AI4Science & Physics-Aware Deep Learning for Modeling Dynamical Systems

Sorbonne Université – Masters DAC et MS2A. PatrickGallinari,
patrick.gallinari@sorbonne-universite.fr, https://pages.isir.upmc.fr/gallinari

Year 2025-2026

Outline

- Context: AI4science
- Background: Neural networks and ordinary differential equations
  - NNs as numerical schemes for solving ODEs
- Modeling Spatio-temporal dynamics with Neural Networks
  - NNs as surrogate models for solving PDEs - Data-driven approaches
    - Discrete space models
    - Continuous space models
  - NNs as surrogate models for solving PