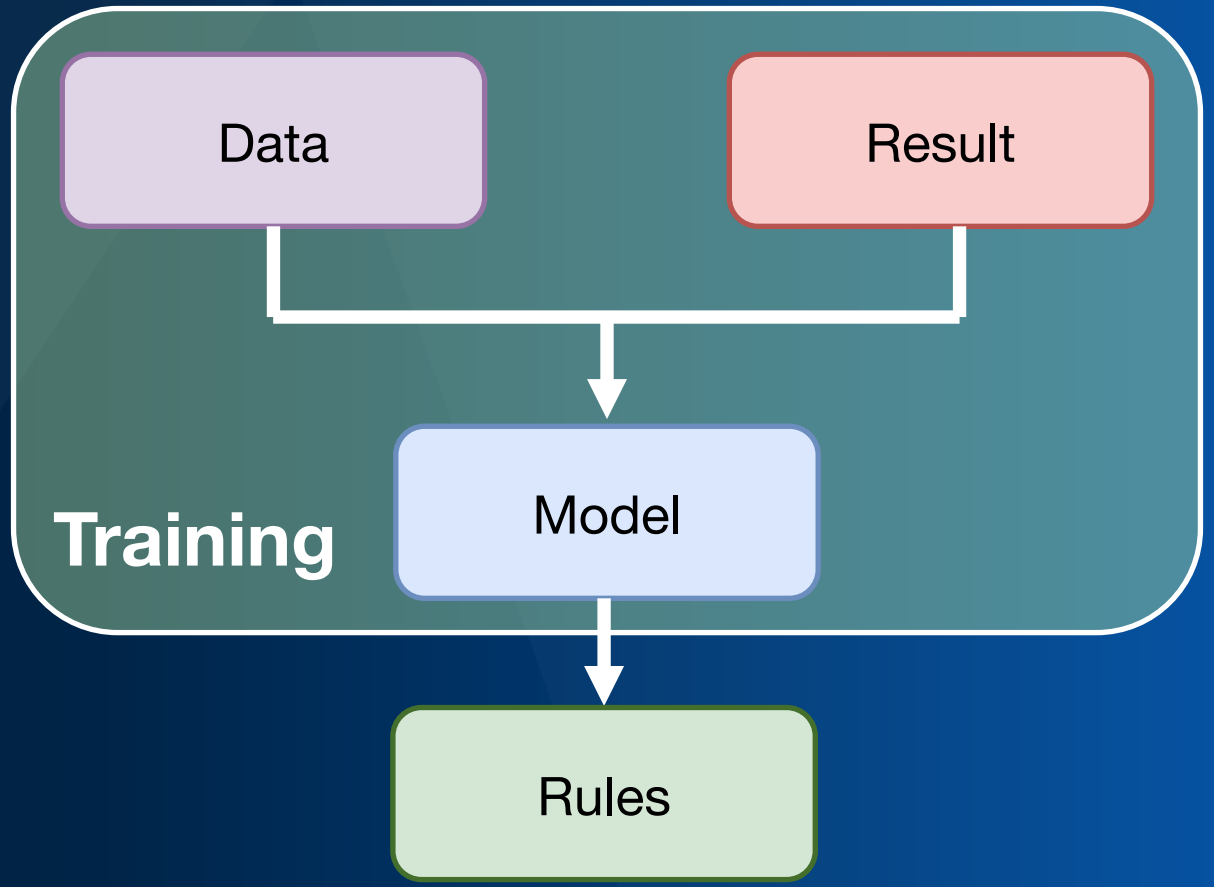
Deep Learning

A subset of machine learning that uses deep neural networks with multiple layers to process complex data and solve advanced tasks.

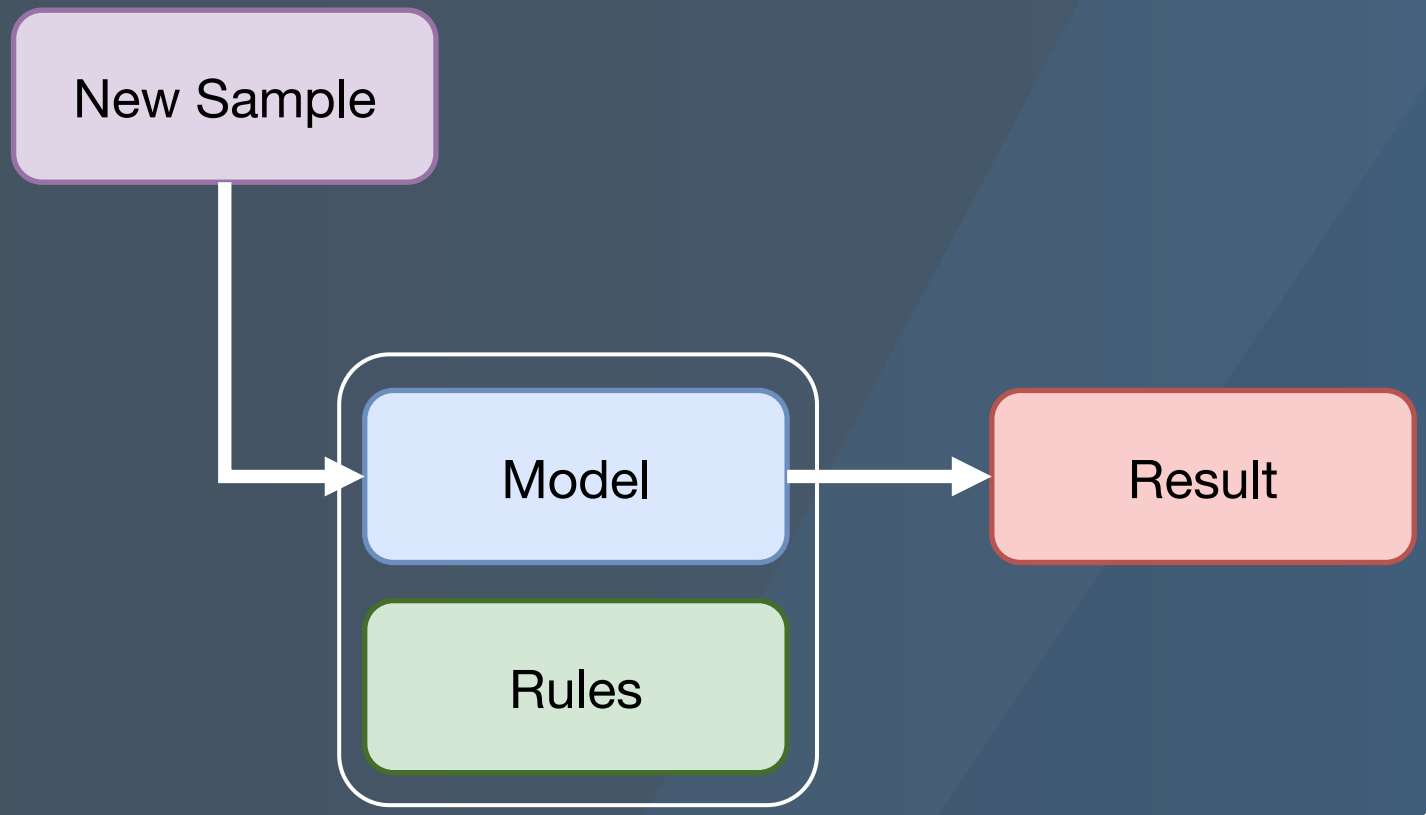
Rule-Based System

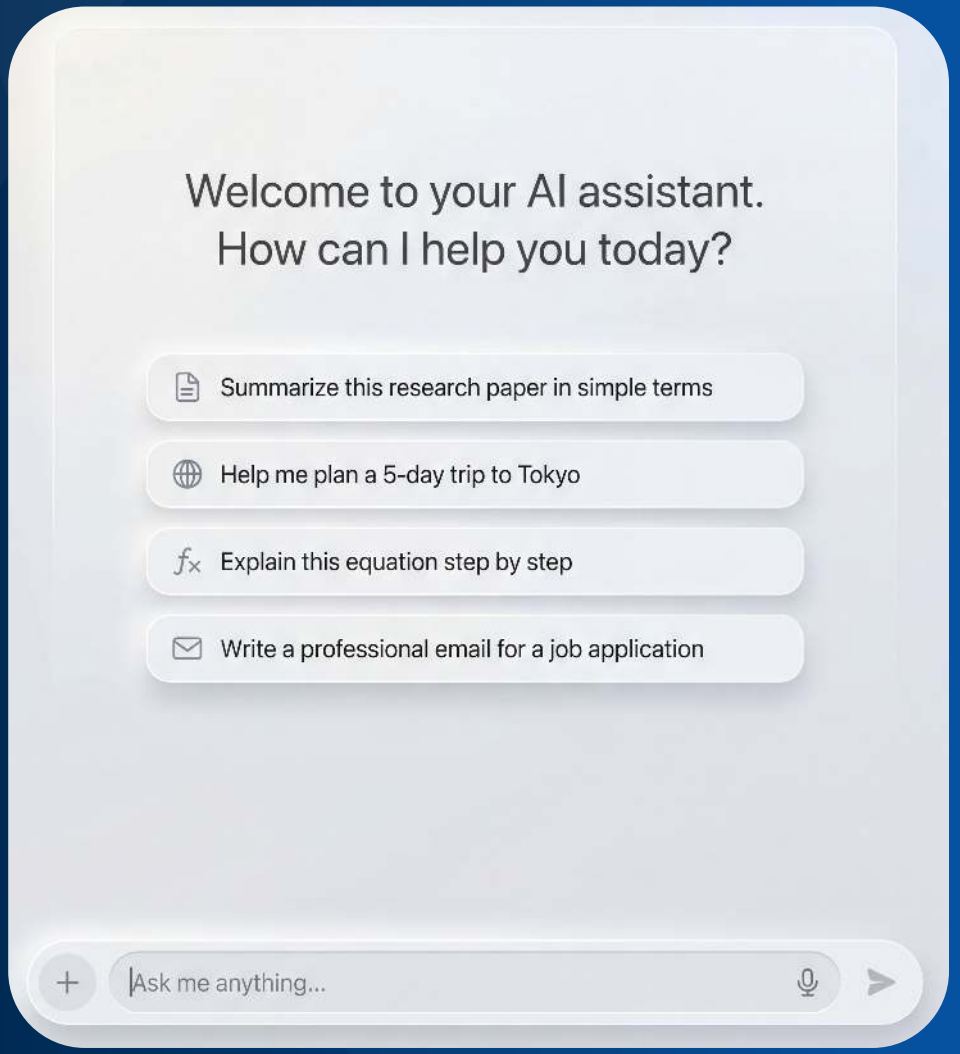


Learning-Based System



Inference


New Sample
Model
Rules
Result


Welcome to your AI assistant.
How can I help you today?
Summarize this research paper in simple terms
Help me plan a 5-day trip to Tokyo
Explain this equation step by step
Write a professional email for a job application
Ask me anything...

Dataset

A collection of examples that form the model's knowledge base.

Model

A neural network with a specific architecture adapted on the data or use case.

Training

The process where a neural network learns patterns from data.

Inference

After training, the model uses what it has learned to make predictions or generate content from unseen inputs.

Define the specific use case and the needed spatial and temperol resolution first

Coordinates: $\mathbf{x} = (x, y, z) \qquad \mathbf{x} \in \mathbb{R}^3$

Discrete time: $t_i = t_0 + i\Delta t$

Velocity vector: $\mathbf{u}(\mathbf{x}, t) \in \mathbb{R}^3$

Velocity components: $\mathbf{u}(\mathbf{x}, t) = \big(u(\mathbf{x}, t), v(\mathbf{x}, t), w(\mathbf{x}, t)\big)$

Wind-speed magnitude: $U(\mathbf{x}, t) = \sqrt{u(\mathbf{x}, t)^2 + v(\mathbf{x}, t)^2 + w(\mathbf{x}, t)^2}$

Urban Area Wind Prediction

Local heat, ventilation, comfort, vegetation effects, urban planning scenario.

3D spatial domain: $\Omega_{3D} = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}] \subset \mathbb{R}^3$

3D grid points: $x_j = x_{\min} + j\Delta x, \quad y_k = y_{\min} + k\Delta y, \quad z_l = z_{\min} + l\Delta z$

Sample location: $\mathbf{x}_{j,k,l} = (x_j, y_k, z_l) \in \mathbb{R}^3$

Velocity sample: $\mathbf{u}(\mathbf{x}_{j,k,l}, t_i) = \left(u(\mathbf{x}_{j,k,l}, t_i), v(\mathbf{x}_{j,k,l}, t_i), w(\mathbf{x}_{j,k,l}, t_i)\right) \in \mathbb{R}^3$

Urban Area Wind Prediction

Local heat, ventilation, comfort, vegetation effects, urban planning scenario.

$$\Omega_{3D}^{urban} = \{(x, y, z) \in \mathbb{R}^3 \mid x \in [0{,}750]\ \mathrm{m}, y \in [0{,}750]\ \mathrm{m}, z \in [0{,}200]\ \mathrm{m}\}$$

$$\Delta x \approx \Delta y \approx \Delta z \approx 1\text{–}10\ \mathrm{m}$$

$$t_i = t_0 + i\Delta t, \qquad \Delta t \text{ ranging from seconds to hours}$$

$$\mathbf{X}_{3D}^{\mathrm{urban}} \in \mathbb{R}^{N_t \times N_x \times N_y \times N_z \times C}, \qquad C = (u, v, w, T, p, q, \text{building mask}, \text{vegetation})$$

Urban Area Wind Prediction

Local heat, ventilation, comfort, vegetation effects, urban planning scenario.

Dense urban wind and microclimate datasets are typically generated from CFD simulations or wind-tunnel experiments over realistic urban geometries, while field measurements and long-term meteorological data are used to define boundary conditions, evaluate seasonal relevance, and validate the predicted flow patterns.

City Geography Markup Language (CityGML)

Open, standardized data model and exchange format used to store and share digital 3D models of cities and landscapes.

LoD0	Building footprints, roof prints, roads, terrain surfaces
LoD1	Buildings are vertical boxes with flat roofs
LoD2	Roof shapes, roof planes, sometimes building parts
LoD3	Windows, doors, façade details, balconies, roof/facade structures

Urban Scene = Buildings + Terrain + Vegetation + Land Cover + Infrastructure

Spacing for urban turbulences

Use the target physics to decide how much of the city must be resolved.

Coarse urban climate

$\Delta \approx 10\text{–}40\ \mathrm{m}$

City-scale thermal or climate screening. Buildings may be parameterized or partly resolved.

Building-resolving flow

$\Delta \approx 1\text{–}5\ \mathrm{m}$

Urban wind, ventilation, pedestrian or drones. Often the relevant regime for turbulence datasets.

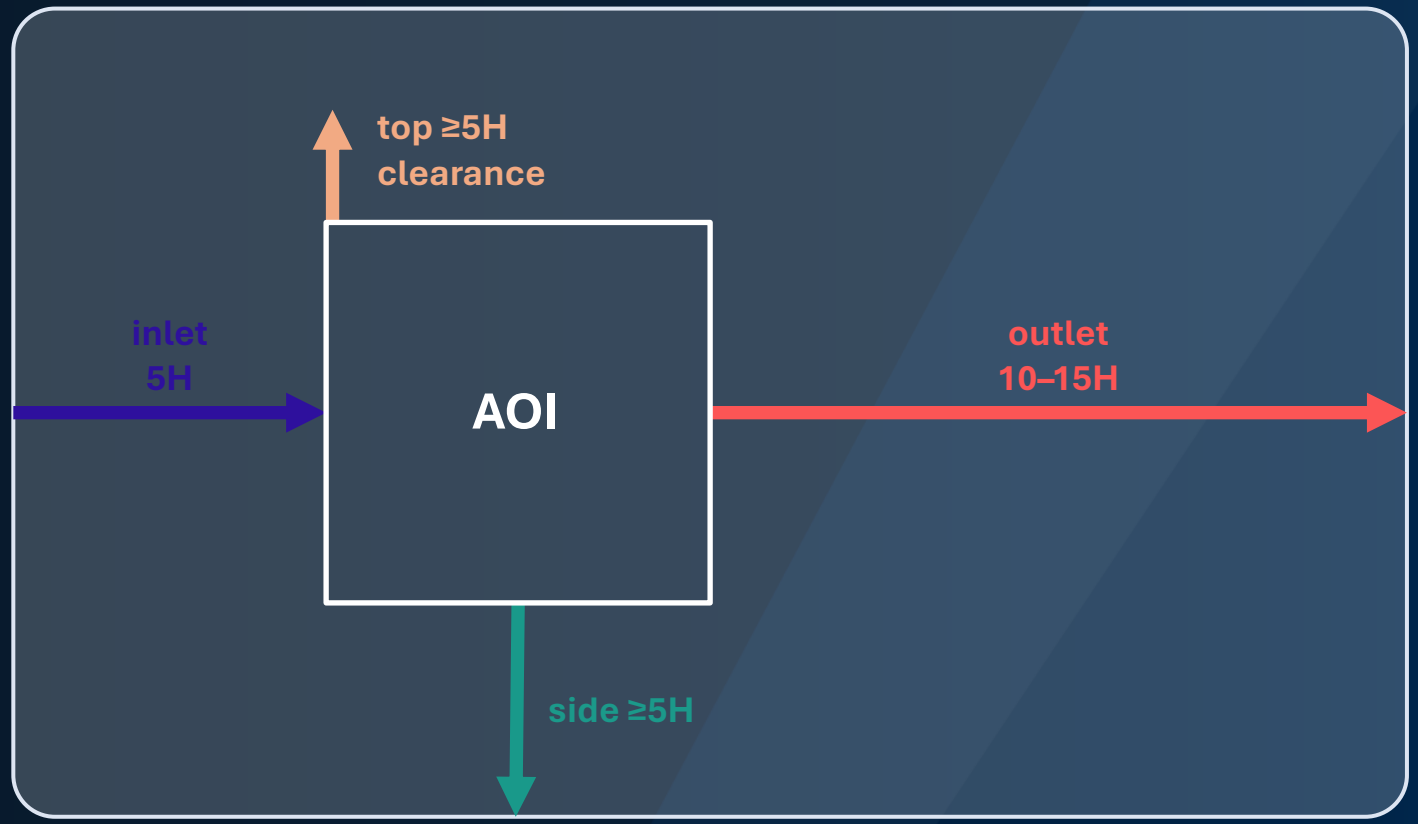
High-detail turbulence

$\Delta \approx 0.5\text{–}2\ \mathrm{m}$

Research-grade LES around buildings. Expensive but better for shear layers and near-ground statistics.

Prominent boundary-distance

AIJ/COST-style guidelines provide starting values. Sensitivity tests remain necessary.



$L_{upstream} \geq 5H$

$L_{downstream} \geq 10/15\ H$ (AIJ/COST)

$L_{lateral} \geq 5H$

$L_{top\ clearance} \geq 5H$

Urban Area Wind Prediction

Local heat, ventilation, comfort, vegetation effects, urban planning scenario.

$H = 40\ \text{m}$

$\text{AOI} = 500\ \text{m} \times 500\ \text{m}$

$\Delta x = \Delta y = \Delta z = 2\ \text{m}$

$\text{Lup} = 5H = 200\ \text{m}$

$\text{Ldown} = 15H = 600\ \text{m}$

$\text{Lside} = 5H = 200\ \text{m}$

$\text{ztop} \geq H + 5H = 240\ \text{m}$

$Lx \approx 200 + 500 + 600 = 1300\ \text{m}$

$Ly \approx 200 + 500 + 200 = 900\ \text{m}$

$Nz \approx 240/2 = 120$ grid levels

$Ncells \approx 650 \cdot 450 \cdot 120 = 35.1$ million

Urban Area Wind Prediction

Local heat, ventilation, comfort, vegetation effects, urban planning scenario.

$Lx \approx 200 + 500 + 600 = 1300$ m

$Ly \approx 200 + 500 + 200 = 900$ m

$Nz \approx 240/2 = 120$ grid levels

$Ncells \approx 650 \cdot 450 \cdot 120 = 35.1$ million

$C = (u, v, w, p, q, \text{ mask}, \text{SDF})$

$Nt = 60$

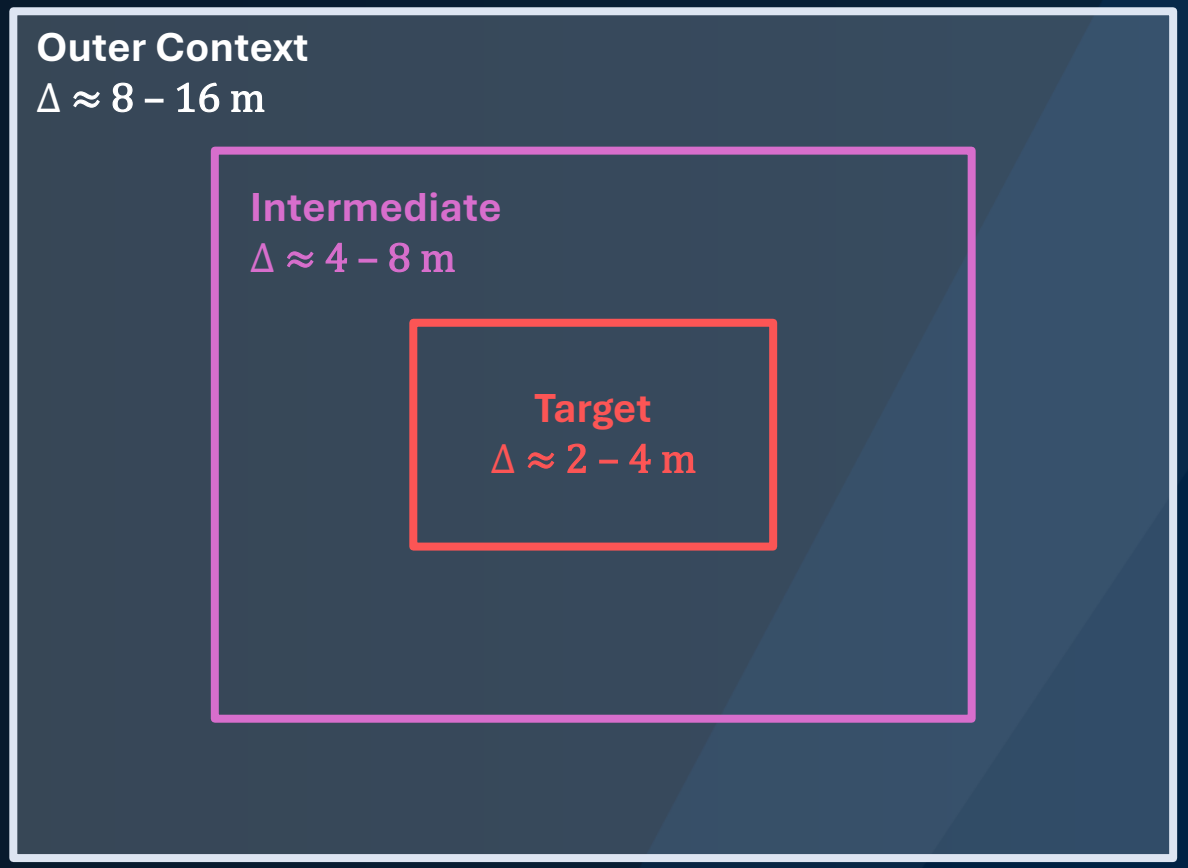
$60 \cdot 31.5$ million $\cdot 7 \cdot 4$ bytes

$= 52{,}920{,}000{,}000$ bytes $= 52.92$ GB

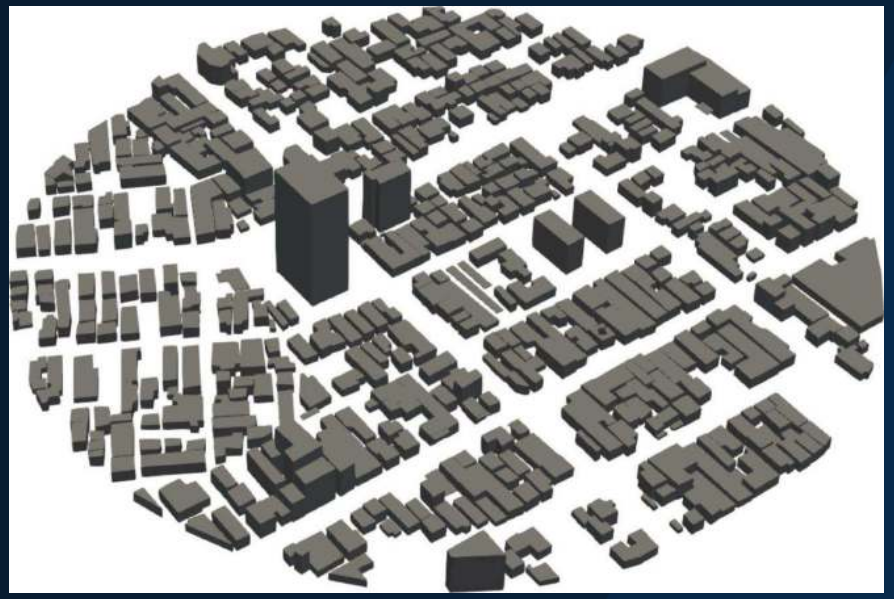
Memory snapshot $\approx Nt \cdot Ncells \cdot C \cdot 4$ bytes (float32)

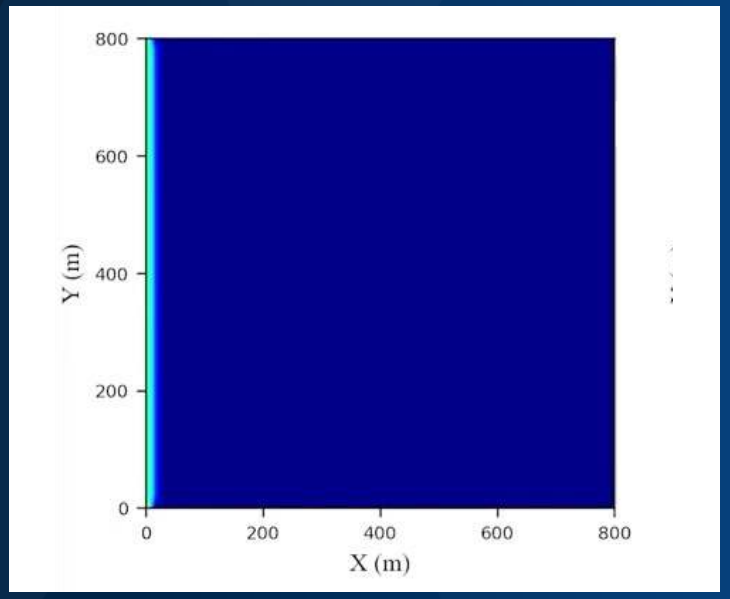
Domain nesting: coarse context, fine target region

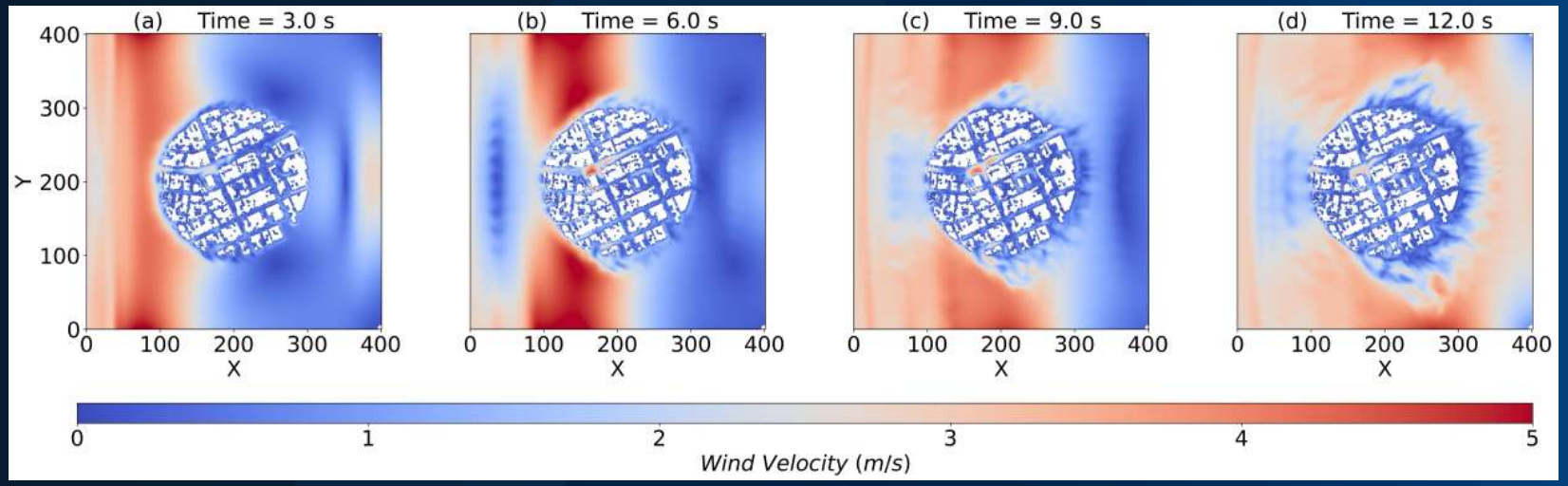
Nested grids let you include city-scale context without forcing 2 m cells everywhere.

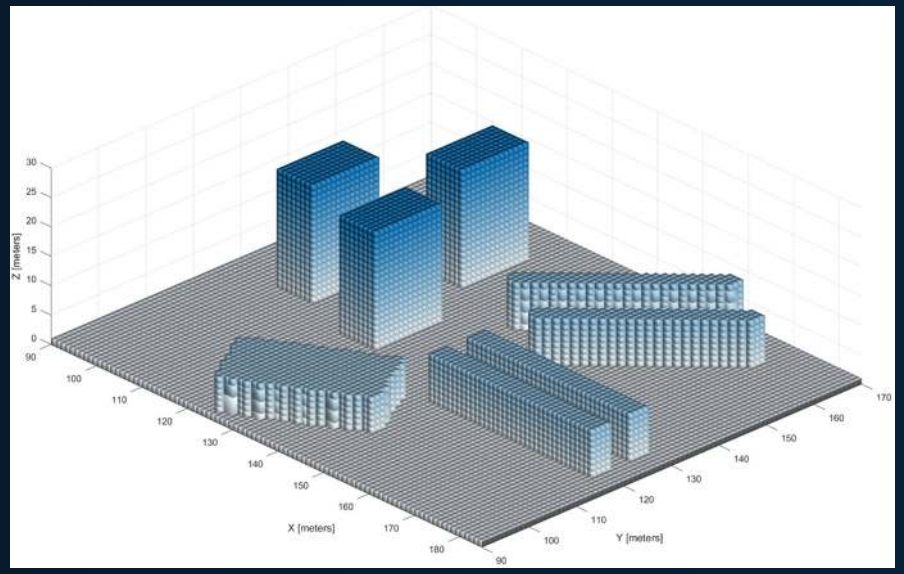


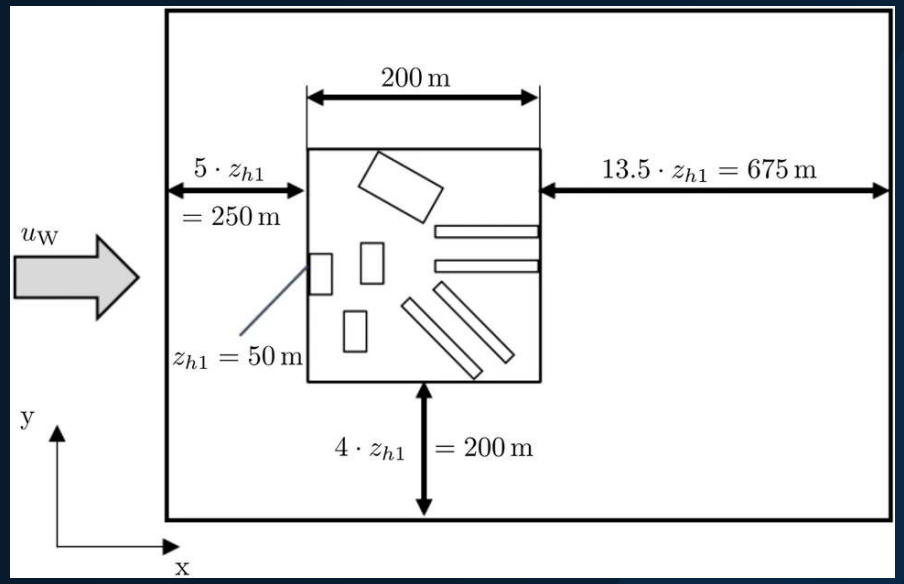
- outer domain provides topography and larger-scale boundary-layer context
- inner domain resolves flow-building interactions
- training labels can be exported only from the high-resolution AOI

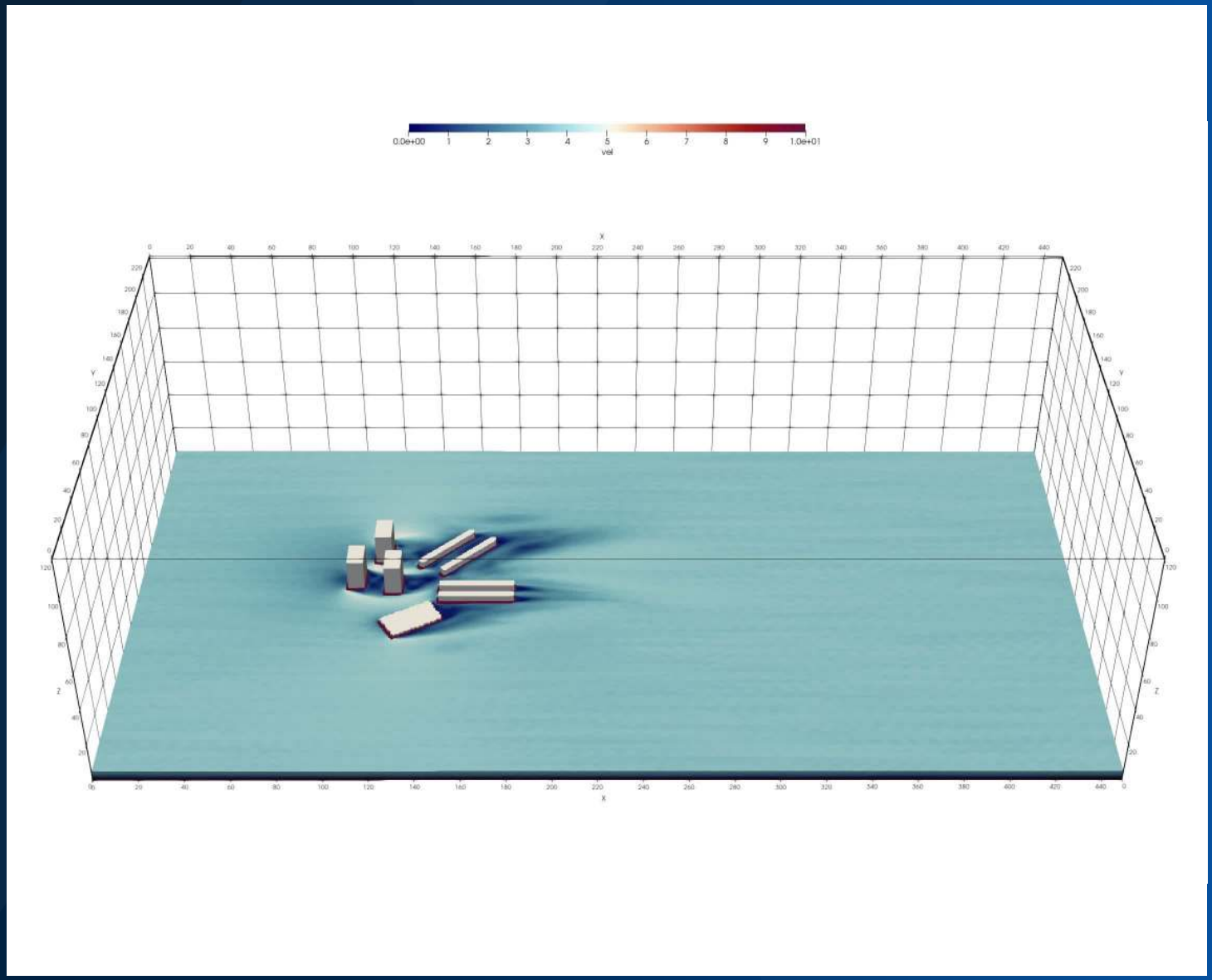


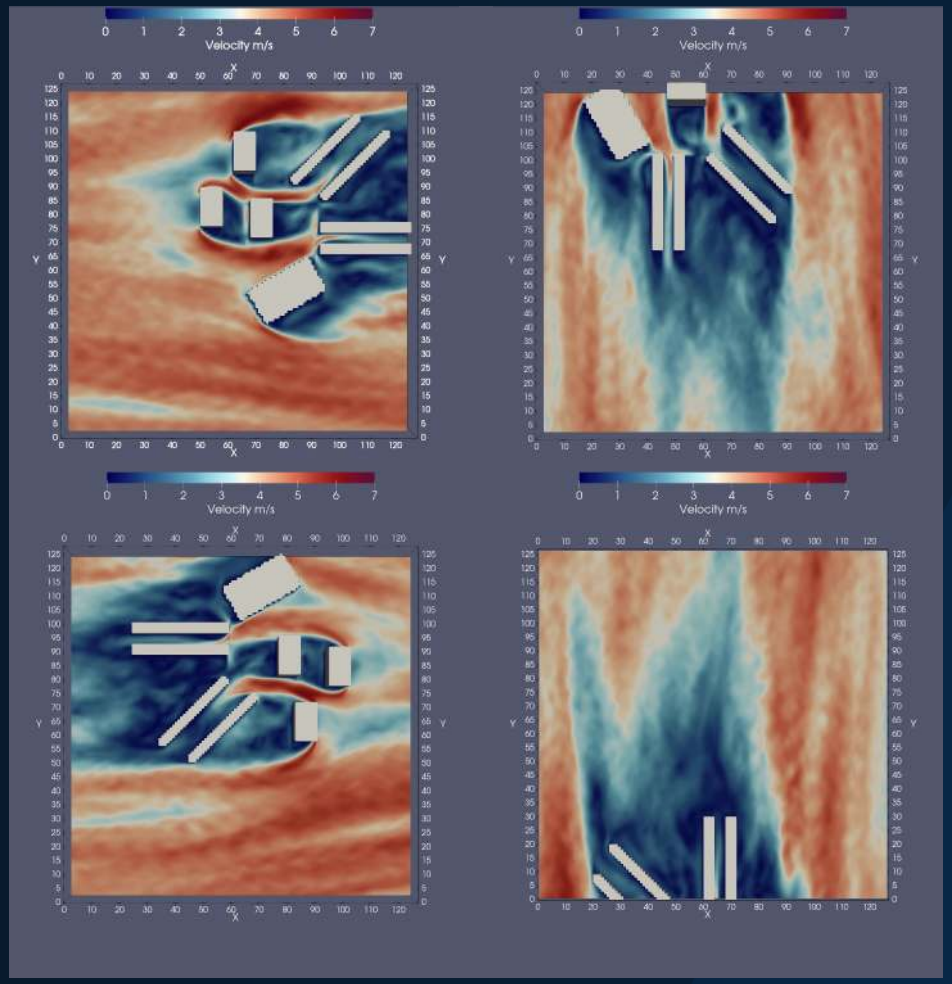

800
600
400
200
0
Y (m)
0
200
400
600
800
X (m)

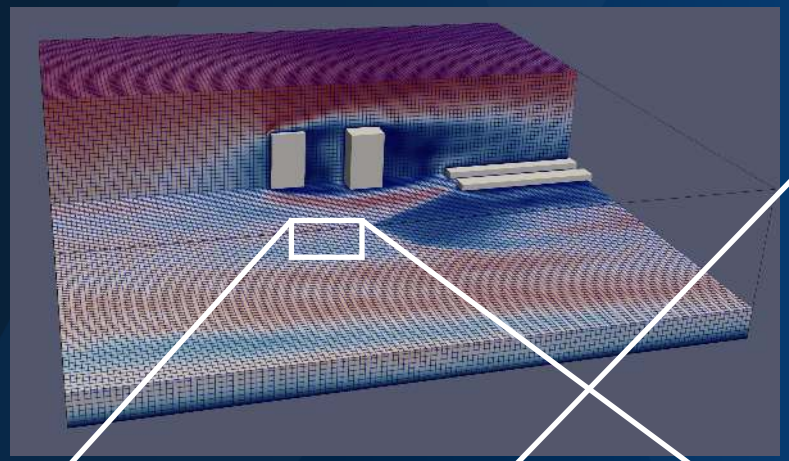

(a) Time = 3.0 s
(b) Time = 6.0 s
(c) Time = 9.0 s
(d) Time = 12.0 s
Y
X
0
100
200
300
400
0
1
2
3
4
5
Wind Velocity (m/s)

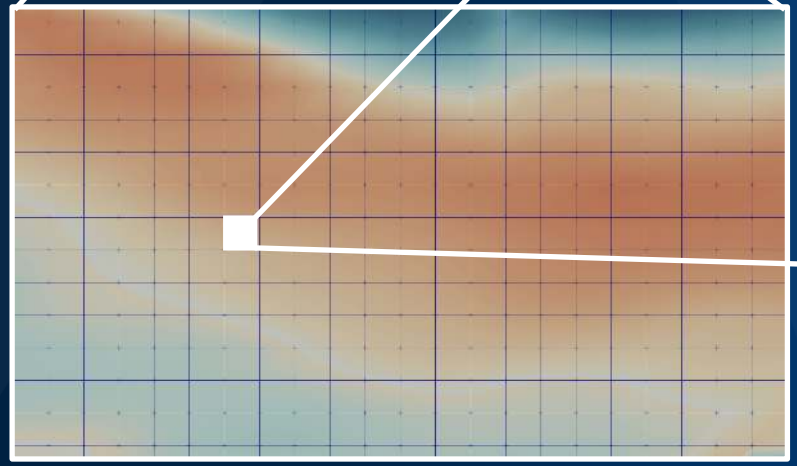

Z [meters]
X [meters]
Y [meters]


200 m
$5 \cdot z_{h1}$
$= 250\,\mathrm{m}$
$13.5 \cdot z_{h1} = 675\,\mathrm{m}$
u_W
$z_{h1} = 50\,\mathrm{m}$
$4 \cdot z_{h1}$
$= 200\,\mathrm{m}$
y
x


0.0e+00
1.0e+01
vel







Type	free / obstacle cell
x	x-coordinates
y	y-coordinates
z	z-coordinates
u	velocity in x-direction
v	velocity in y-direction
w	velocity in z-direction

Global Weather Simulation

1	**Define task**	mean comfort map, dynamic wind forecast, thermal microclimate, pollutant spread
2	**Acquire geometry**	LoD2/CityGML/CityJSON, terrain, vegetation, surfaces, licenses
3	**Prepare domain**	clean, simplify, mesh/voxelize, set boundaries, define grid
4	**Design cases**	directions, speeds, stability, weather episodes, surface scenarios
5	**Run + validate**	CFD/LES/microclimate model, monitoring, field/wind-tunnel checks
6	**Export ML dataset**	Zarr/HDF5/NetCDF, masks/SDF, metadata, splits, documentation

Lab 1

From PDE Simulations to Training Data

In this lab, you will explore a 1D PDEBench dataset and learn how physical simulation data is structured, visualized, analyzed, and transformed into supervised learning pairs.

By computing trajectory statistics, studying distributions, checking data splits, and testing simple baselines, you will understand why dataset quality and preprocessing are essential for reliable scientific machine learning.

https://syncandshare.lrz.de/getlink/fiVMAZBH1ha1h4Lrn4a98C/public_data

MUC.DAI
Munich Center for
Digital Sciences and AI

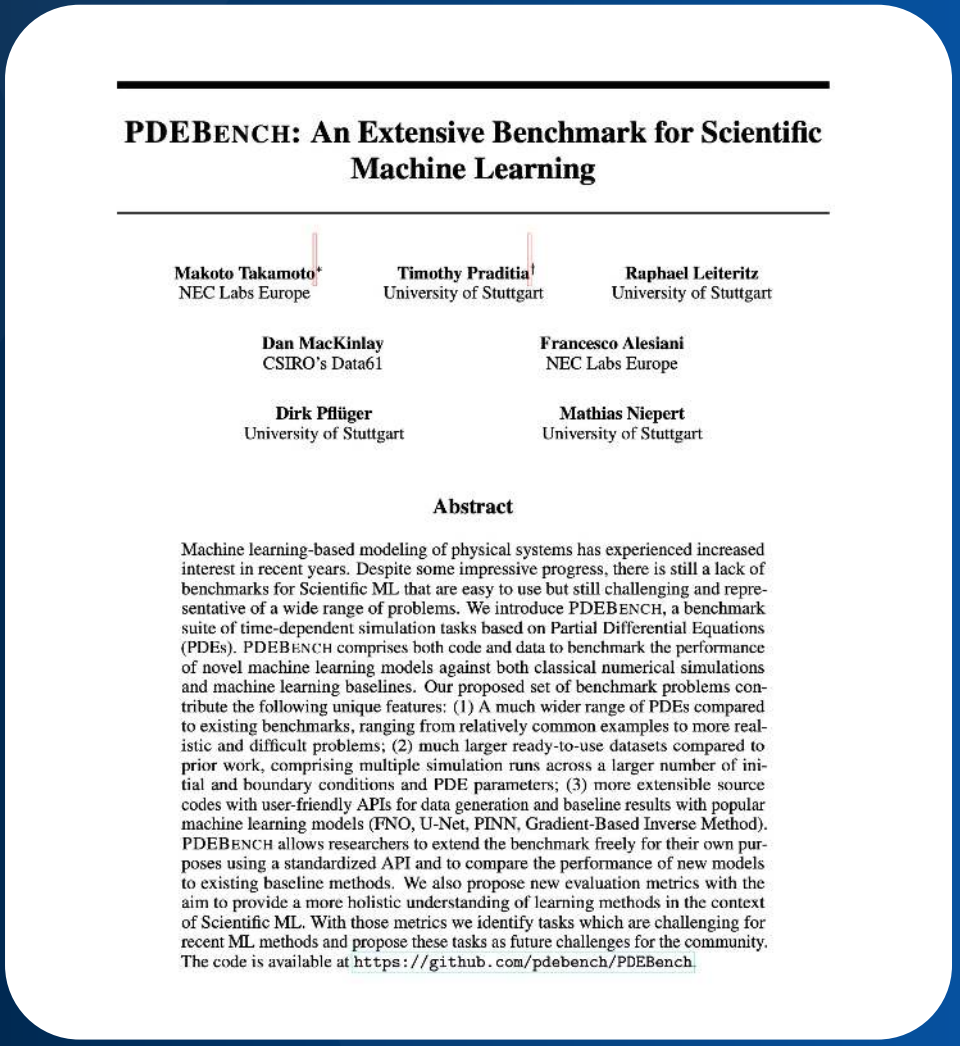
CIMPA

Lab 1

From PDE Simulations to Training Data

In this lab, you will explore a 1D PDEBench dataset and learn how physical simulation data is structured, visualized, analyzed, and transformed into supervised learning pairs.

By computing trajectory statistics, studying distributions, checking data splits, and testing simple baselines, you will understand why dataset quality and preprocessing are essential for reliable scientific machine learning.



PDEBENCH: An Extensive Benchmark for Scientific Machine Learning

Makoto Takamoto[*] NEC Labs Europe

Timothy Praditia[†] University of Stuttgart

Raphael Leiteritz University of Stuttgart

Dan MacKinlay CSIRO's Data61

Francesco Alesiani NEC Labs Europe

Dirk Pflüger University of Stuttgart

Mathias Niepert University of Stuttgart

Abstract

Machine learning-based modeling of physical systems has experienced increased interest in recent years. Despite some impressive progress, there is still a lack of benchmarks for Scientific ML that are easy to use but still challenging and representative of a wide range of problems. We introduce PDEBENCH, a benchmark suite of time-dependent simulation tasks based on Partial Differential Equations (PDEs). PDEBENCH comprises both code and data to benchmark the performance of novel machine learning models against both classical numerical simulations and machine learning baselines. Our proposed set of benchmark problems contribute the following unique features: (1) A much wider range of PDEs compared to existing benchmarks, ranging from relatively common examples to more realistic and difficult problems; (2) much larger ready-to-use datasets compared to prior work, comprising multiple simulation runs across a larger number of initial and boundary conditions and PDE parameters; (3) more extensible source codes with user-friendly APIs for data generation and baseline results with popular machine learning models (FNO, U-Net, PINN, Gradient-Based Inverse Method). PDEBENCH allows researchers to extend the benchmark freely for their own purposes using a standardized API and to compare the performance of new models to existing baseline methods. We also propose new evaluation metrics with the aim to provide a more holistic understanding of learning methods in the context of Scientific ML. With those metrics we identify tasks which are challenging for recent ML methods and propose these tasks as future challenges for the community. The code is available at `https://github.com/pdebench/PDEBench`.

MUC.DAI
Munich Center for
Digital Sciences and AI

CIMPA

Learning from Data

A brief overview of supervised and unsupervised learning.

Supervised Learning

The training data are paired observations

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$$

sampled from an unknown joint distribution

$$p_{data}(x, y)$$

The goal is to learn a function

$$f_\theta : \mathcal{X} \to \mathcal{Y}$$

that predicts y from x by minimizing a supervised prediction loss between $f_\theta(x_i)$ and y_i.

Minimize prediction error

Unsupervised Learning

The training data consists only of observations

$$\mathcal{D} = \{x_i\}_{i=1}^{N}$$

sampled from an unknown distribution

$$p_{data}(x)$$

The goal is to learn structure of that distribution, such as latent representations, clusters, reconstructions, or a generative density model.

Learn distributional structure

Supervised Learning – Dataset

From PDE definition to dataset construction.

A supervised PDE dataset consists of input-output pairs:

$$\mathcal{D} = \{(a_i, u_i)\}_{i=1}^{N}$$

a_i : initial condition, boundary condition, geometry, parameters

u_i : solution field, often multi-channel and time dependent

Example: $a_i = (u_0(x), \nu, f(x,t), \Omega, \partial\Omega), \quad u_i = u(x,t)$

Supervised Learning – Learning Objective

From PDE definition to dataset construction.

Learning Goal:

$$f_\theta : \mathcal{X} \to \mathcal{Y} \quad \text{such that} \quad f_\theta(x) \approx y$$

Training Objective:

$$\theta^* = \arg min_\theta \frac{1}{N} \Sigma_{i=1}^{N} \mathcal{L}(f_\theta(\mathrm{x_i}), \mathrm{y_i})$$

$$\approx \arg min_\theta \, \mathbb{E}_{(x,y)} \sim p_{data}[\, \mathcal{L}(f_\theta(\mathrm{x}), \mathrm{y})\,]$$

What makes f_θ learn?

Building the learned function step by step — from affine mappings to neural networks.

Dataset

A collection of examples that form the model's knowledge base.

Model

A neural network with a specific architecture adapted on the data or use case.

Training

The process where a neural network learns patterns from data.

Inference

After training, the model uses what it has learned to make predictions or generate content from unseen inputs.

Neural Network

A parameterized differentiable function.

Neural Network: f_θ: $\mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{\text{out}}}$

One dimensional affine model:

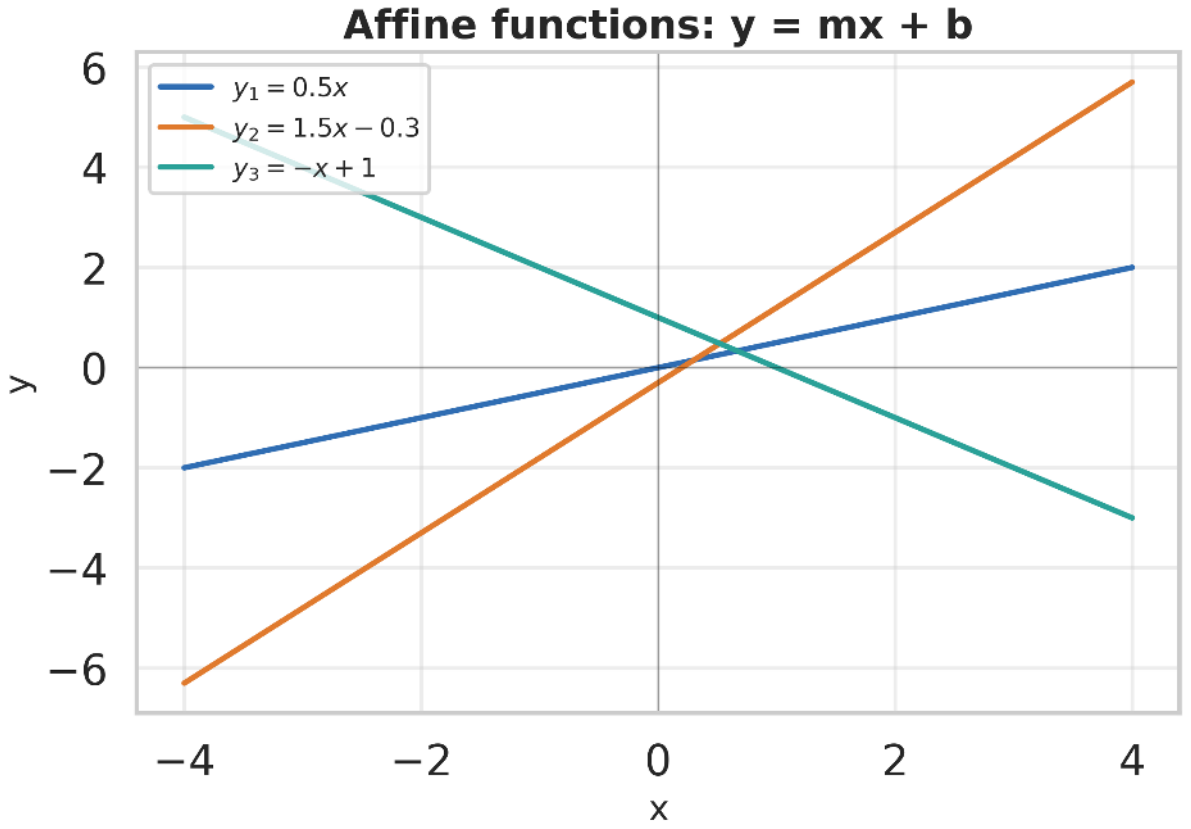
$$\hat{y} = f_\theta(x) = mx + b$$

$$\theta = \{m, b\}$$

m controls the slope, b shifts the line vertically

Error for one point:

$$e = mx + b - y = f_\theta(x) - y = \hat{y} - y$$



Neural Network

A parameterized differentiable function.

Neural Network: $f_\theta\colon \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{\text{out}}}$

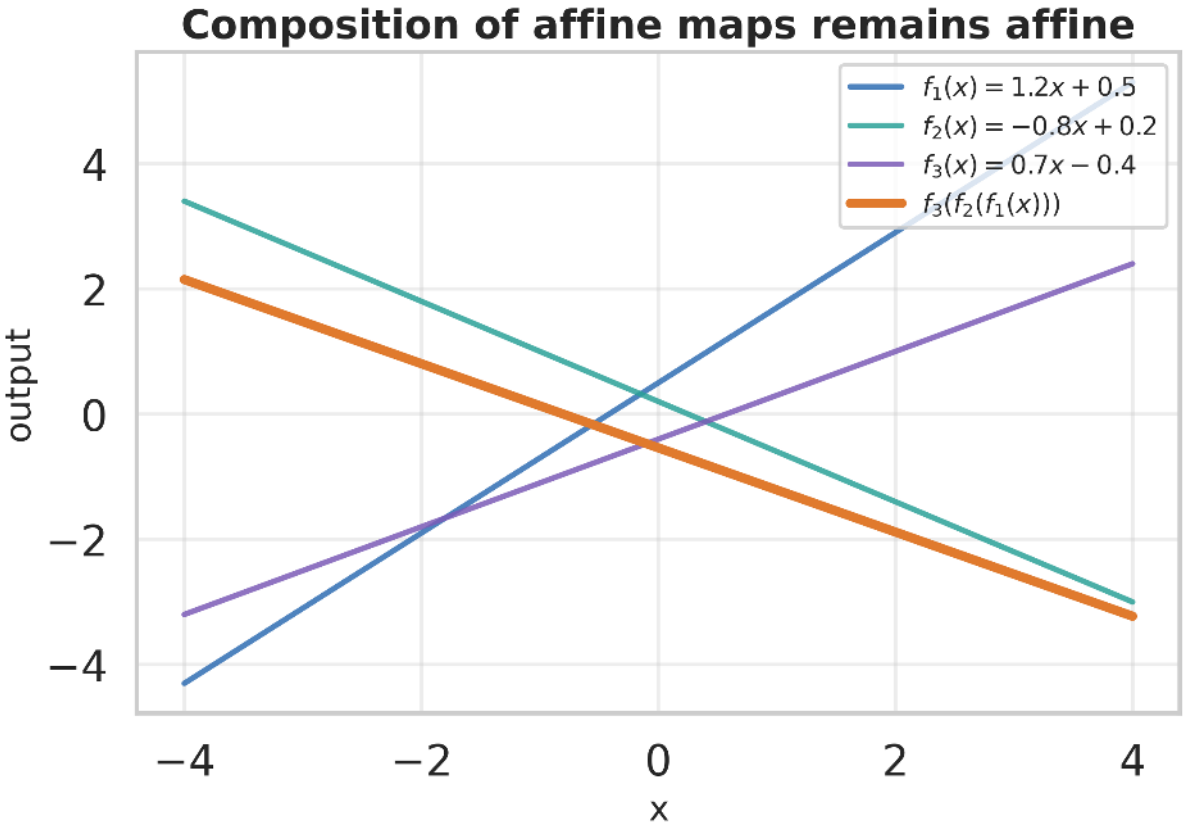
Stacking affine functions is still affine:

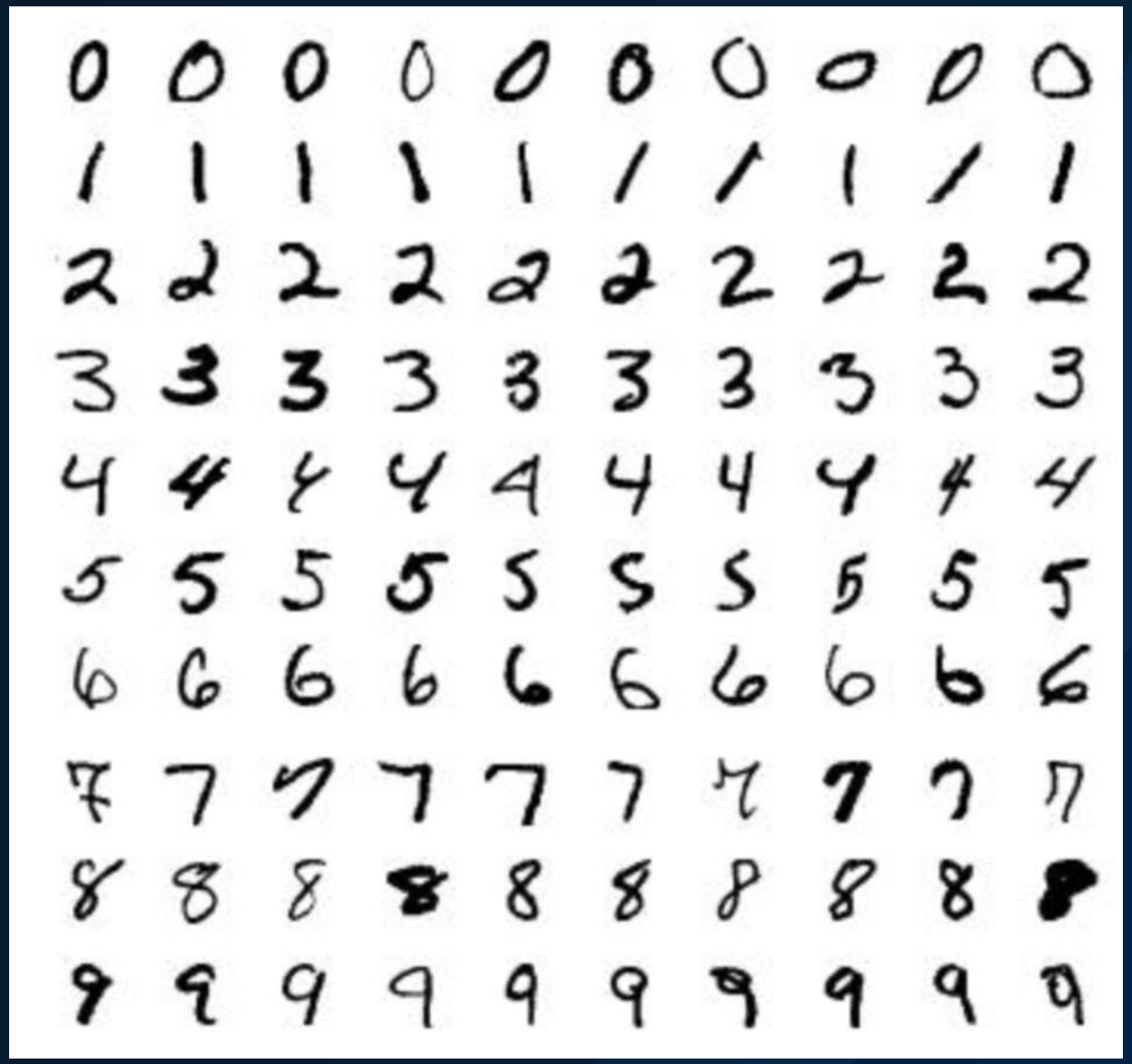
Let $f_1(x) = a_1x + b_1$ and $f_2(x) = a_2x + b_2$

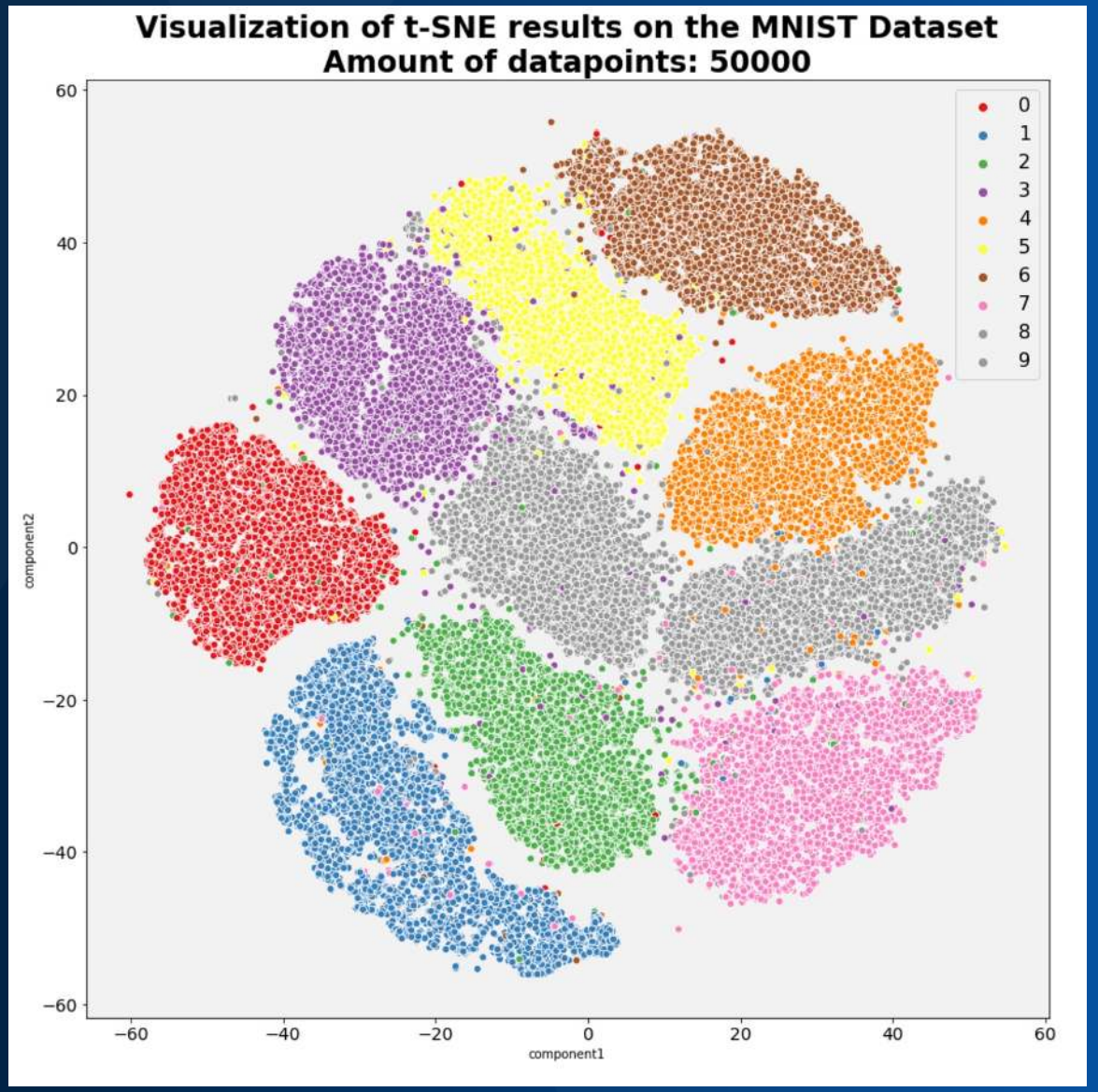
$$f_2\big(f_1(x)\big) = a_2(a_1x + b_1) + b_2$$

$$= (a_2a_1)x + (a_2b_1 + b_2) = mx + b$$

Depth alone does not create curvature





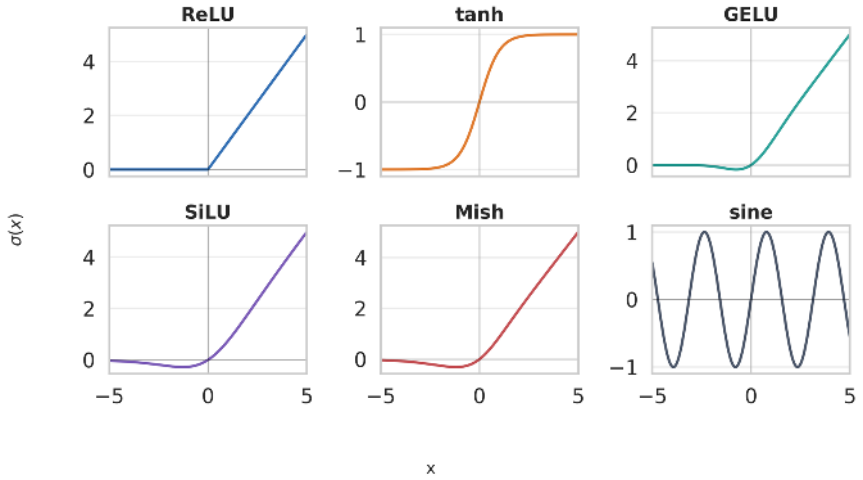

Visualization of t-SNE results on the MNIST Dataset
Amount of datapoints: 50000
component1
component2
0
1
2
3
4
5
6
7
8
9

Neural Network

A parameterized differentiable function.

Neural Network: $f_\theta\colon \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{\text{out}}}$

affine → nonlinear



Activation Functions

ReLU

$$\sigma(x) = \max(0, x) = \{\, 0, x < 0 \,;\, x, x \geq 0 \,\}$$

tanh

$$\sigma(x) = \tanh(x) = (e^{x} - e^{-x}) \,/\, (e^{x} + e^{-x})$$

GELU standard normal CDF

$$\sigma(x) = x\,\Phi(x) = 1/2\; x\,[1 + \operatorname{erf}(x/\sqrt{2})]$$

SiLU / Swish

$$= x \,/\, \sigma(x) = x\,\text{sigmoid}(x) \quad (1 + e^{-x})$$

Mish

$$\sigma(x) = x\tanh(\text{softplus}(x)) = x\tanh(\ln(1 + e^{x}))$$

Sine periodic activation

$$\sigma(x) = \sin(\omega_0\, x)$$

plot example: $\sin(2x)$

Neural Network

A parameterized differentiable function.

Neural Network: f_θ: $\mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{\text{out}}}$

affine → nonlinear

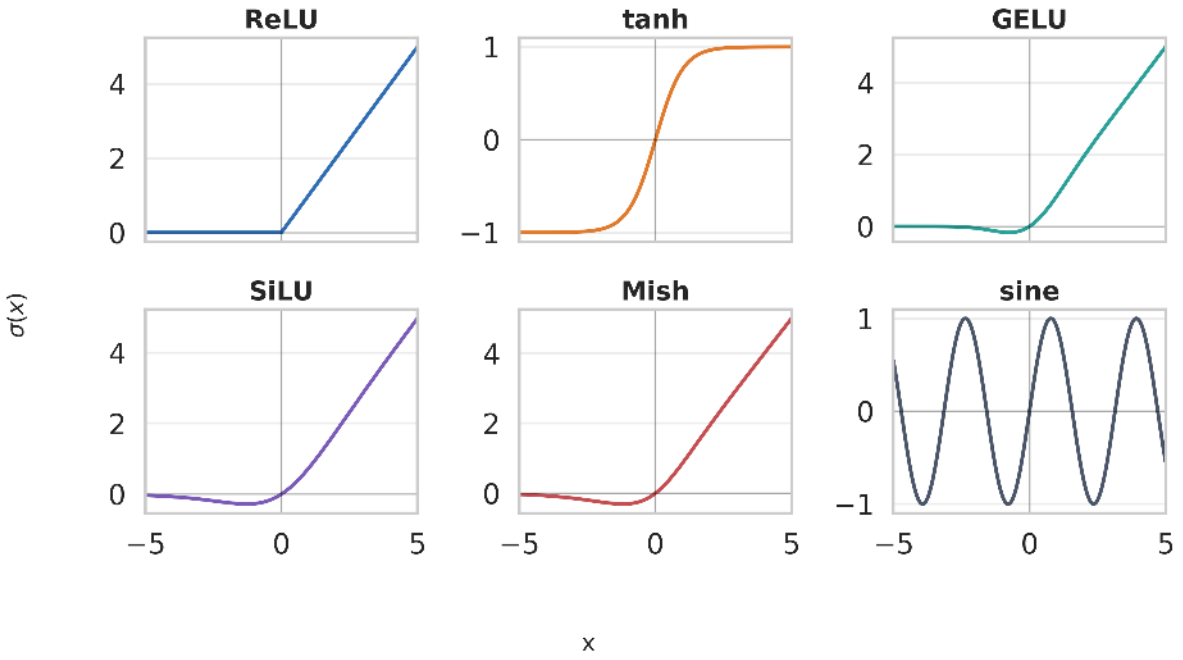
Activation Functions

ReLU: piecewise linear and sparse

tanh: smooth but saturating

GELU / SiLU / Mish: smooth gated

sine: periodic, useful in coordinate models



Neural Network

A parameterized differentiable function.

Neural Network: $f_\theta\colon \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{\text{out}}}$

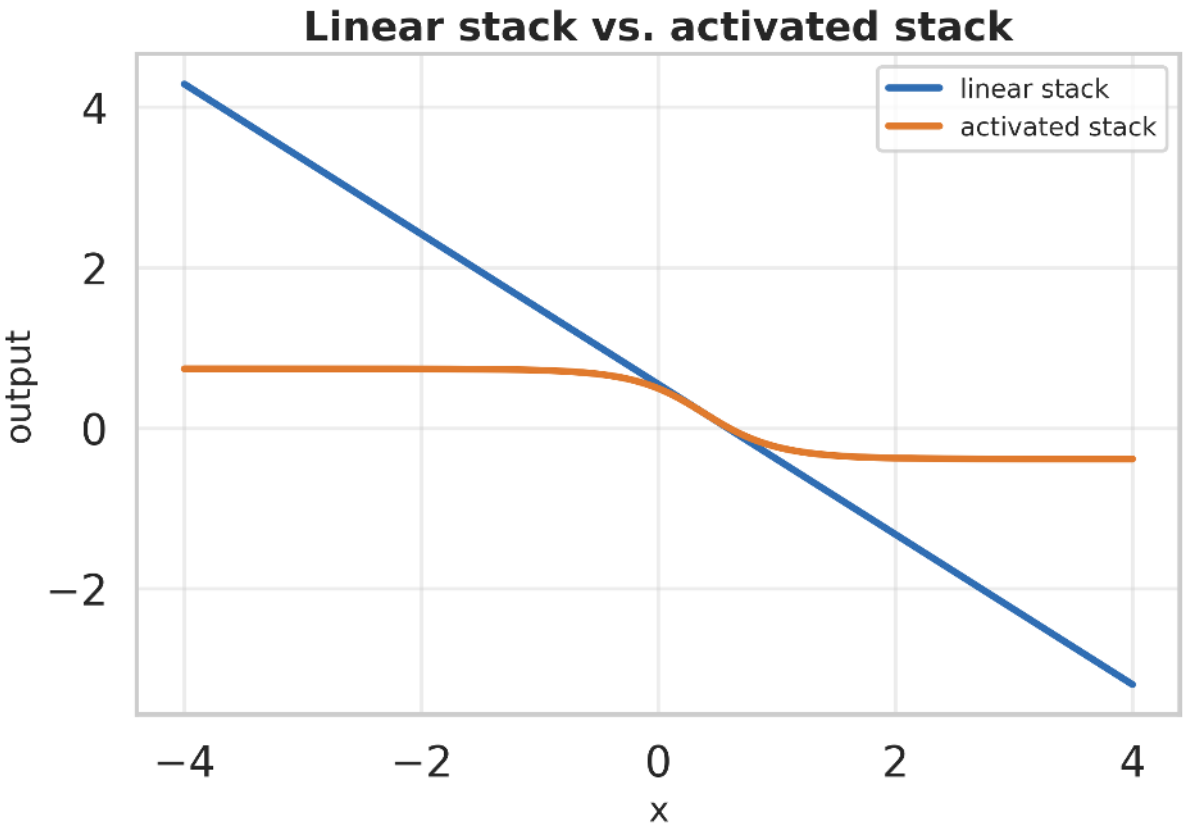
affine → nonlinear

$$f_1(x) = a_1 x + b_1 \qquad f_2(x) = a_2 x + b_2$$

$$f_3(x) = a_3 x + b_3$$

$$f_3(f_2(f_1(x))) = a_3(a_2(a_1 x + b_1) + b_2) + b_3$$

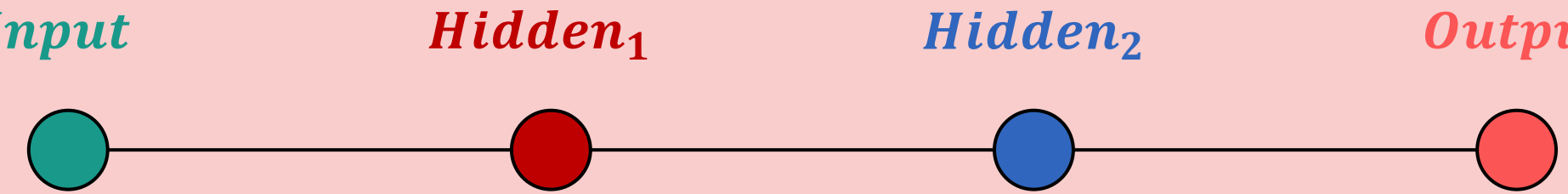
$$f_3(\sigma(f_2(\sigma(f_1(x))))) = a_3(\sigma(a_2(\sigma(a_1 x + b_1)) + b_2)) + b_3$$



Neural Network

A parameterized differentiable function.

Neural Network: $f_\theta\colon \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{\text{out}}}$



$$f_3(\delta(f_2(\delta(f_1(x))))) = a_3(\delta(a_2\delta((a_1x + b_1)) + b_2)) + b_3$$

$$f_\theta\colon \mathbb{R} \to \mathbb{R}$$

From Scalars to Structured Data

Extending simple 1D mappings toward vectors and matrices, used to represent realistic data.

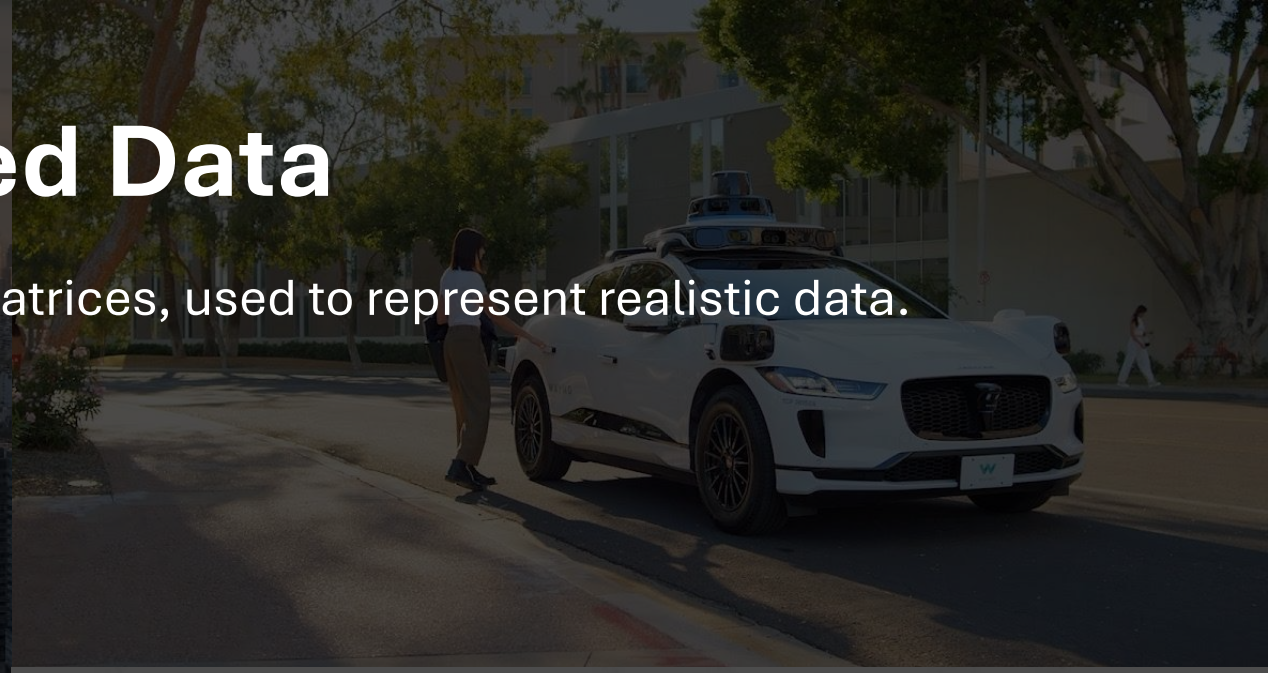
$$\mathbf{X}_{1D}^{\text{mast}} \in \mathbb{R}^{N_t \times N_z \times C} \qquad C = (U, \theta, T, p, TI)$$

$$\mathbf{X}_{2D}^{\text{ped}} \in \mathbb{R}^{N_t \times N_x \times N_y \times C} \qquad C = (u, v, w, U, \text{mask}, TI).$$

$$\mathbf{X}_{3D}^{\text{urban}} \in \mathbb{R}^{N_t \times N_x \times N_y \times N_z \times C} \qquad C = (u, v, w, T, p, q, \text{building mask}, \text{vegetation})$$

From Scalars to Structured Data

Extending simple 1D mappings toward vectors and matrices, used to represent realistic data.



$$\mathbf{X}_{3D}^{\text{urban}} \in \mathbb{R}^{N_t \times N_x \times N_y \times N_z \times C}$$

$$C = (u, v, w, T, p, q, \text{mask}, \text{vegetation})$$

$$\mathbf{X}_{2D}^{\text{image}} \in \mathbb{R}^{N_x \times N_y \times C}$$

$$C = (R, G\ B) \rightarrow \textit{color image}$$
$$C \in \{0, 1\} \rightarrow \textit{binary image}$$
$$C \in \{0, 1, 2, \ldots, 255\} \rightarrow \textit{grey image}$$

From Scalars to Structured Data

Extending simple 1D mappings toward vectors and matrices, used to represent realistic data.

Neural Network: $f_\theta : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{out}}$

Scalar line: $f(x) = ax + b = \hat{y}$ $\quad f: \mathbb{R} \rightarrow \mathbb{R}$

Vector-valued layer: $f(x) = Wx + b = \hat{y}$ $\quad f: \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{out}}$

$W \in \mathbb{R}^{d_{in} \times d_{out}}$

$b \in \mathbb{R}^{d_{out}}$

Neural Network

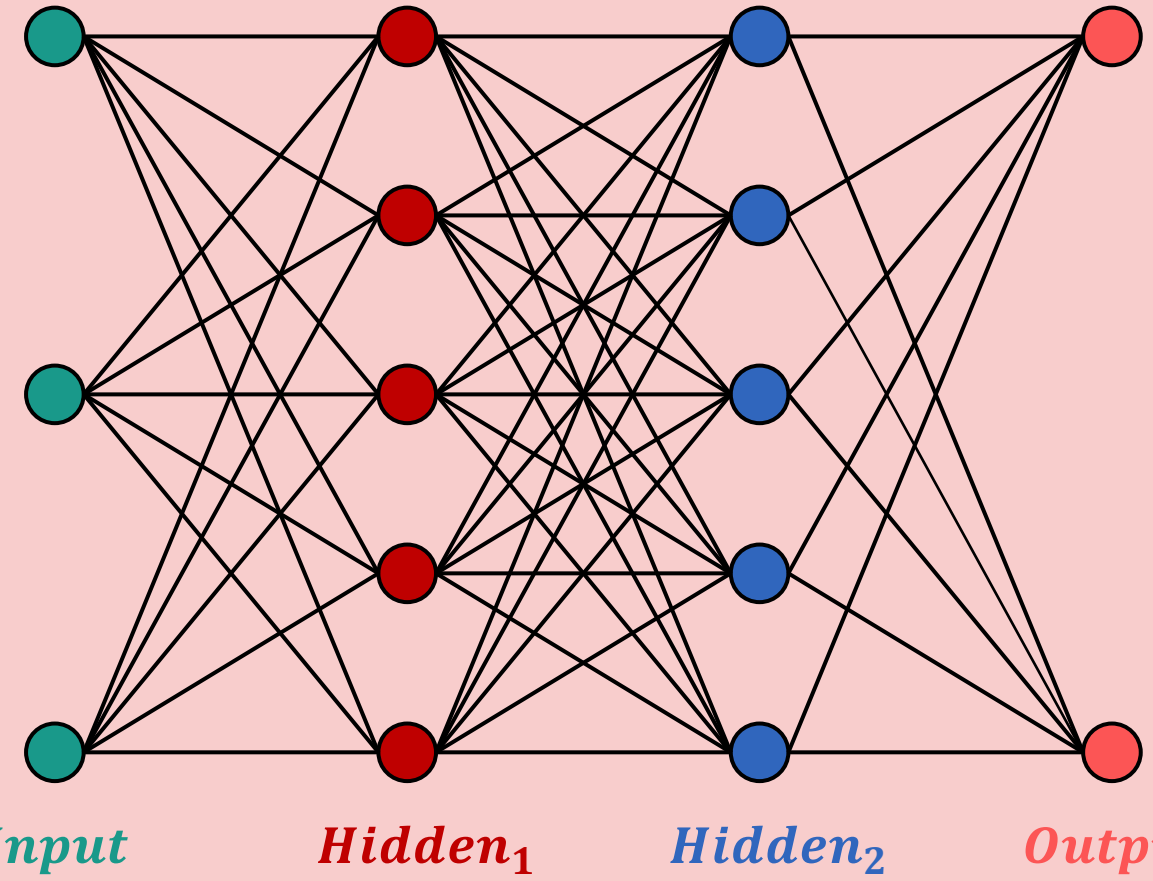
A parameterized differentiable function.

Neural Network: f_θ: $\mathbb{R}^{\text{din}} \rightarrow \mathbb{R}^{\text{dout}}$

$$z = Wx + b$$

$$x \in d_{in}$$
$$W \in \mathbb{R}^{d_{out} \times d_{in}}$$
$$b \in d_{out}$$

Parameter $\theta = \{W^{(l)}, b^{(l)}\}_{l=1}^{L}$



Number of Parameters

The number of parameters define how much neural network can learn.

W_1: $32 \cdot 1 = 32$ $\quad b_1 = 32$

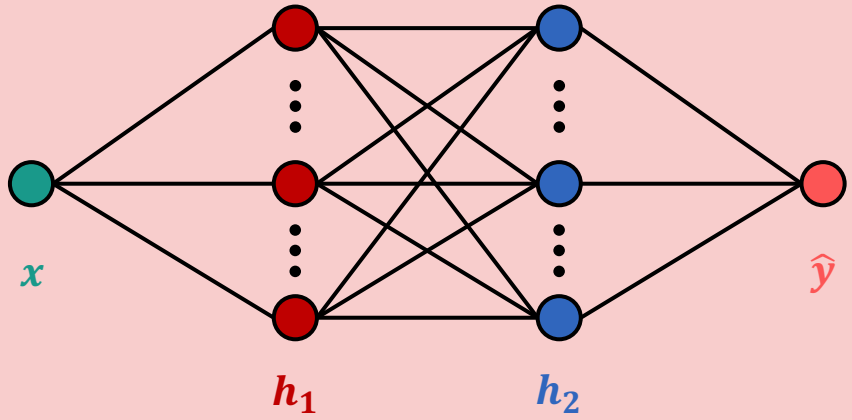
W_2: $32 \cdot 32 = 1024$ $\quad b_2 = 32$

W_3: $1 \cdot 32 = 32$ $\quad b_3 = 32$

$$32 + 32 + 1024 + 32 + 32 + 1 = \mathbf{1153}$$

W_1 b_1 W_2 b_2 W_3 b_3

Neural Network



$$f_\theta = A_3\left(\tanh\left(A_2\left(\tanh\left(A_1(x)\right)\right)\right)\right)$$

$A_1(x) = W_1(x) + b_1$ $\quad W_1 \in \mathbb{R}^{32\times 1}, b_1 \in \mathbb{R}^{32}$

$A_2(h_1) = W_2(h_1) + b_2$ $\quad W_2 \in \mathbb{R}^{32\times 32}, b_2 \in \mathbb{R}^{32}$

$A_3(h_2) = W_3(h_2) + b_3$ $\quad W_3 \in \mathbb{R}^{1\times 32}, b_3 \in \mathbb{R}$

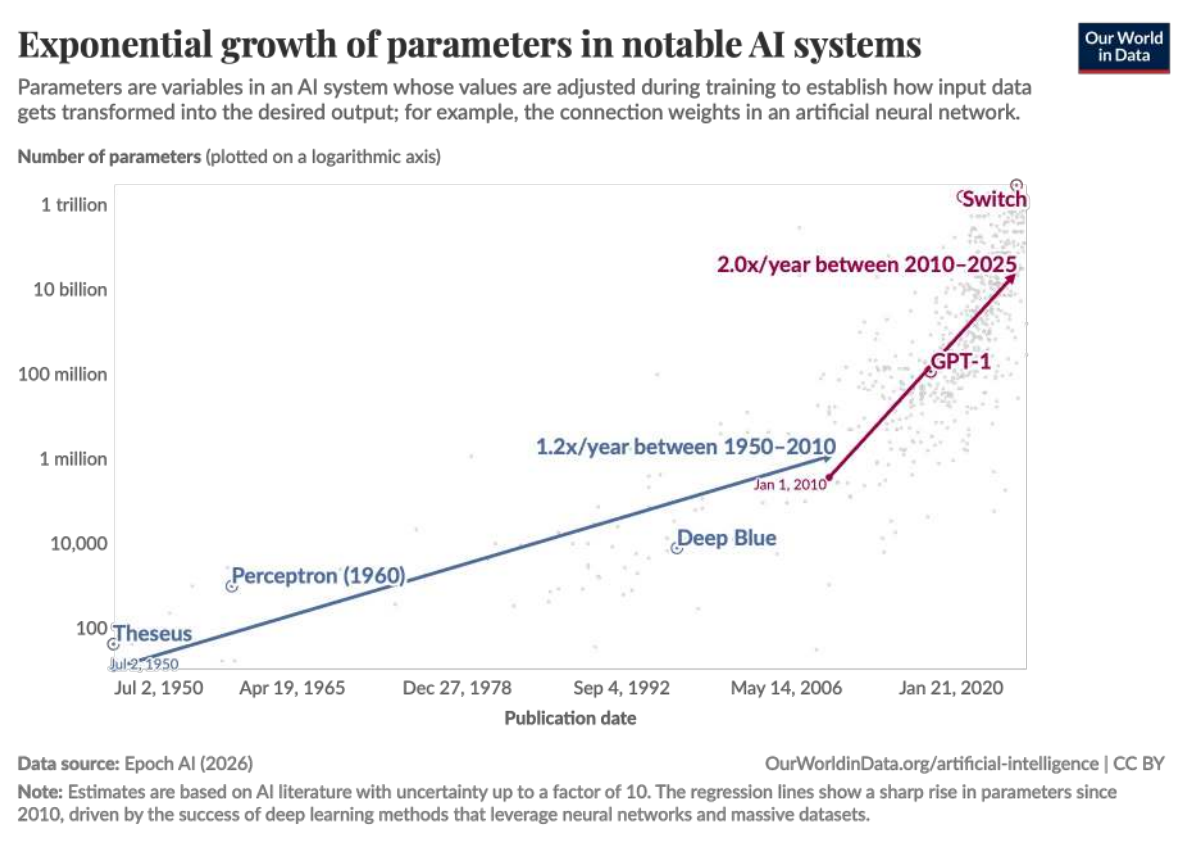
$h_1 = \tanh(W_1 x + b_1)$

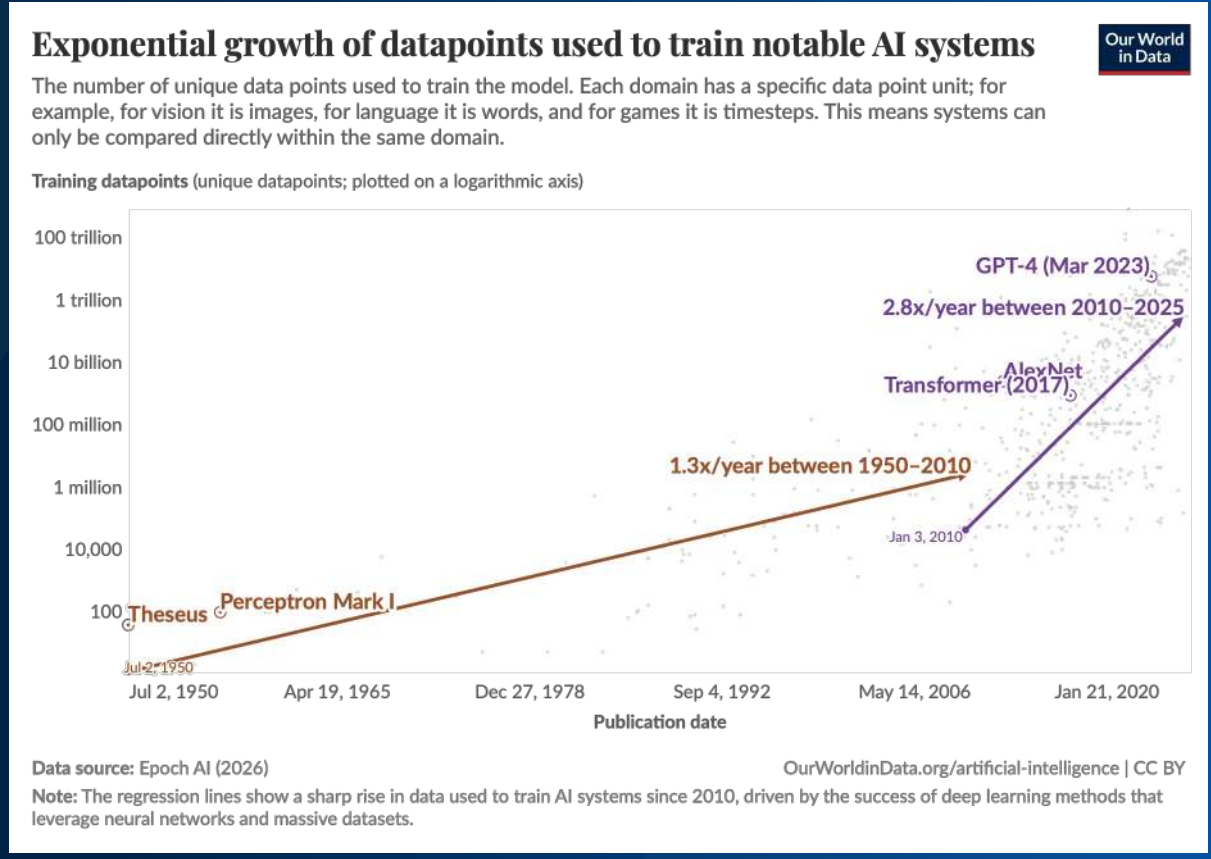
$h_2 = \tanh(W_2 x + b_2)$

$\hat{y} = W_3 h_2 + b_3)$

Number of Parameters

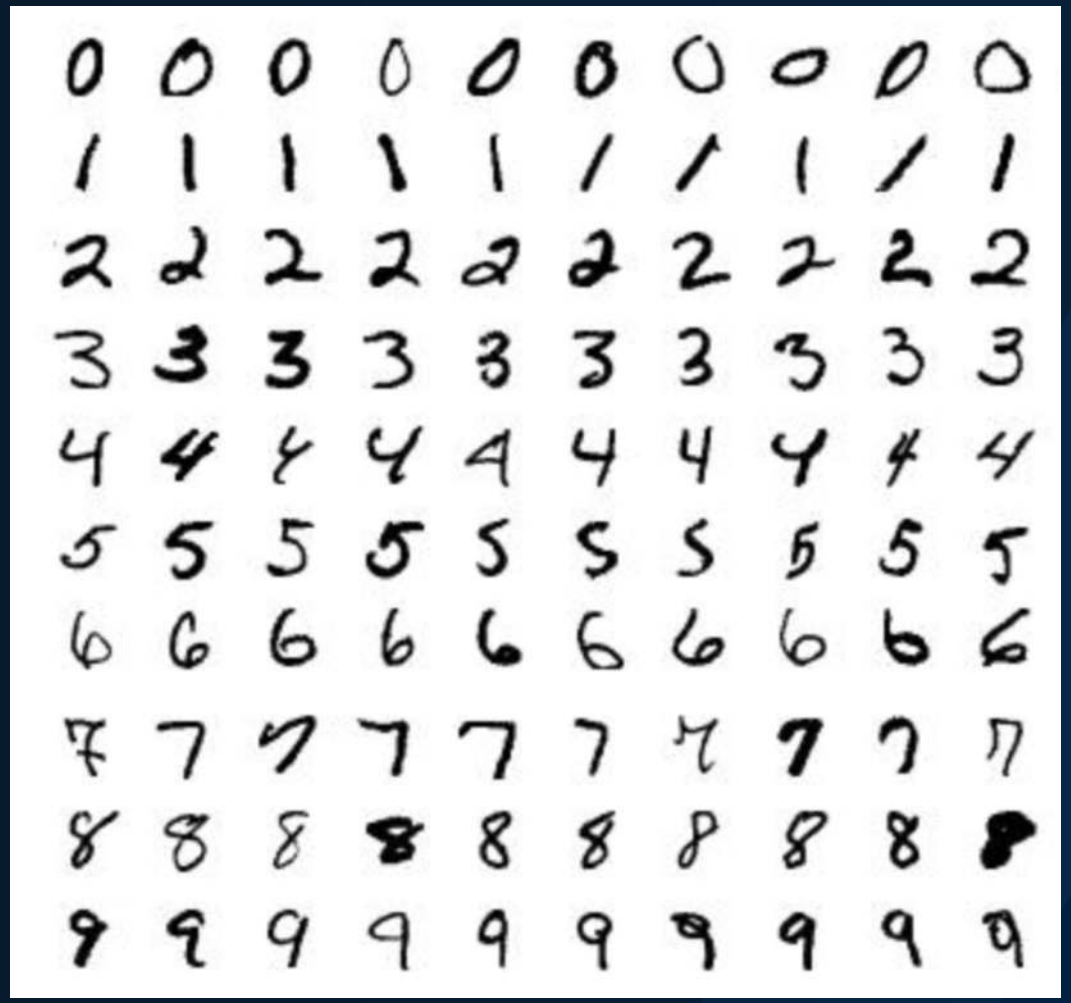
The number of parameters define how much neural network can learn.





From Pixels to Predictions?

Understanding how fully connected linear neural network turns raw input values into meaningful class predictions.



Each image in that dataset has 28×28 Pixels

Binary image channels $C \in \{0, 1\}$

So the dimension is: $\mathbf{X}_{2D}^{\text{image}} \in \mathbb{R}^{28 \times 28 \times 1}$

Transform the multi dimensional matrix into a multi dimensional vector

$$x \in \mathbb{R}^{28 \times 28 \times 1} \rightarrow 28 \cdot 28 \cdot 1 = 784$$

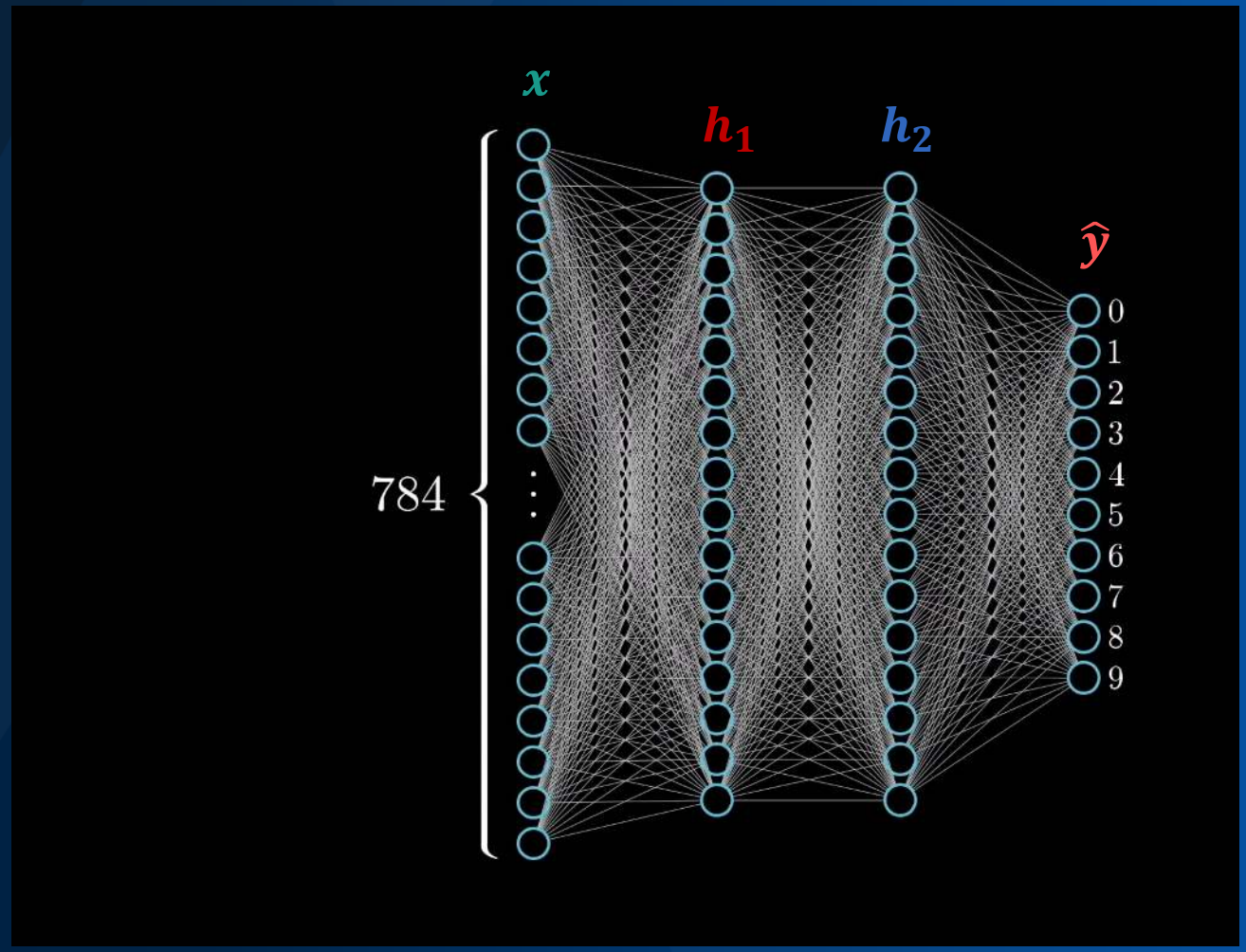
Each image in that dataset has 28×28 Pixels

Binary image channels $C \in \{0, 1\}$

So the dimension is: $\mathbf{X}_{2D}^{\text{image}} \in \mathbb{R}^{28 \times 28 \times 1}$

Transform the multi dimensional matrix into a multi dimensional vector

$x \in \mathbb{R}^{28 \times 28 \times 1} \rightarrow 28 \cdot 28 \cdot 1 = 784$



$$f_\theta = A_3\left(\text{relu}\left(A_2(\text{relu}(A_1(x)))\right)\right)$$

$$A_1(x) = W_1(x) + b_1$$

$$W_1 \in \mathbb{R}^{16\times 784}, b_1 \in \mathbb{R}^{16}$$

$$A_2(h_1) = W_2(h_1) + b_2$$

$$W_2 \in \mathbb{R}^{16\times 16}, b_1 \in \mathbb{R}^{16}$$

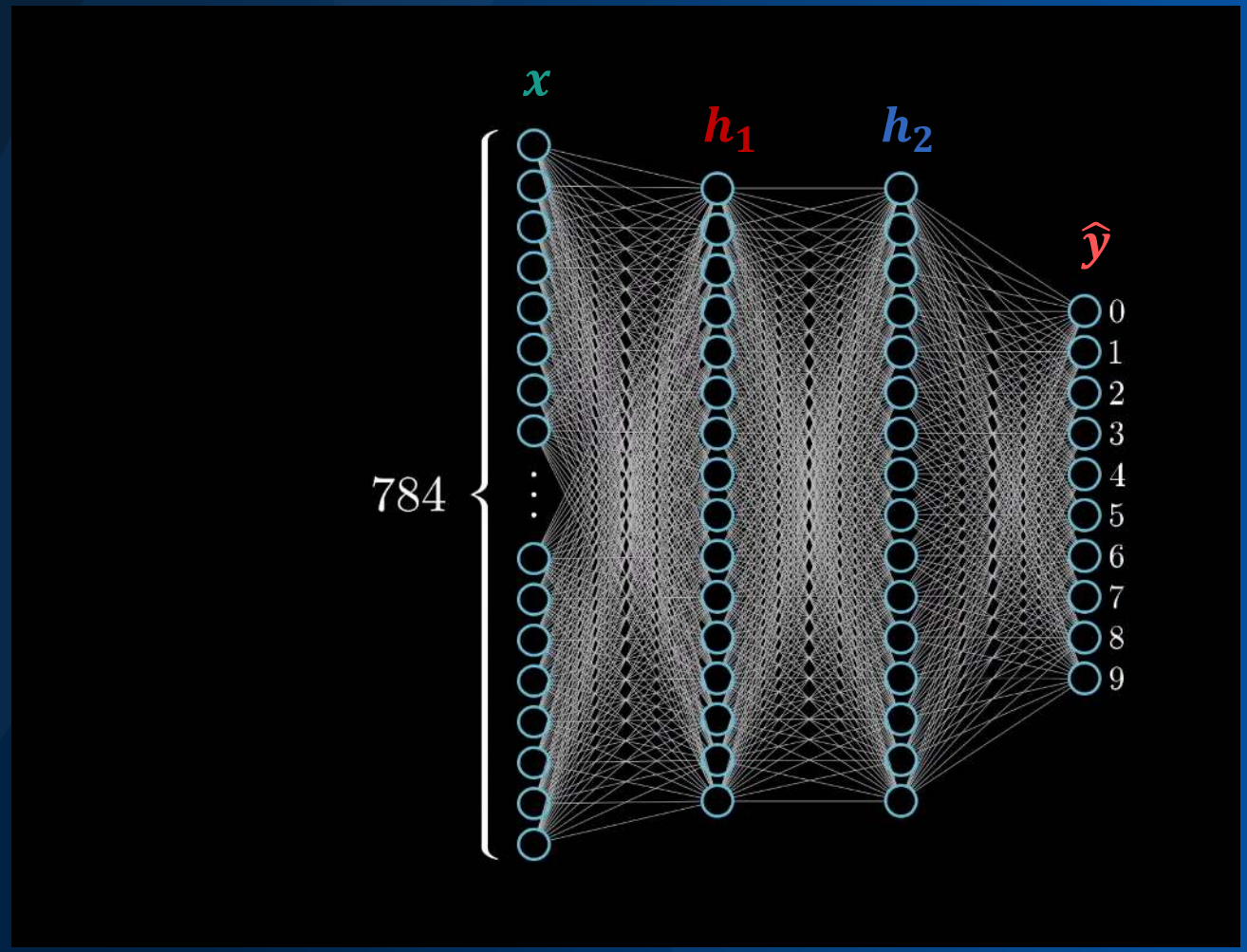
$$A_3(h_2) = W_3(h_2) + b_3$$

$$W_3 \in \mathbb{R}^{10\times 16}, b_3 \in \mathbb{R}^{10}$$

$$h_1 = \text{relu}(W_1 x + b_1)$$

$$h_2 = \text{relu}(W_2 x + b_2)$$

$$\hat{y} = W_3 h_2 + b_3$$



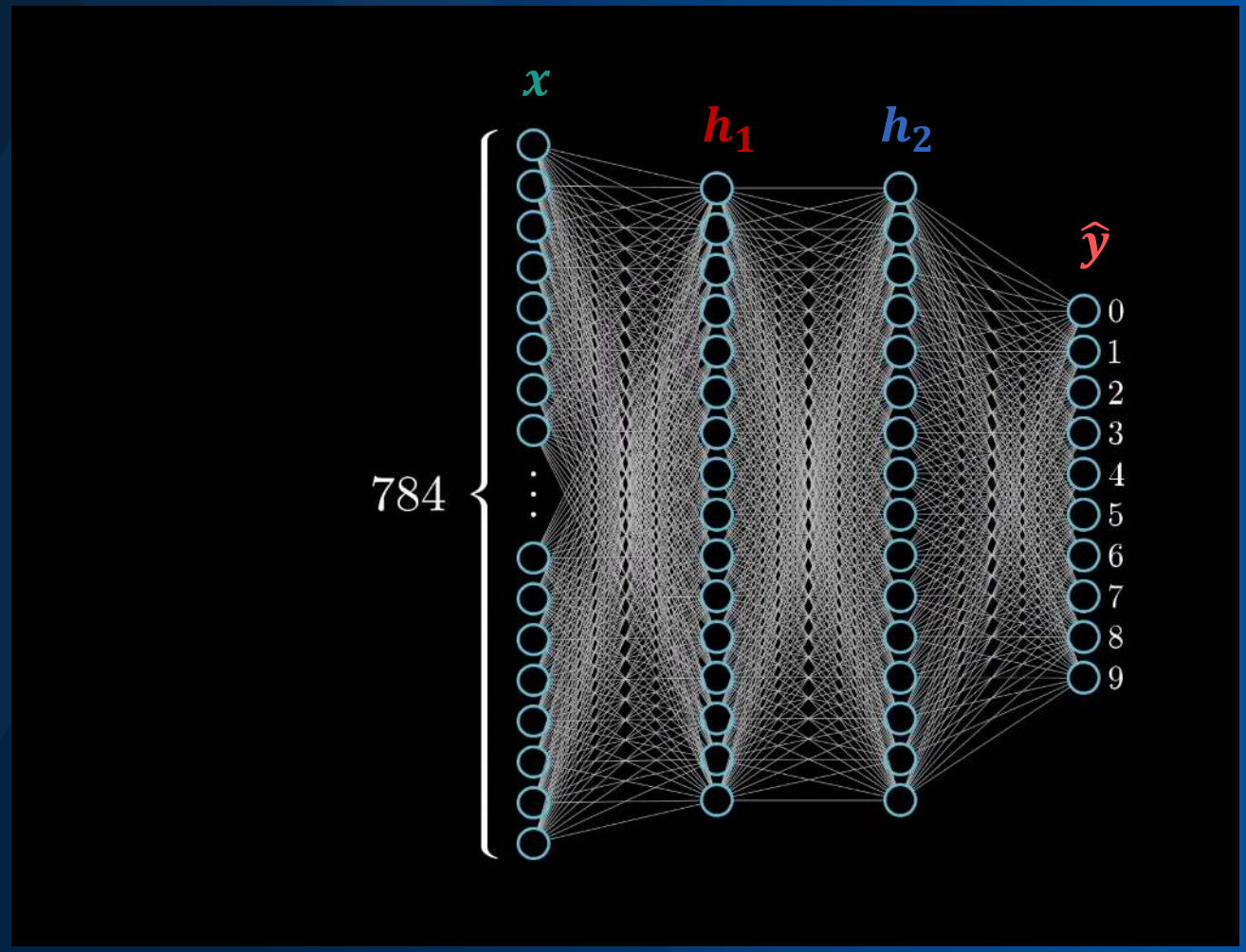
$$f_\theta = A_3\left(\text{relu}\left(A_2\left(\text{relu}\left(A_1(x)\right)\right)\right)\right)$$

$W_1 \in \mathbb{R}^{16\times 784}, b_1 \in \mathbb{R}^{16}$

$W_2 \in \mathbb{R}^{16\times 16}, b_1 \in \mathbb{R}^{16}$

$W_3 \in \mathbb{R}^{10\times 16}, b_3 \in \mathbb{R}^{10}$

$$784 \cdot 16 + 16 + 16 \cdot 16 + 16 + 16 \cdot 10 + 10$$
$$= 12544 + 16 + 256 + 16 + 160 + 10$$
$$= \mathbf{13002}$$



Can a feedforward network approximate any function?

Can a feedforward network approximate any function?

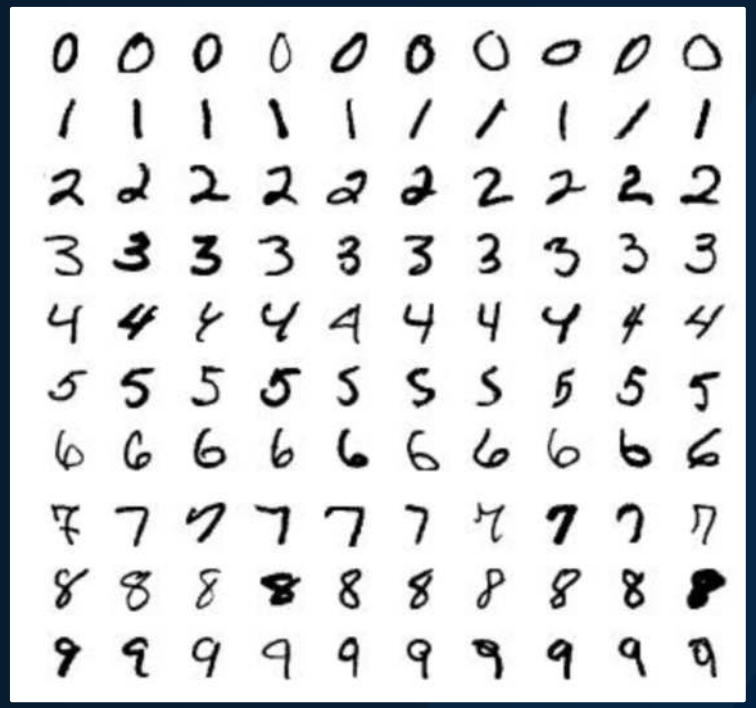
Theoretically yes

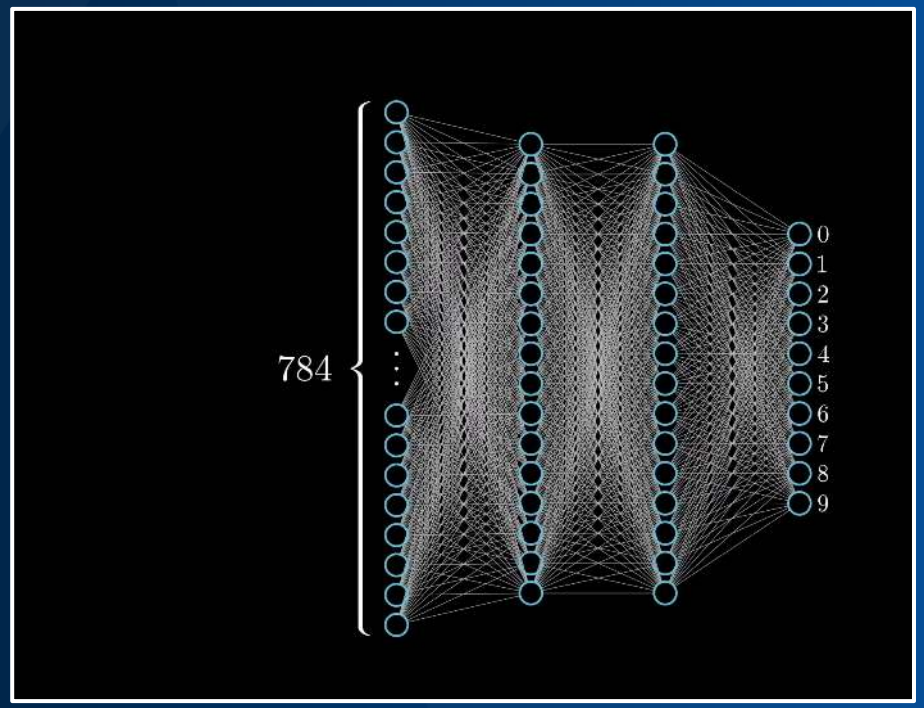
Universal Approximation Theorem: A feedforward neural network with nonlinear activations and sufficiently many neurons can approximate any continuous function on a bounded domain arbitrarily well.

Practically no

Universal approximation only guarantees that a suitable network exists. It does not guarantee enough data, feasible model size, successful training, or reliable generalization.

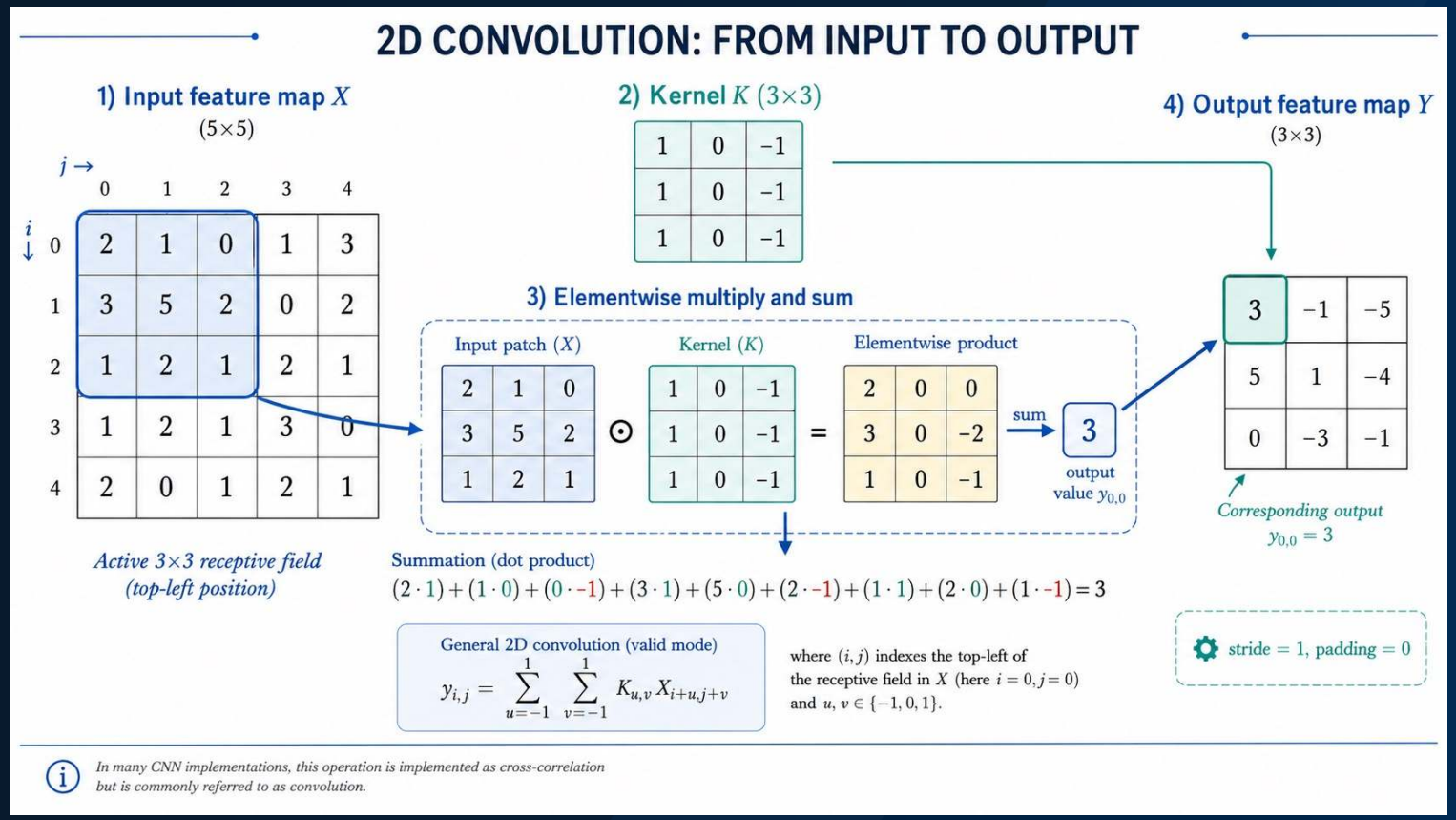
Which information do we lose in this setup?

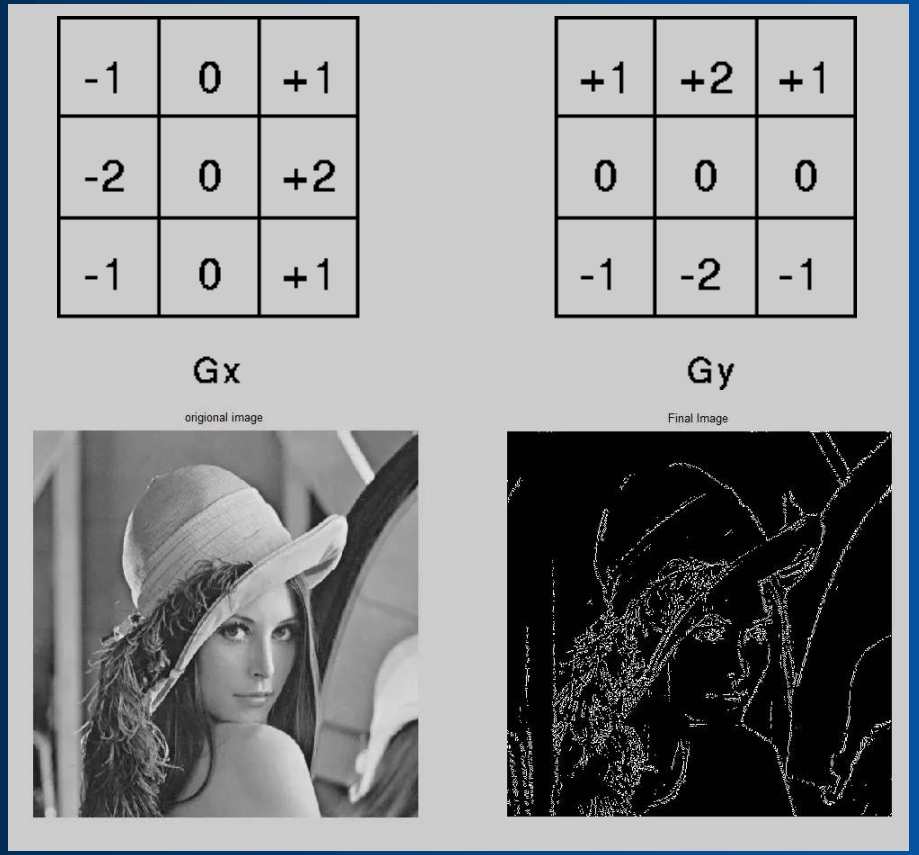




Grid Valued Input Fields

Images and CFD states are functions on a discrete grid.





Grid Valued Input Fields

Images and CFD states are functions on a discrete grid.

$$\Omega_h = \{(i,j): i = 1, \dots, N_x, j = 1, \dots, N_y\}$$

$$x: \Omega_h \rightarrow \mathbb{R}^{C_{in}}$$

$$x(i,j) \in \mathbb{R}^{C_{in}} \qquad x(i,j) = \begin{bmatrix} R(i,j) \\ G(i,j) \\ B(i,j) \end{bmatrix} \qquad x(i,j) = \begin{bmatrix} u(i,j) \\ v(i,j) \\ w(i,j) \\ p(i,j) \end{bmatrix}$$

$$y: \Omega_h \rightarrow \mathbb{R}^{C_{out}}$$

C_{out} is the number of output channels, also called the number of filters or feature maps

Grid Valued Input Fields

Images and CFD states are functions on a discrete grid.

Convolutional produces feature maps. For a 2D convolution centered at (i, j), the kernel uses neighboring values $x(i + p, j + q)$ where $p, q \in \{-r, \dots, r\}$. So r controls the spatial reach of the local neighborhood.

For a kernel size $k \times k$, we have $k = 2r + 1$.

$r = 1 \implies k = 3$

$r = 2 \implies k = 5$

$r = 3 \implies k = 7$

K is the learnable convolution kernel. It contains the weights of the convolutional layer.

$K \in \mathbb{R}^{C_{out} \times C_{in} \times k \times k}$ $\quad b \in \mathbb{R}^{C_{out}}$

Grid Valued Input Fields

Images and CFD states are functions on a discrete grid.

K is the learnable convolution kernel. It contains the weights of the convolutional layer.

$$K \in \mathbb{R}^{C_{out} \times C_{in} \times k \times k}, \qquad b \in \mathbb{R}^{C_{out}}$$

Each weight is a scalar value and indexed by

$$K_{C_{out},c_{in},p,q}$$

$$y_{C_{out}}(i,j) = b_{C_{out}} + \sum_{c_{in}=1}^{C_{in}} \sum_{p=-r}^{r} \sum_{q=-r}^{r} K_{C_{out},c_{in},p,q} x_{c_{in}}(i+p, j+q)$$

$$y = K * x + b$$

$$h = \sigma(K * x + b)$$

$$h^{(l+1)} = \sigma(K^{(l)} * h^{(l)} + b^{(l)})$$

What do standard layers learn?

Linear and convolutional layers learn transformations between fixed-size arrays.

Layer type	Mathematical view	What it learns	Limitation
Linear layer	$(y = Wx + b)$	global mixing of all input values	fixed input size
Convolution	$(y = K * x + b)$	local spatial patterns	local receptive field
Deep CNN	stacked convolutions	larger spatial dependencies	tied to grid resolution

Same Physics, Different Grids

The flow field is continuous in space. The grid is only the numerical representation we choose.

Convolutions are a good first step because they respect the grid. But in CFD, the grid is not the physical object itself.

A linear layer $y = Wx + b$ maps one finite dimensional vector to another:

$$f_\theta : \mathbb{R}^n \to \mathbb{R}^m$$

A convolution layer $y = W * x + b$ maps one finite dimensional grid valued tensor to another:

$$f_\theta : \mathbb{R}^{N_x, N_y, C_{in}} \to \mathbb{R}^{N_x^*, N_y^*, C_{out}}$$

The flow field is continuous in space. The grid is only the numerical representation we choose.

From Array Maps to Field Maps

Neural operators learn mappings between physical fields, not only between fixed-size arrays.

$$\text{Array} \rightarrow \text{Array} \qquad f_\theta : \mathbb{R}^{N_x, N_y, C_{in}} \rightarrow \mathbb{R}^{N_x^*, N_y^*, C_{out}}$$

$$\text{Function} \rightarrow \text{Function} \qquad \mathcal{G}_\theta : a(x) \rightarrow u(x)$$

From Numerical Solvers to Learned Surrogates

The learning task is to approximate a solution map.

A PDE problem can be written abstractly as $\mathcal{N}(u; a) = 0$ in Ω with boundary and initial conditions.

The solver defines an operator $\mathcal{G}$: $a \mapsto u$; the model learns $\mathcal{G}\theta(a) \approx \mathcal{G}(a)$.

Neural Operator

Learning maps between function spaces

A neural operator does **not** learn a map from one vector to another vector.
It learns a map from one **function** to another **function**.

For PDEs, this means it learns the **solution operator** of a family of PDEs.

Neural Operator

Parametric PDE and Solution Operator

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

$$u(x) = 0 \qquad x \in \partial \mathrm{D}$$

$PDE\ Parameter$: $a \in \mathcal{A}$

$PDE\ Solution$: $u \in \mathcal{U}$

Learned neural operator: $\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}$

$\mathcal{G}_\theta(a) \approx u$

A classical neural network usually learns a finite-dimensional map:

$$f_\theta : \mathbb{R}^{\mathrm{n}} \to \mathbb{R}^{\mathrm{m}}$$

Neural Operator

The PDE induces a mapping from parameters to solutions

For every admissible input function a, the PDE has a corresponding solution u.
This defines the true solution operator.

$$\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$$

$$u = \mathcal{G}^\dagger(a)$$

The operator $\mathcal{G}^\dagger$ is the exact but usually unknown PDE solution operator.
The neural operator aims to approximate this mapping.

Neural Operator

Approximating the true PDE solution map

The goal is to train a neural operator $\mathcal{G}_\theta$ such that it approximates the true solution operator $\mathcal{G}^\dagger$.

$$\mathcal{G}_\theta \approx \mathcal{G}^\dagger$$

$$\mathcal{G}_\theta(a) \approx u$$

Once trained, the neural operator predicts the full solution field for a new input field a in one forward pass.

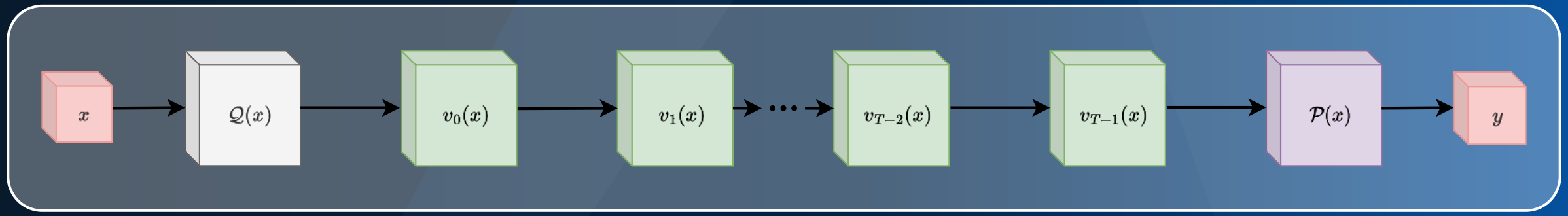
Neural Operator

Architecture: Lifting, operator layers, and projection

A neural operator first lifts the input field to a higher-dimensional feature space, applies several operator layers, and then projects the hidden representation back to the physical output field.

$$\mathcal{G}_\theta = \mathcal{P} \circ \mathcal{v}_{\mathrm{T}-1} \circ \ldots \circ \mathcal{v}_{\mathrm{T}-2} \circ \mathcal{v}_1 \circ \mathcal{Q}$$

$$\mathcal{G}_\theta(\mathrm{a}) = \mathcal{P}(\mathcal{v}_{\mathrm{T}-1}(\mathcal{v}_{\mathrm{T}-2}(\ldots \mathcal{v}_1(\mathcal{Q}(\mathrm{a})))))$$



Neural Operator

Architecture: Lifting, operator layers, and projection

Start with input field:
$a: \Omega \rightarrow \mathbb{R}^{C_{in}}$

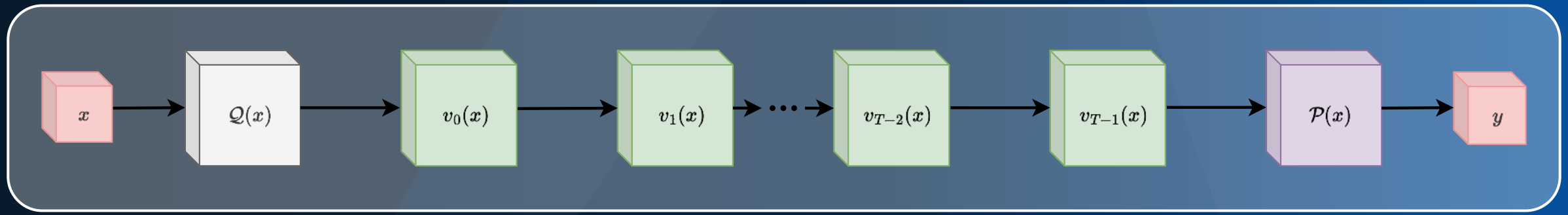
Up-projection maps it to latent field:
$z_0(x) = Q(a(x))$

$z_0: \Omega \rightarrow \mathbb{R}^d$

Apply several Neural Operator blocks
$z_{t+1}(x) = v_t(z_t)$

$t = 0, \ldots, T-2$

Project back to output channel:
$u_\theta(x) = P(z_{T-1}(x))$



Neural Operator

Uplifting: Changing the channel dimension of the field

Assume the input is a field over a spatial domain Ω:

$$\mathrm{a}: \rightarrow \mathbb{R}^{C_{in}}$$

For every spatial location $x \in \Omega$, the input field has C_{in} channels.

$$\mathrm{a(x)} = \begin{bmatrix} u_{in}(x) \\ v_{in}(x) \\ w_{in}(x) \\ mask(x) \end{bmatrix} \in \mathbb{R}^4$$

The up-projection Q maps the input channels into a higher-dimensional latent feature space:

$$Q: \mathbb{R}^{C_{in}} \rightarrow \mathbb{R}^{d}$$

Applied pointwise over the domain:

$$\mathrm{z}_0(x) = Q\big(a(x)\big)$$

So the dimensions change as:

$$\mathrm{a(x)} \in \mathbb{R}^{C_{in}} \rightarrow \mathrm{z}_0(x) \in \mathbb{R}^{d}$$

Neural Operator

Uplifting: Changing the channel dimension of the field

Assume the input is a field over a spatial domain Ω:

$$a: \rightarrow \mathbb{R}^{C_{in}}$$

Concrete CFD example:

$$a_h \in \mathbb{R}^{200 \times 200 \times 50 \times 7}$$

$C_{in} = 7$
with $[u_{in}, v_{in}, w_{in}, p, q, mask, terrain]$

The up-projection Q maps the input channels into a higher-dimensional latent feature space:

$$Q: \mathbb{R}^{C_{in}} \rightarrow \mathbb{R}^{d}$$

Up projection with $d = 64$:

$$Q: \mathbb{R}^{200 \times 200 \times 50 \times 7} \rightarrow \mathbb{R}^{200 \times 200 \times 50 \times 64}$$

Neural Operator

Neural operator blocks keep the latent field shape but update each feature using spatial interactions across the domain.

$$z_t : \Omega \to \mathbb{R}^d$$

$$z_{t+1} = v_t(z_t)$$

$$v_t : \mathbb{R}^d \to \mathbb{R}^d$$

$$v_t : (\Omega \to \mathbb{R}^d) \to (\Omega \to \mathbb{R}^d)$$

$$z_{t+1}(x) = \sigma(W_t v_t(x) + K_t v_t(x) + b_t)$$

Neural Operator

Neural operator blocks keep the latent field shape but update each feature using spatial interactions across the domain.

Concrete CFD example:

$$a_h \in \mathbb{R}^{200 \times 200 \times 50 \times 7}$$

$C_{in} = 7$
with $[u_{in}, v_{in}, w_{in}, p, q, mask, terrain]$

Up projection with $d = 64$:

$$\mathcal{Q}: \mathbb{R}^{200 \times 200 \times 50 \times 7} \rightarrow \mathbb{R}^{200 \times 200 \times 50 \times 64}$$

$$z_t \in \mathbb{R}^{N_x \times N_y \times N_z \times d}$$

$$z_{t+1} = \mathcal{v}_t(z_t)$$

$$z_{t+1} \in \mathbb{R}^{N_x \times N_y \times N_z \times d}$$

$$z_t \in \mathbb{R}^{200 \times 200 \times 50 \times 64}$$

$$\mathcal{v}_t: \mathbb{R}^{200 \times 200 \times 50 \times 64} \rightarrow \mathbb{R}^{200 \times 200 \times 50 \times 64}$$

$$z_{t+1} \in \mathbb{R}^{200 \times 200 \times 50 \times 64}$$

Neural Operator

The down-projection converts the latent field into the desired output channels, such as velocity and pressure.

$$z_{T-1}: \Omega \rightarrow \mathbb{R}^d$$

$$u_\theta(x) = \mathcal{P}(z_{T-1}(x))$$

$$\mathcal{P}: \mathbb{R}^d \rightarrow \mathbb{R}^{C_{out}}$$

$$\mathcal{P}: (\Omega \rightarrow \mathbb{R}^d) \rightarrow (\Omega \rightarrow \mathbb{R}^{C_{out}})$$

Neural Operator

The down-projection converts the latent field into the desired output channels, such as velocity and pressure.

Concrete CFD example:

$$a_h \in \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 7}$$
$$u_h \in \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 3}$$

$C_{in} = 7$
with $[u_{in}, v_{in}, w_{in}, p, q, \text{mask}, \text{terrain}]$

$C_{out} = 3$
with $[u_{out}, v_{out}, w_{out}]$

Up projection with $d = 64$:

$$\mathcal{Q}: \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 7} \rightarrow \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 64}$$

Neural Operator Blocks with:

$$\mathcal{v}_t: \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 64} \rightarrow \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 64}$$

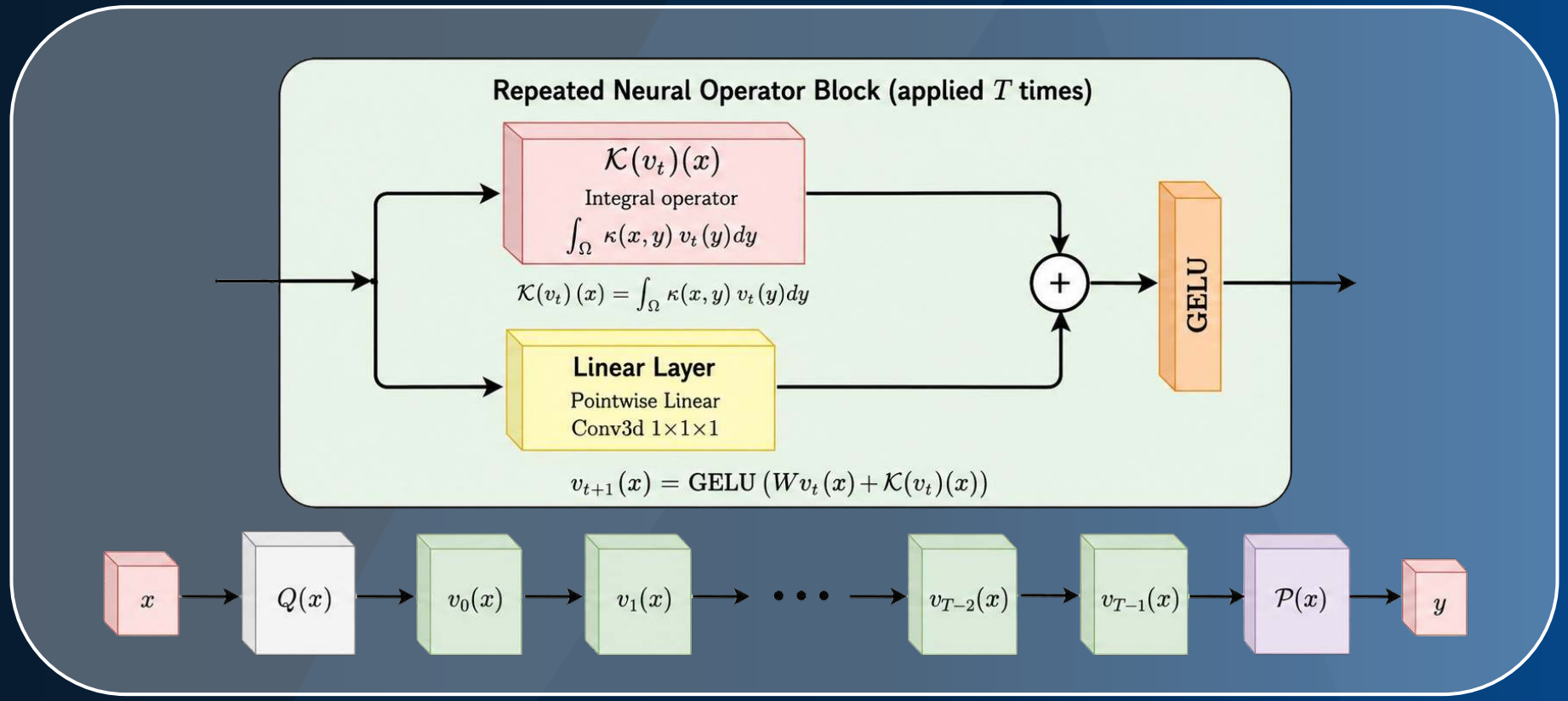
$$\mathcal{v}_{T-1} \in \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 64}$$

Down projection with $d = 3$:

$$\mathcal{P}: \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 64} \rightarrow \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 3}$$

Neural Operator

Architecture: Lifting, operator layers, and projection



From Neural Operators to Fourier Neural Operators

The main difference between neural operator variants is how they compute the nonlocal transformation inside each block.

Fourier Neural Operator

The nonlocal operator is computed in Fourier space by learning transformations of global frequency modes.

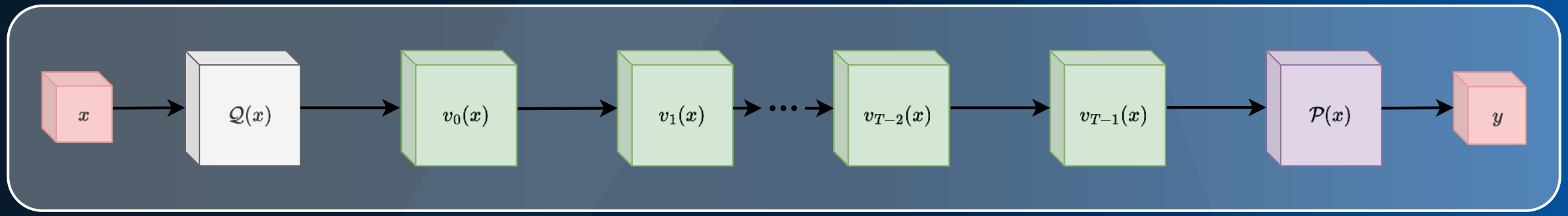
$$v_{t+1}(x) = \sigma(W_t v_t(x) + K_t v_t(x))$$

Generic Neural Operator

$$K_t v_t(x) = \int K_t(x, y) v_t(y)\, dy$$

Fourier Neural Operator

$$K_t(v_t = \mathcal{F}^{-1}(\mathcal{R}_t \cdot \mathcal{F}(v_t))$$



Fourier Neural Operator

The spatial grid and latent width stay fixed, but the latent field is temporarily represented in Fourier space.

Start with a latent space

$$v_t \in \mathbb{R}^{N_x \times N_y \times N_z \times d}$$

Fourier Transform

$$\mathcal{F}(v_t) = \hat{v}_t \in \mathbb{C}^{N_x \times N_y \times N_z \times d}$$

Mode Truncation

$$\hat{v}_t \rightarrow \hat{v}_t^M \in \mathbb{C}^{M_x \times M_y \times M_z \times d}$$

Learned Spectral Transformation

$$R_t \cdot \hat{v}_t^M$$

with

$$R_t \in \mathbb{C}^{M_x \times M_y \times M_z \times d \times d}$$

Inverse Fourier Transform

$$\mathcal{F}^{-1}(R_t \cdot \hat{v}_t^M) \in \mathbb{R}^{N_x \times N_y \times N_z \times d}$$

Fourier Block output

$$v_{t+1} \in \mathbb{R}^{N_x \times N_y \times N_z \times d}$$

Why Fourier Space?

Many physical fields contain large-scale spatial structures that can be represented efficiently through low-frequency modes.

Fourier Modes

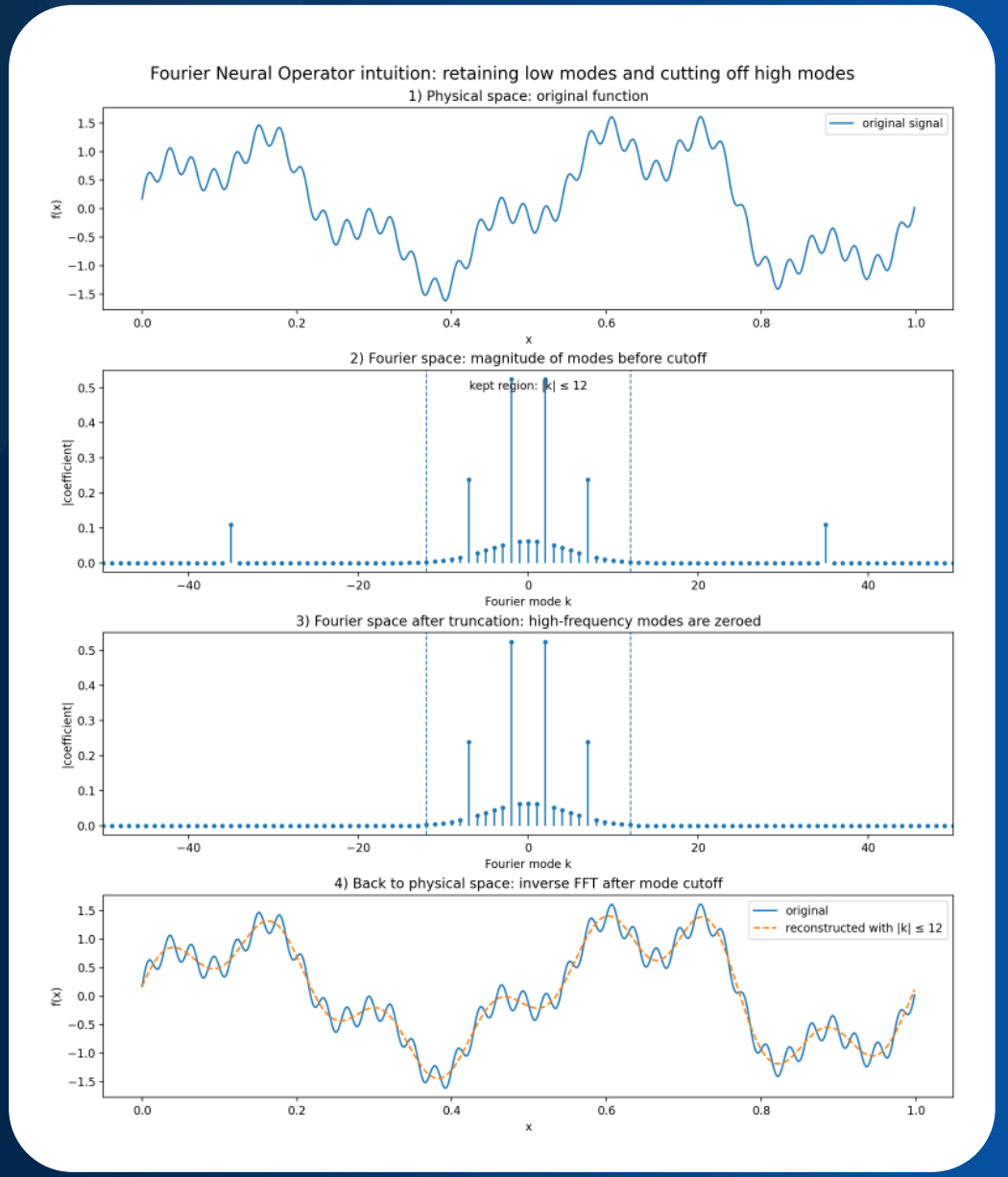
Retaining low-frequency Fourier modes while truncating high-frequency details to reduce complexity.

In an FNO layer, the high-frequency modes get cut off:

$$f(x) \rightarrow \mathcal{F}(f) \text{ keep only } |k| \leq k_{max} \rightarrow \mathcal{F}^{-1}$$

So this acts like a **learned low-pass spectral operator**.
The high modes are not used in the Fourier branch.

The representation changes to Fourier space, but the block output keeps the same latent grid shape.



Fourier Neural Operator

Concrete CFD example:

$$a_h \in \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 7}$$
$$u_h \in \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 3}$$

$C_{in} = 7$
with $[u_{in}, v_{in}, w_{in}, p, q, \text{mask}, \text{terrain}]$

$C_{out} = 3$
with $[u_{out}, v_{out}, w_{out}]$

Up projection with $d = 64$:

$$\mathcal{Q}: \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 7} \rightarrow \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 64}$$

Fourier Neural Operator Blocks with:

$$v_t: \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 64}$$

3D Fourier Transform
$$\mathcal{F}(v_t) = \hat{v}_t \in \mathbb{C}^{200 \times 200 \times 200 \times 64}$$

Keep selected Modes $M_x = M_y = 16, M_z = 12$
$$\hat{v}_t^M \in \mathbb{C}^{16 \times 16 \times 12 \times 64}$$

Learned spectral weights
$$R_t \in \mathbb{C}^{16 \times 16 \times 12 \times 64 \times 64}$$

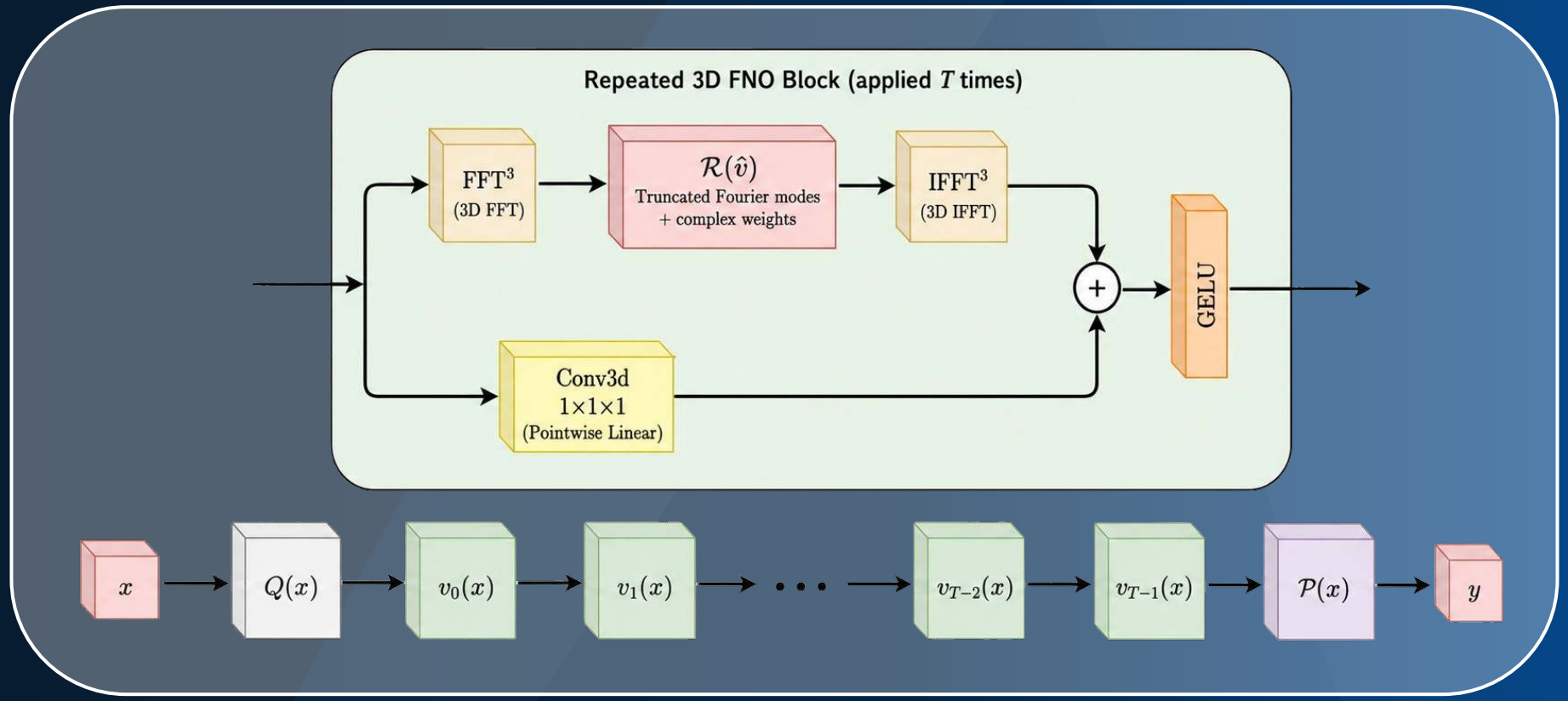
Inverse Fourier Transform
$$\mathcal{F}^{-1}(R_t \cdot \hat{v}_t^M) \in \mathbb{R}^{200 \times 200 \times 50 \times 64}$$

Down projection with $d = 3$:

$$\mathcal{P}: \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 64} \rightarrow \mathbb{R}^{200 \times 200 \text{ x } 50 \text{ x } 3}$$

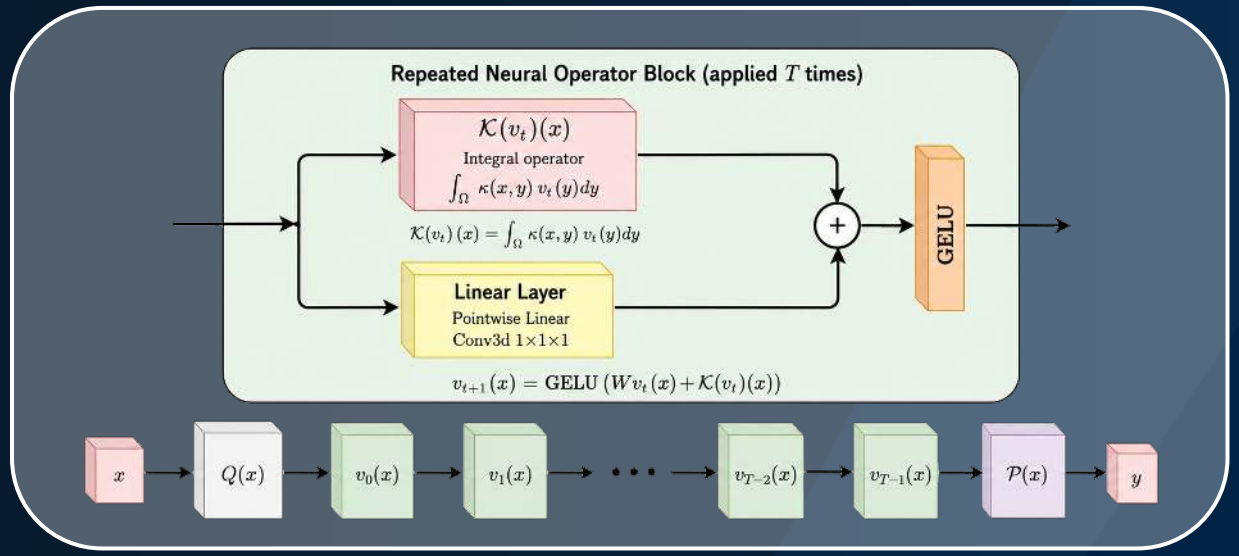
Fourier Neural Operator

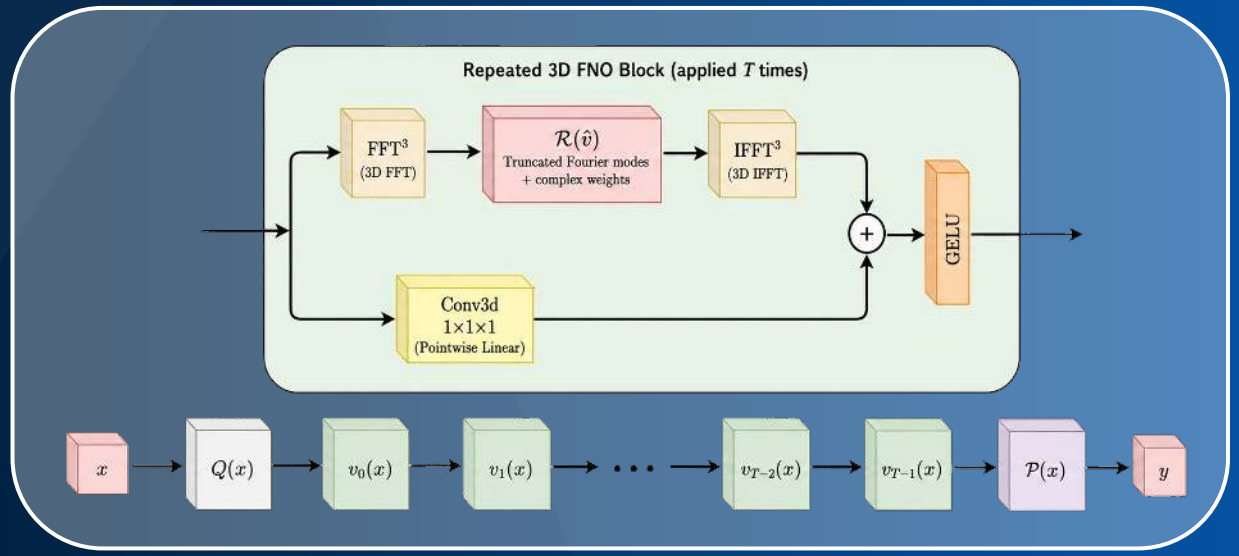
Architecture: Lifting, operator layers, and projection



Neural Operator vs Fourier Neural Operator

Architecture: Lifting, operator layers, and projection





From FNO to Factorized FNO

The main difference between neural operator variants is how they compute the nonlocal transformation inside each block.

Factorized Fourier Neural Operator

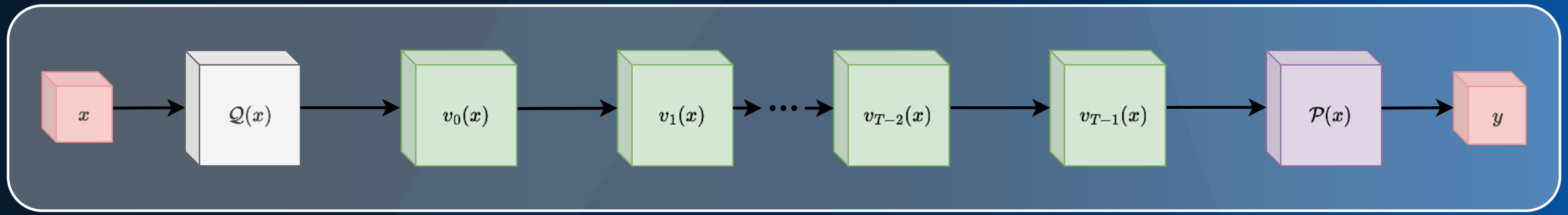
FFNO reduces the cost of full 3D spectral mixing by factorizing the Fourier operation along spatial dimensions.

Standard 3D FNO uses one joint 3D spectral transform:

$$\mathcal{F}_{\mathrm{x,y,z}}$$

FFNO uses separate directional transforms:

$$\mathcal{F}_{\mathrm{x,}}, \mathcal{F}_{\mathrm{y,}} \mathcal{F}_{\mathrm{z,}}$$



Factorized Fourier Neural Operator

FFNO reduces the cost of full 3D spectral mixing by factorizing the Fourier operation along spatial dimensions.

Standard 3D FNO uses one joint 3D spectral transform:

$$\mathcal{F}_{x,y,z}$$

Instead of learning one large spectral tensor:

$$R \in \mathbb{C}^{M_x \times M_y \times M_z \times d \times d}$$

FFNO uses separate directional transforms:

$$\mathcal{F}_{x,}, \mathcal{F}_{y,} \mathcal{F}_{z,}$$

It learns smaller directional operators:

$$R_x \in \mathbb{C}^{M_x \times d \times d}$$
$$R_y \in \mathbb{C}^{M_y \times d \times d}$$
$$R_z \in \mathbb{C}^{M_z \times d \times d}$$

Factorized Fourier Neural Operator

Each directional branch transforms the latent field along one axis while preserving the full grid shape after reconstruction.

Start with a latent space

$$v_t \in \mathbb{R}^{N_x \times N_y \times N_z \times d}$$

Directional Fourier Branches

Directional spectral transforms

Along x:

$$\mathcal{F}_x(v_t) \in \mathbb{C}^{N_x \times N_y \times N_z \times d}$$

Retain M_x modes along x:

$$\mathcal{F}_x(v_t) = \hat{v}_{t,x}^{M} \in \mathbb{C}^{M_x \times N_y \times N_z \times d}$$

$$R_x \cdot \hat{v}_{t,x}^{M} \quad \text{with } R_x \in \mathbb{C}^{M_x \times d \times d}$$

Along y:

$$\mathcal{F}_y(v_t) \in \mathbb{C}^{N_x \times N_y \times N_z \times d}$$

Retain M_y modes along x:

$$\mathcal{F}_y(v_t) = \hat{v}_{t,y}^{M} \in \mathbb{C}^{N_x \times M_y \times N_z \times d}$$

$$R_y \cdot \hat{v}_{t,y}^{M} \quad \text{with } R_x \in \mathbb{C}^{M_y \times d \times d}$$

Along z:

$$\mathcal{F}_z(v_t) \in \mathbb{C}^{N_x \times N_y \times N_z \times d}$$

Retain M_z modes along x:

$$\mathcal{F}_z(v_t) = \hat{v}_{t,z}^{M} \in \mathbb{C}^{N_x \times N_y \times M_z \times d}$$

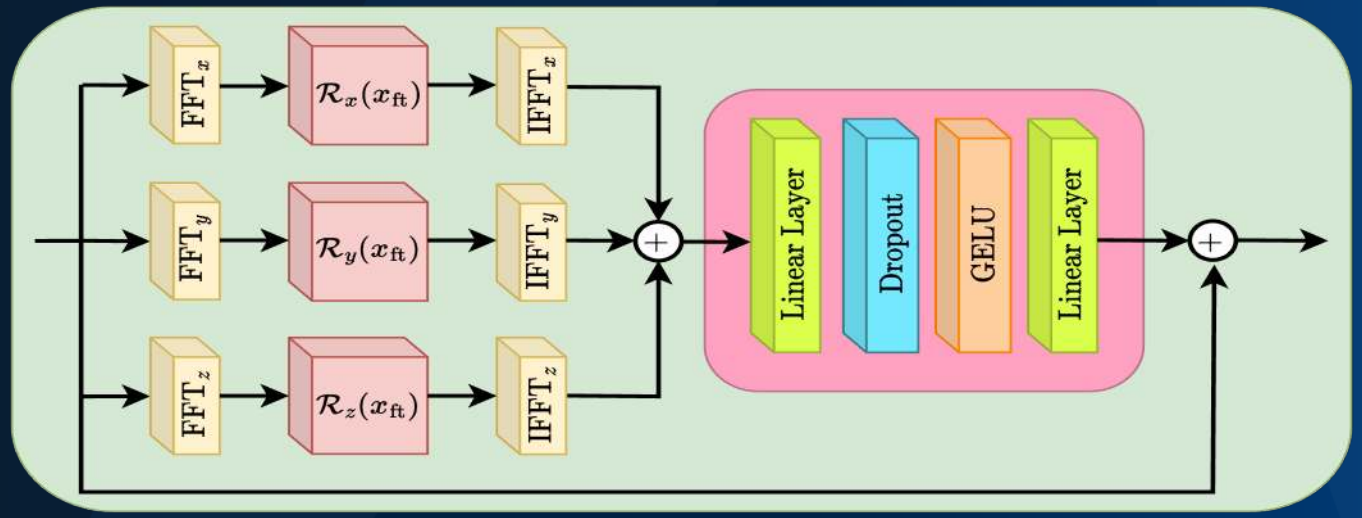
$$R_z \cdot \hat{v}_{t,z}^{M} \quad \text{with } R_x \in \mathbb{C}^{M_z \times d \times d}$$

Factorized Fourier Neural Operator

FFNO reduces the cost of full 3D spectral mixing by factorizing the Fourier operation along spatial dimensions.

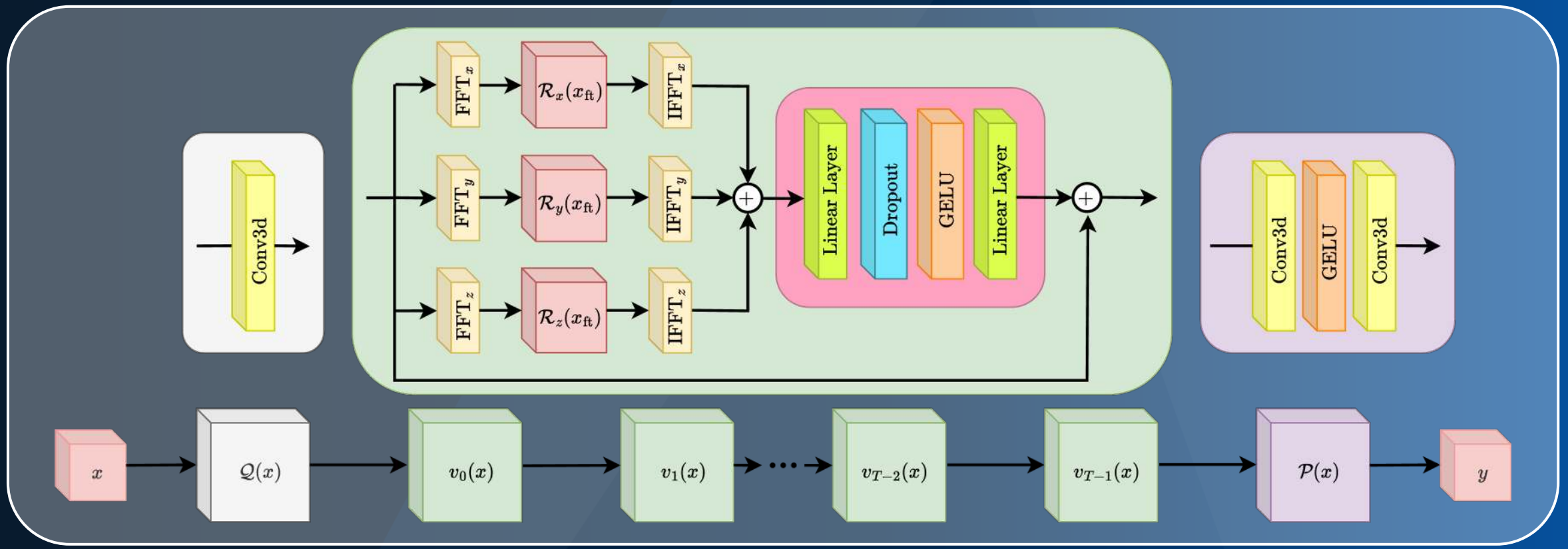
$$\mathrm{K_{FFNO}}\,(v_t) = \mathcal{F}_{\mathrm{x}}^{-1}(\mathrm{R_x}\mathcal{F}_{\mathrm{x}}\,(v_{\mathrm{t}})) + \mathcal{F}_{\mathrm{y}}^{-1}(\mathrm{R_y}\mathcal{F}_{\mathrm{y}}\,(v_{\mathrm{t}})) + \mathcal{F}_{\mathrm{z}}^{-1}(\mathrm{R_z}\mathcal{F}_{\mathrm{z}}\,(v_{\mathrm{t}}))$$

$$v_{t+1}(x) = v_{\mathrm{t}} + \mathrm{W_{t,2}}\sigma(\mathrm{W_{t,1}}\mathrm{K_{FFNO}}\,(v_t))$$



Factorized Fourier Neural Operator

FFNO reduces the cost of full 3D spectral mixing by factorizing the Fourier operation along spatial dimensions.



Why Factorize the Fourier Operator

Factorization makes spectral neural operators more practical for high-dimensional 3D simulation data.

Parameter-cost comparison

Full 3D FNO spectral weights:
$\mathcal{O}(\mathrm{M_x M_y M_z d^2})$

3D Factorized FNO spectral weights:
$\mathcal{O}((\mathrm{M_x} + \mathrm{M_y} + \mathrm{M_z})\mathrm{d}^2)$

The spectral parameter structure is much smaller for FFNO.

Example with:
$\mathrm{M_x} = \mathrm{M_y} = 16, \mathrm{M_z} = 12, \mathrm{d} = 64$

Full FNO:
$16 \cdot 16 \cdot 12 \cdot 64^2 = \mathbf{12,582,912}$

FFNO:
$(16 + 16 + 12) \cdot 64^2 = \mathbf{4,140}$

How Neural Operator Architectures Differ

Comparison of the core idea, operator implementation, and latent-field transformation inside each block.

Architecture	Core Idea	Operator Implementation	Shape through Blocks
Neural Operator	Learn a map between functions	Nonlocal integral operator $\mathcal{K}$	$\Omega \rightarrow \mathbb{R}^{d} \rightarrow \Omega \rightarrow \mathbb{R}^{d}$
Fourier Neural Operator	Implement the nonlocal operator in Fourier space	Complex weights on selected Fourier modes	$v_{t} \in \mathbb{R}^{N_x \times N_y \times N_z \times d}$ $\hat{v}_{t}^{M} = \mathcal{F}(v_{t}) \in \mathbb{C}^{M_x \times M_y \times M_z \times d}$ $v_{t+1} \in \mathbb{R}^{N_x \times N_y \times N_z \times d}$
Factorized Fourier Neural Operator	Reduce spectral cost by factorizing across dimensions	Separate Fourier transforms	Same grid shape; separate directional spectral branches along x, y, and z

When to Use Which Neural Operator?

Each architecture offers a different trade-off between generality, efficiency, and scalability.

Architecture	Main strength	Main limitation	Best use case
Neural Operator	General framework for field-to-field learning	Abstract concept; requires a concrete implementation of $\mathcal{K}$	Explaining PDE solution maps and operator learning principles
Fourier Neural Operator	Global spatial mixing and efficient learning of smooth PDE fields	Full 3D spectral weights can be memory- and parameter-intensive	General PDE surrogate modeling on structured grids
Factorized Fourier Neural Operator	Lower memory and parameter cost; better scalability to large 3D fields	Less direct joint 3D spectral coupling than full FNO	Large 3D CFD domains and high-dimensional spatiotemporal data

Dataset

A collection of examples that form the model's knowledge base.

Model

A neural network with a specific architecture adapted on the data or use case.

Training

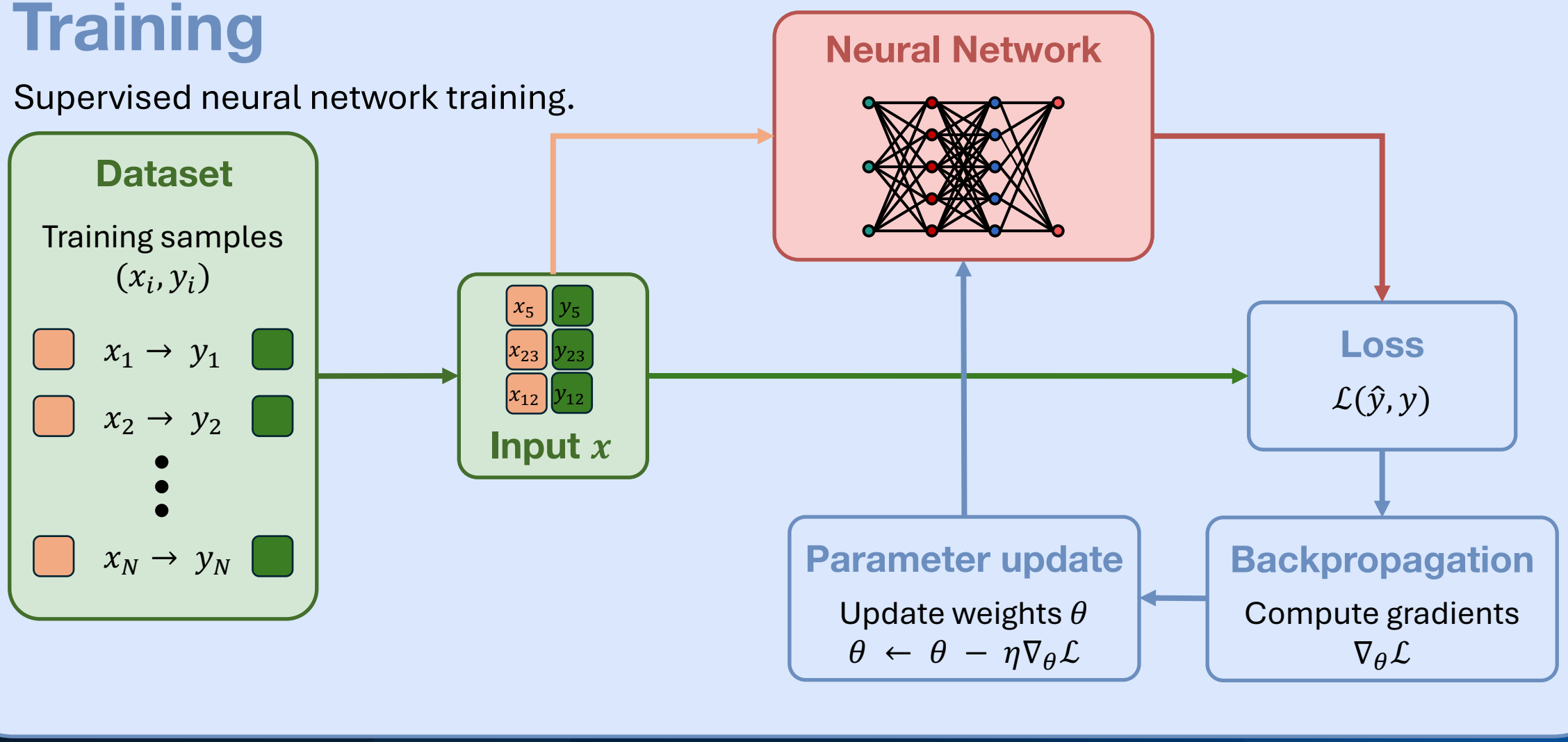
The process where a neural network learns patterns from data.

Inference

After training, the model uses what it has learned to make predictions or generate content from unseen inputs.

Training

Supervised neural network training.



Training

A neural operator first lifts the input field to a higher-dimensional feature space, applies several operator layers, and then projects the hidden representation back to the physical output field.

$$\theta^* = \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i)$$

Mean Squared Error (MSE)

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Root Mean Squared Error (RMSE)

$$\mathcal{L}_{\text{RMSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

Relative L^2 Loss

$$\mathcal{L}_{\text{rel}L^2} = \frac{\| \hat{u} - u \|_2}{\| u \|_2} = \frac{\sqrt{\sum_{i=1}^{N}(\hat{u} - u)^2}}{\sqrt{\sum_{i=1}^{N} u^2}}$$

Training

Backpropagation: Computing gradients of the loss with respect to network parameters

Backpropagation is the algorithm used to compute how each trainable parameter of a neural network contributes to the final loss.

$$\mathcal{L}(\hat{y}, y) = \mathcal{L}(f_\theta(x), y)$$

It applies the chain rule from the output layer back to the earlier layers and produces the gradients needed for parameter updates.

Backpropagation computes the gradient of the loss with respect to the trainable parameters:

$$\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta}$$

Training

Optimization: Updating the parameters using the gradients computed by backpropagation

Backpropagation computes how the loss changes with respect to each parameter:

$$\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta}$$

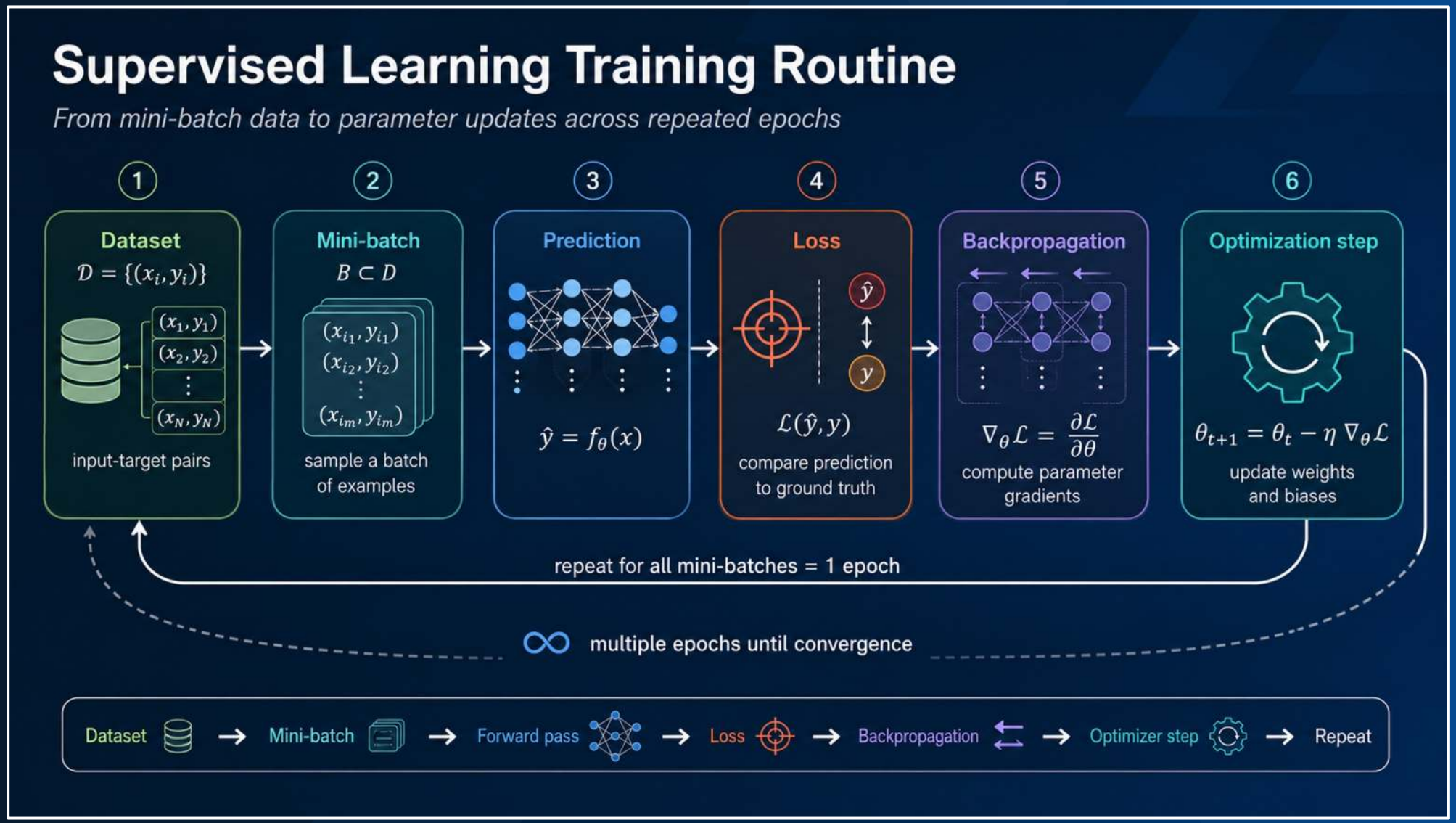
The optimizer uses these gradients to update the parameters in a direction that reduces the loss:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}$$

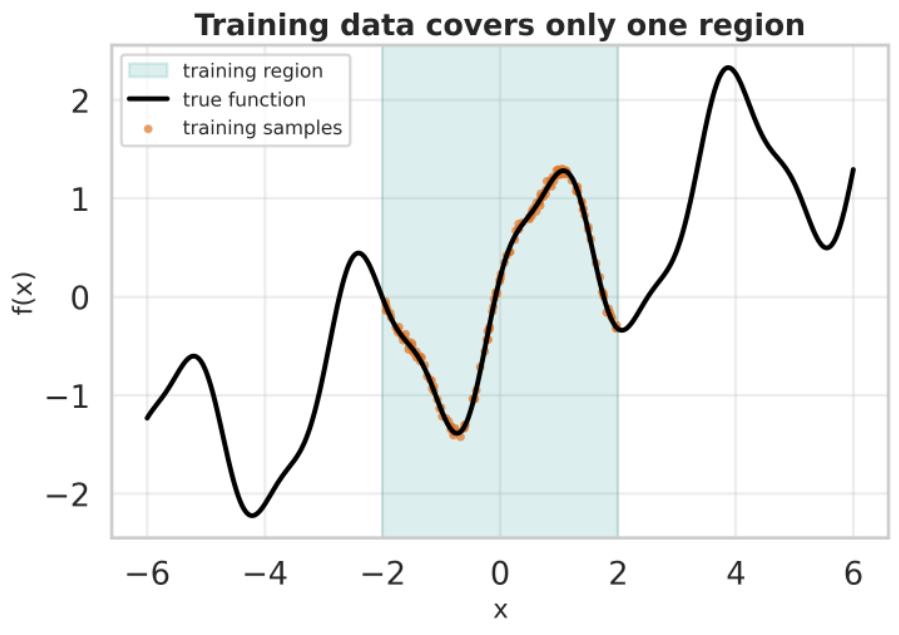
The learning rate η controls the size of each update step. For a weight matrix and bias vector:

$$W_{t+1}^{(l)} = W_t^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(l)}}$$

$$b_{t+1}^{(l)} = b_t^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial b^{(l)}}$$


Supervised Learning Training Routine
From mini-batch data to parameter updates across repeated epochs
1
Dataset
$\mathcal{D} = \{(x_i, y_i)\}$
(x_1, y_1)
(x_2, y_2)
(x_N, y_N)
input-target pairs
2
Mini-batch
$B \subset D$
(x_{i_1}, y_{i_1})
(x_{i_2}, y_{i_2})
(x_{i_m}, y_{i_m})
sample a batch of examples
3
Prediction
$\hat{y} = f_\theta(x)$
4
Loss
$\hat{y}$
y
$\mathcal{L}(\hat{y}, y)$
compare prediction to ground truth
5
Backpropagation
$\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta}$
compute parameter gradients
6
Optimization step
$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}$
update weights and biases
repeat for all mini-batches = 1 epoch
multiple epochs until convergence
Dataset
Mini-batch
Forward pass
Loss
Backpropagation
Optimizer step
Repeat

Dataset



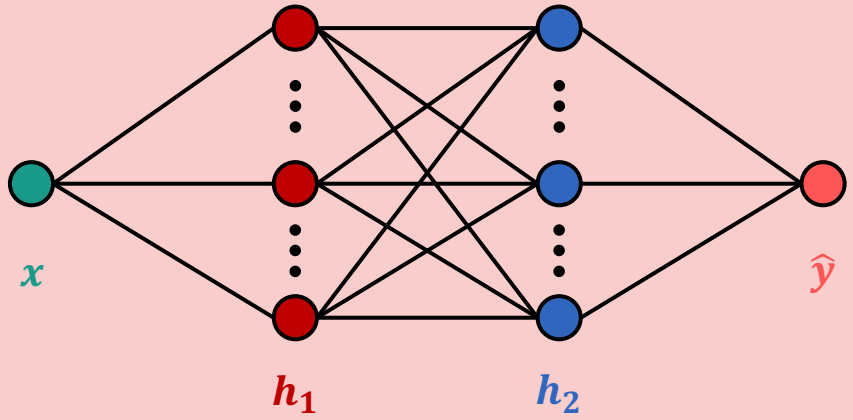
$$f(x) = \sin(2x) + 0.3x + 0.2\cos(5x)$$

Evaluation domain: $x \in [-6, 6]$

Training domain: $x \in [-2, 2]$

$$y_i = f(x_i) + \varepsilon_i, \quad \varepsilon_i \in \mathcal{N}(0, 0.003^2)$$

Neural Network



$$f_\theta = A_3\left(\tanh\left(A_2\left(\tanh\left(A_1(x)\right)\right)\right)\right)$$

$$A_1(x) = W_1(x) + b_1 \qquad W_1 \in \mathbb{R}^{32\times 1}, b_1 \in \mathbb{R}^{32}$$

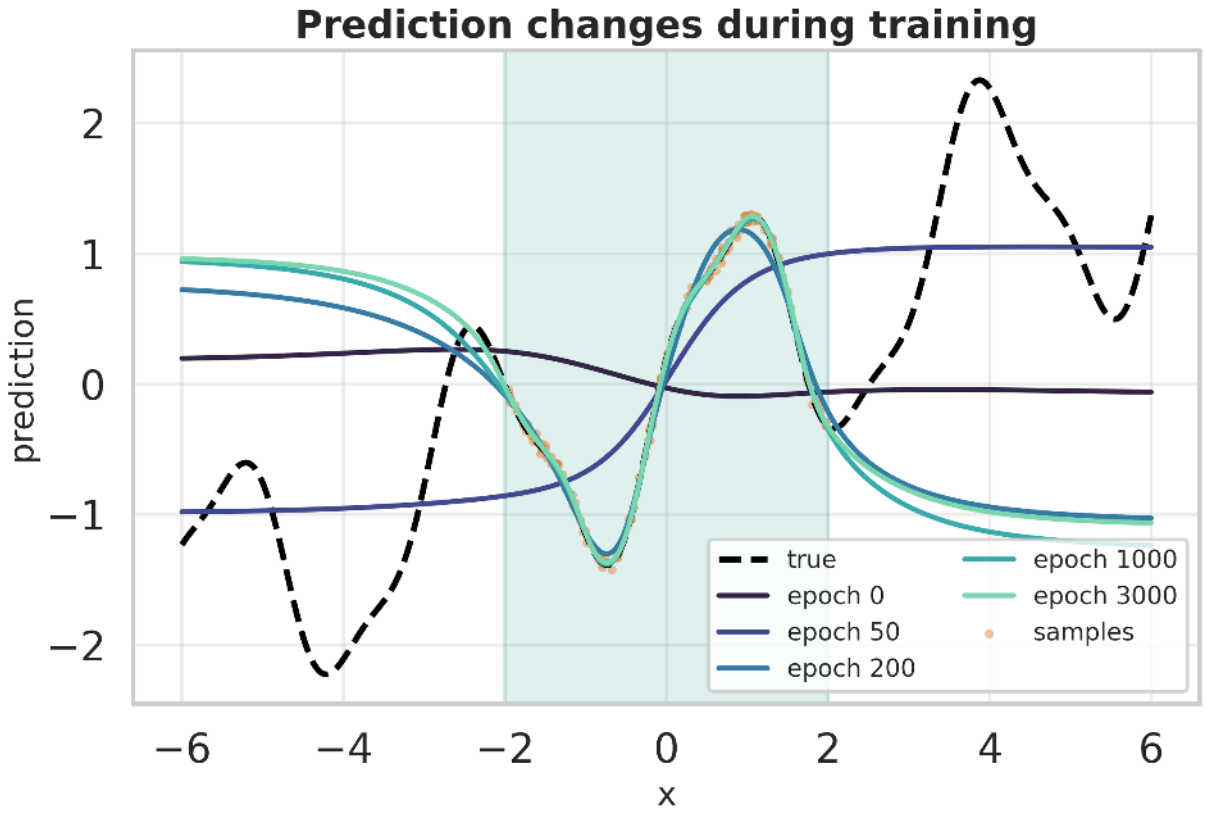
$$A_2(h_1) = W_2(h_1) + b_2 \qquad W_2 \in \mathbb{R}^{32\times 32}, b_2 \in \mathbb{R}^{32}$$

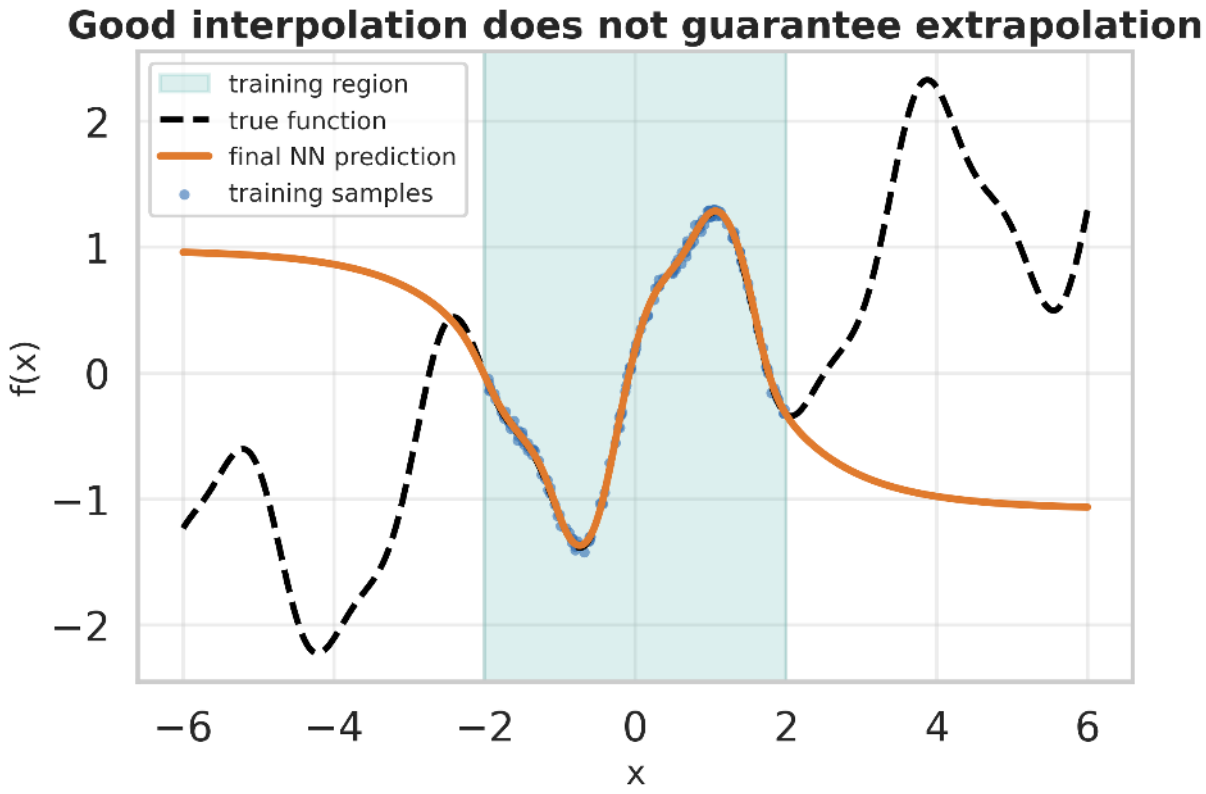
$$A_3(h_2) = W_3(h_2) + b_3 \qquad W_3 \in \mathbb{R}^{1\times 32}, b_3 \in \mathbb{R}$$

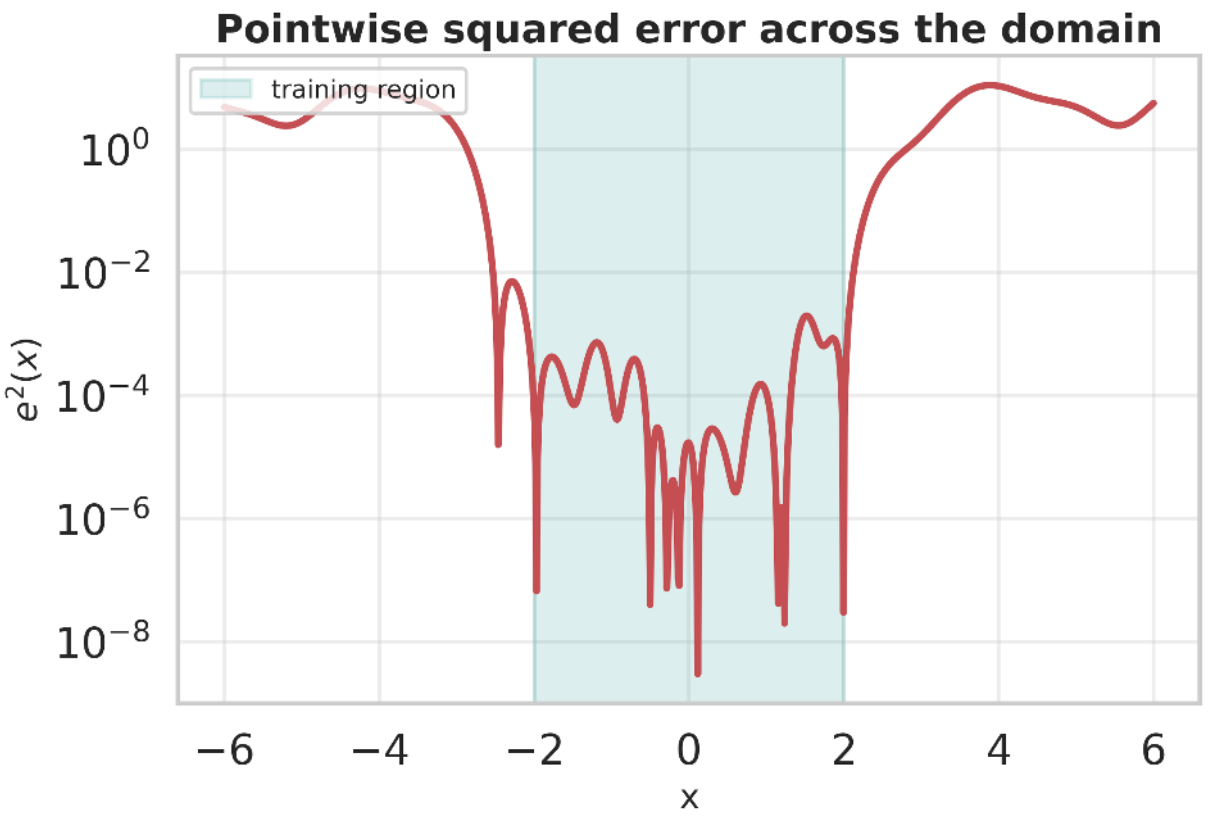
$$h_1 = \tanh(W_1 x + b_1)$$

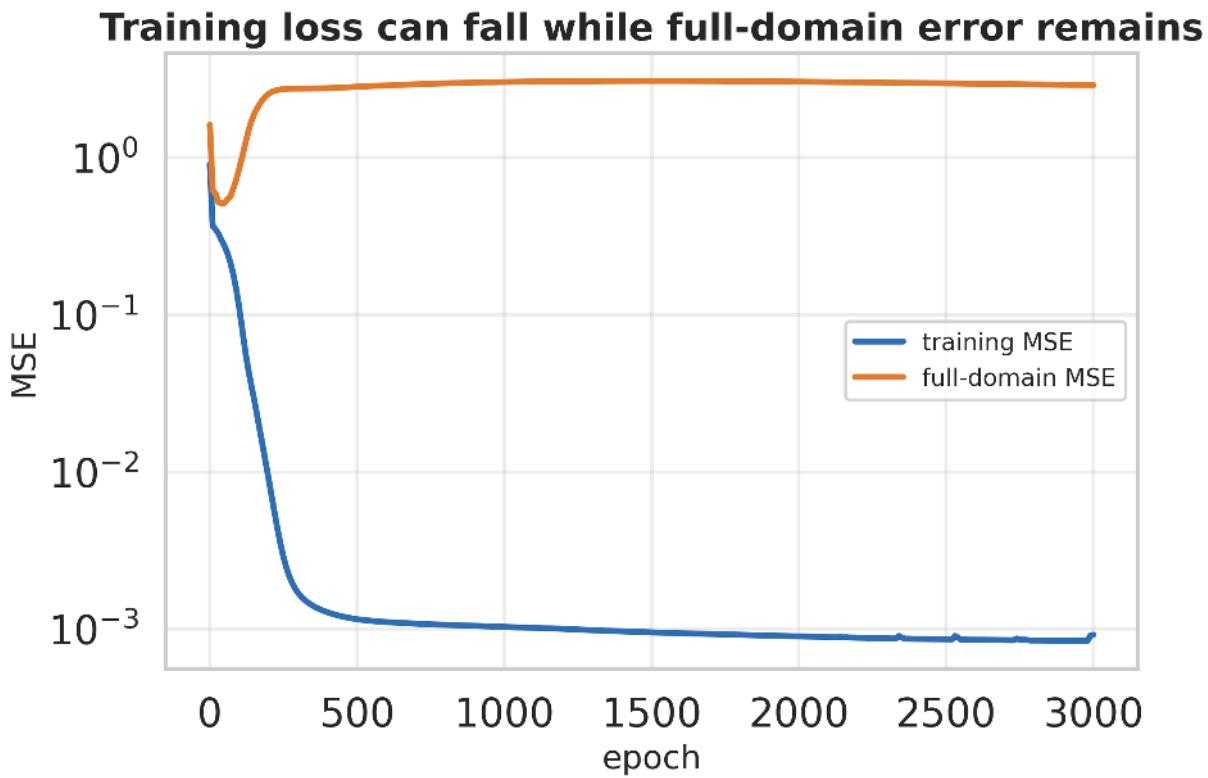
$$h_2 = \tanh(W_2 x + b_2)$$

$$\hat{y} = W_3 h_2 + b_3)$$


Prediction changes during training
prediction
x
true
epoch 0
epoch 50
epoch 200
epoch 1000
epoch 3000
samples


Good interpolation does not guarantee extrapolation
f(x)
x
training region
true function
final NN prediction
training samples


Pointwise squared error across the domain
training region
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
$e^2(x)$
−6
−4
−2
0
2
4
6
x


Training loss can fall while full-domain error remains
10^{0}
10^{-1}
10^{-2}
10^{-3}
MSE
training MSE
full-domain MSE
0
500
1000
1500
2000
2500
3000
epoch

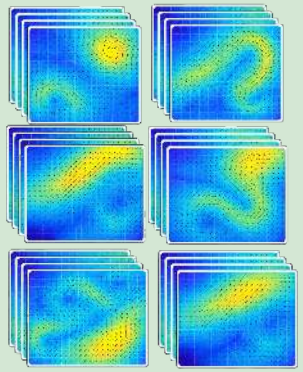
Training Neural Operators

The model is trained to map current flow fields to future states, while teacher forcing stabilizes learning over multiple prediction steps.

Teacher Forcing

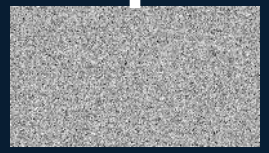
Prediction

Dataset



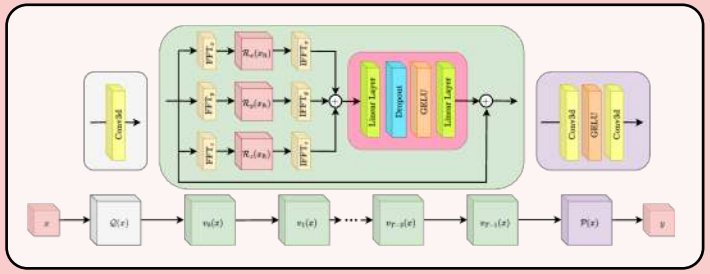
u_i

+



$[u_{i+1}]$

Factorized Fourier Neural Operator



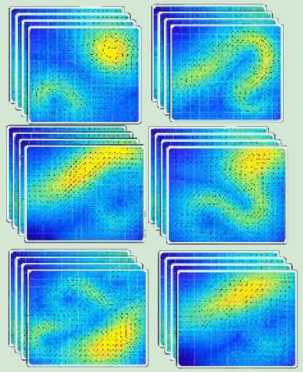
$\hat{u}_{i+1}$

$[\hat{u}_{i+1}]$

Teacher Forcing

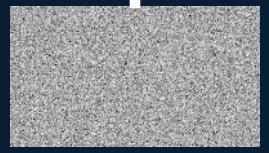
Prediction

Dataset



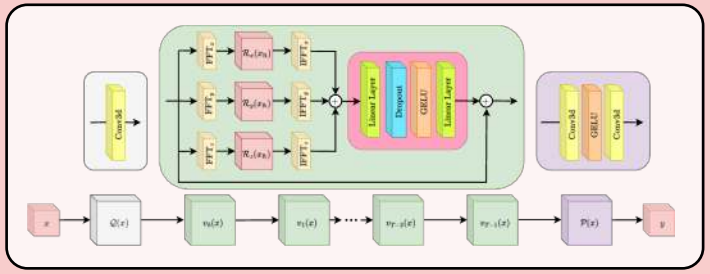
u_{i+1}

$+$



$\begin{bmatrix} u_{i+1} \\ u_{i+2} \end{bmatrix}$

Factorized Fourier Neural Operator



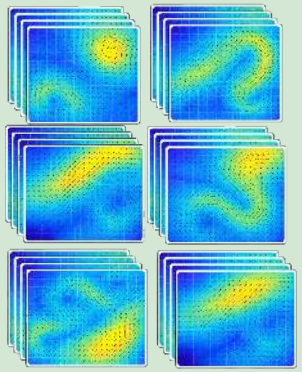
$\hat{u}_{i+2}$

$\begin{bmatrix} \hat{u}_{i+1} \\ \hat{u}_{i+2} \end{bmatrix}$

Teacher Forcing

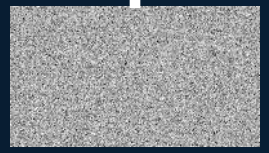
Prediction

Dataset



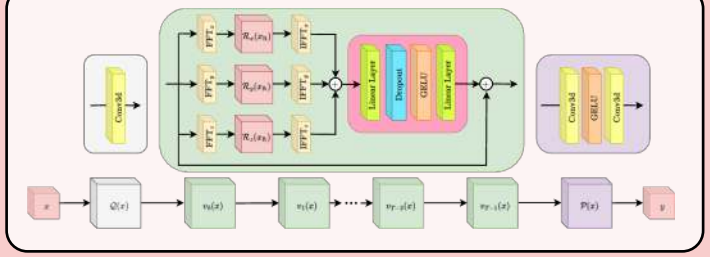
u_{i+2}

$+$



$$\begin{bmatrix} u_{i+1} \\ u_{i+2} \\ u_{i+3} \end{bmatrix}$$

Factorized Fourier Neural Operator



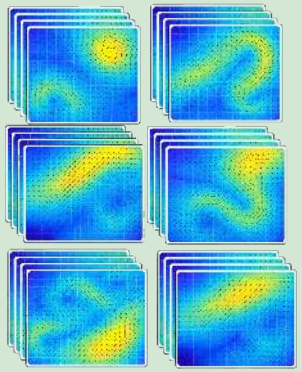
$\hat{u}_{i+3}$

$$\begin{bmatrix} \hat{u}_{i+1} \\ \hat{u}_{i+2} \\ \hat{u}_{i+3} \end{bmatrix}$$

Teacher Forcing

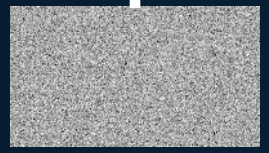
Prediction

Dataset

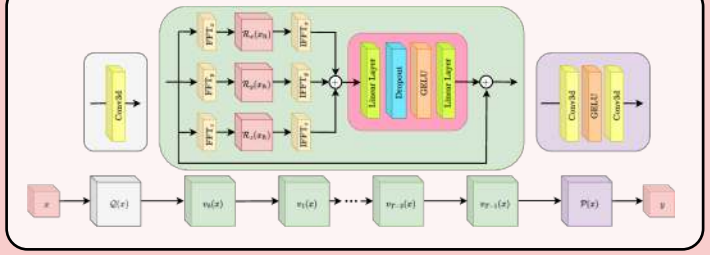


u_{i+3}

$+$



Factorized Fourier Neural Operator



$\hat{u}_{i+4}$

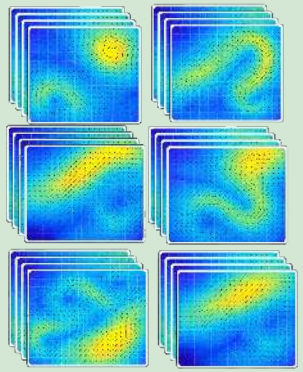
$$\begin{bmatrix} u_{i+1} \\ u_{i+2} \\ u_{i+3} \\ u_{i+4} \end{bmatrix}$$

$$\begin{bmatrix} \hat{u}_{i+1} \\ \hat{u}_{i+2} \\ \hat{u}_{i+3} \\ \hat{u}_{i+4} \end{bmatrix}$$

Teacher Forcing

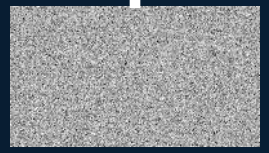
Prediction

Dataset

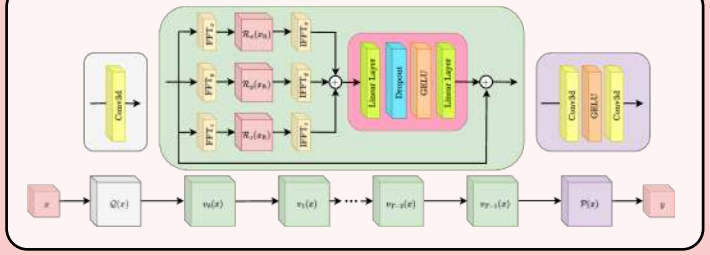


u_{i+4}

$+$



Factorized Fourier Neural Operator



$\hat{u}_{i+5}$

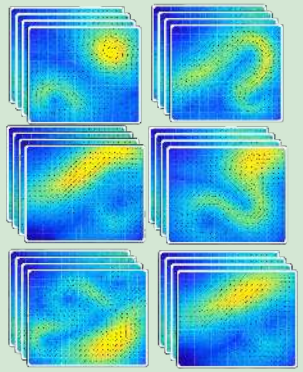
$$\begin{bmatrix} u_{i+1} \\ u_{i+2} \\ u_{i+3} \\ u_{i+4} \\ u_{i+5} \end{bmatrix}$$

$$\begin{bmatrix} \hat{u}_{i+1} \\ \hat{u}_{i+2} \\ \hat{u}_{i+3} \\ \hat{u}_{i+4} \\ \hat{u}_{i+5} \end{bmatrix}$$

Teacher Forcing

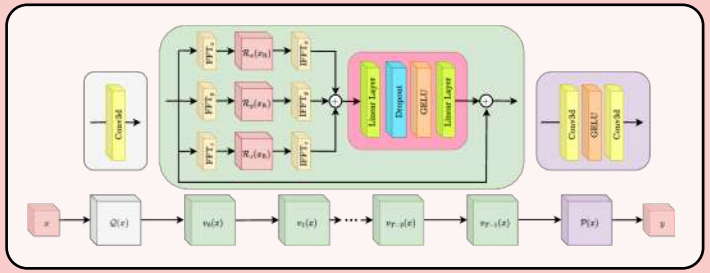
Prediction

Dataset



u_{i+4}

Factorized Fourier Neural Operator



$\hat{u}_{i+5}$

$$\begin{bmatrix} \hat{u}_{i+1} \\ \hat{u}_{i+2} \\ \hat{u}_{i+3} \\ \hat{u}_{i+4} \\ \hat{u}_{i+5} \end{bmatrix}$$

Backpropagation

Optimization

$$\begin{bmatrix} u_{i+1} \\ u_{i+2} \\ u_{i+3} \\ u_{i+4} \\ u_{i+5} \end{bmatrix}$$

Loss RMSE

$$\mathcal{L}_{\text{RMSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

Dataset

A collection of examples that form the model's knowledge base.

Model

A neural network with a specific architecture adapted on the data or use case.

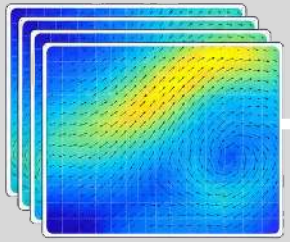
Training

The process where a neural network learns patterns from data.

Inference

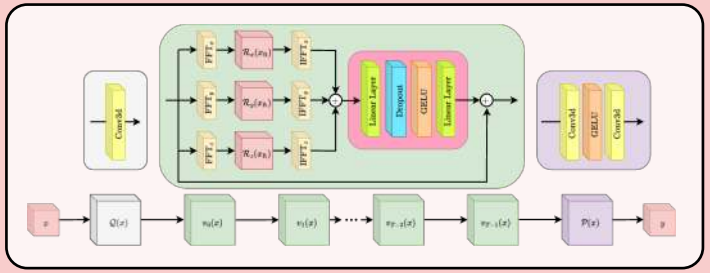
After training, the model uses what it has learned to make predictions or generate content from unseen inputs.

Inference

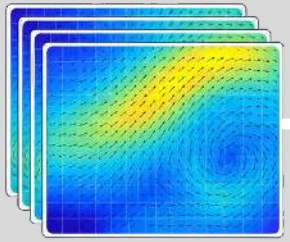


New sample u_i

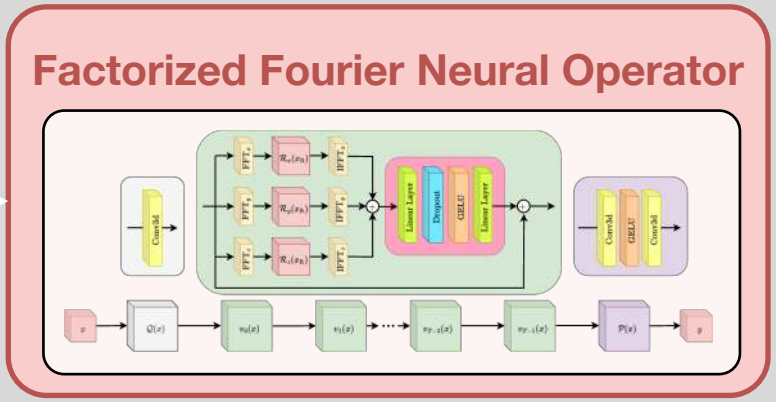
Factorized Fourier Neural Operator



Inference

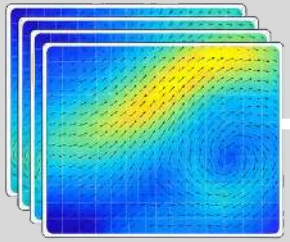


New sample u_i

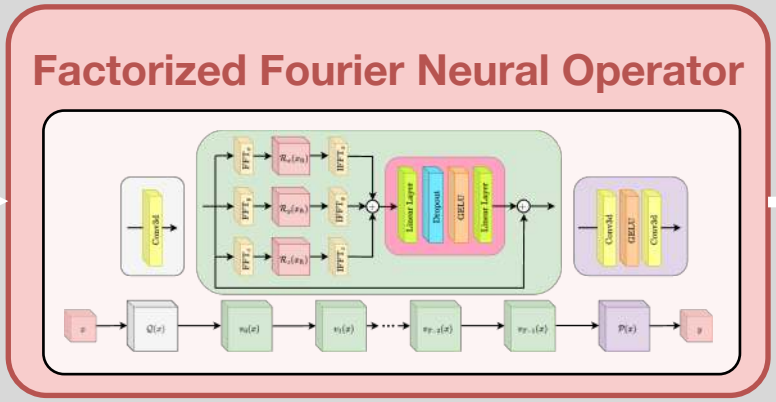


Forward Pass

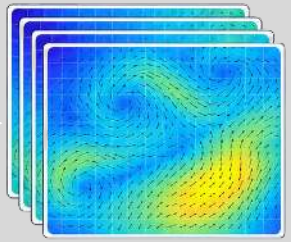
Inference



New sample u_i



Forward Pass

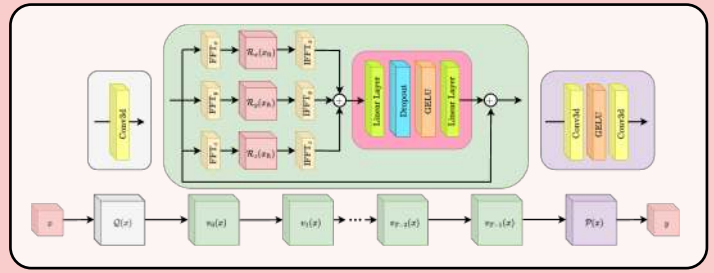


Prediction $\hat{u}_{i+1}$

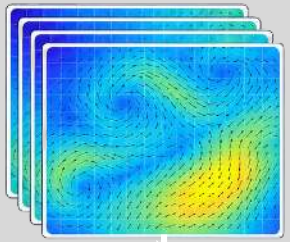
Inference

Input $\hat{u}_{i+1}$

Factorized Fourier Neural Operator

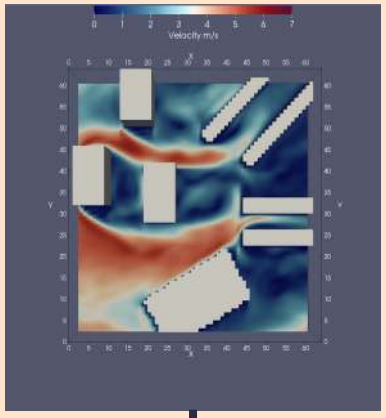


Prediction $\hat{u}_{i+1}$

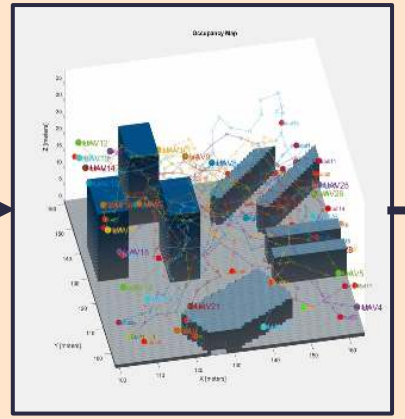


Autoregressive Model

Static 3D Wind Field

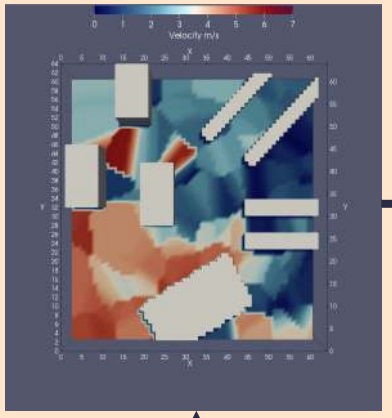


Scattered Wind Measurements



Extra- Interpolation on Grid

Discretized 3D Wind Field

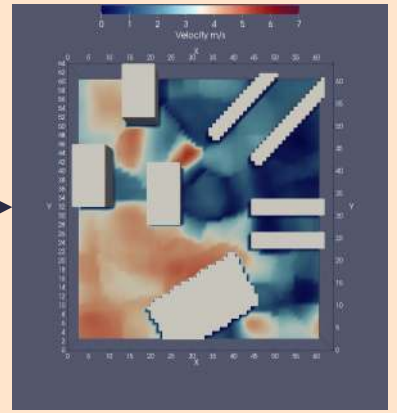


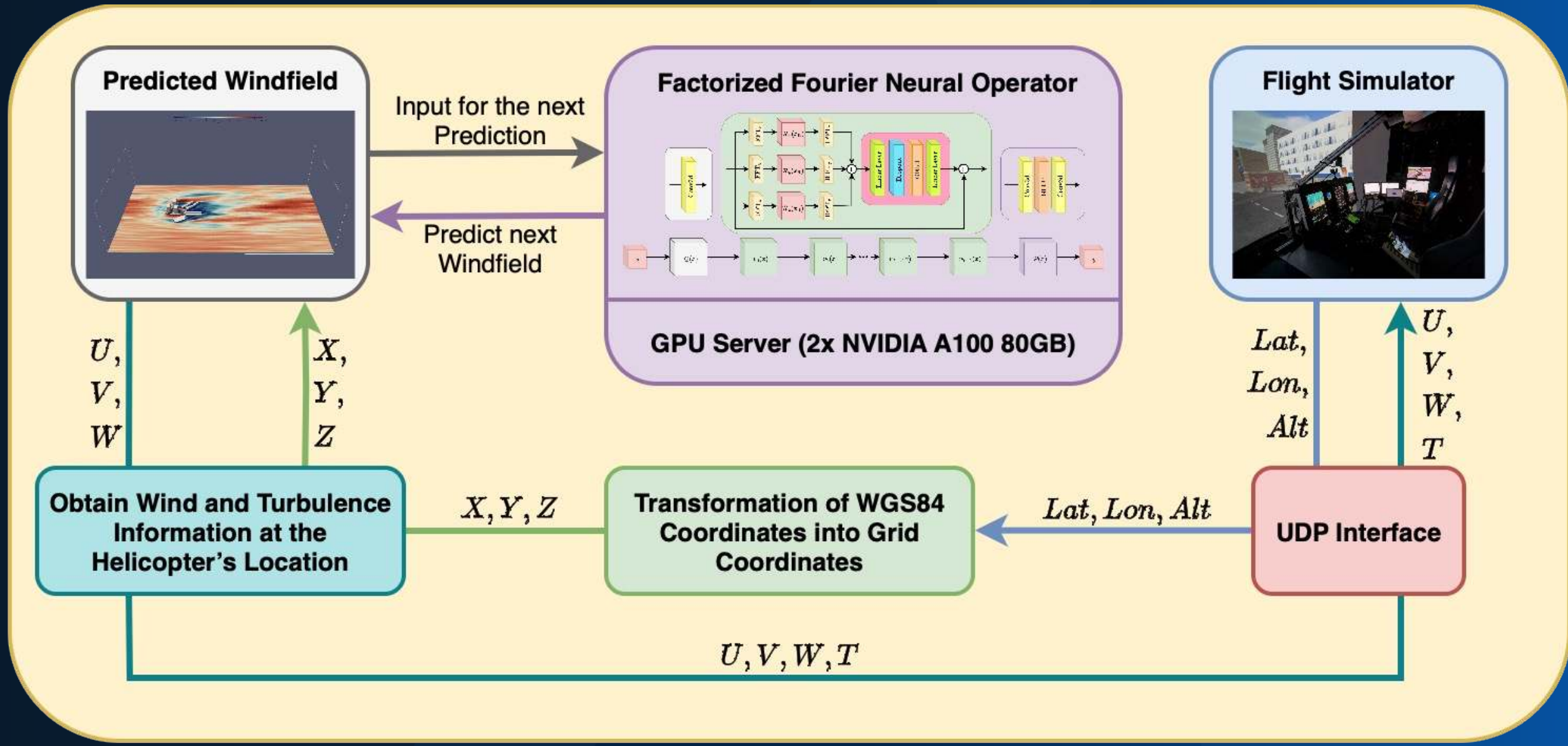
Input

Neural Network

Output

Predicted 3D Wind Field




Predicted Windfield
Input for the next Prediction
Predict next Windfield
Factorized Fourier Neural Operator
GPU Server (2x NVIDIA A100 80GB)
Flight Simulator
U, V, W
X, Y, Z
Lat, Lon, Alt
U, V, W, T
Obtain Wind and Turbulence Information at the Helicopter's Location
X, Y, Z
Transformation of WGS84 Coordinates into Grid Coordinates
Lat, Lon, Alt
UDP Interface
U, V, W, T

The Cost of Predicting Step by Step

Thank you for your attention!

Maximilian Dauner
maximilian.dauner0@hm.edu
Munich University of Applied Sciences
Munich Center for Digital Sciences and AI (MUC.DAI)

HM

MUC.DAI
Munich Center for
Digital Sciences and AI

CIMPA

Lab 2

From Neural Networks to PDE Prediction

In this lab, you will build an intuitive understanding of neural network training, starting from affine transformations, activation functions, and multilayer perceptrons before moving toward field-based learning for PDE data.

By training a small neural network and a 1D Fourier Neural Operator on PDEBench data, you will learn how models are optimized, how losses are evaluated, and how one-step predictions can be extended to autoregressive inference for physical systems.

https://syncandshare.lrz.de/getlink/fi3EzfcA5PCj7EBgFNkMQD/public

MUC.DAI
Munich Center for
Digital Sciences and AI

CIMPA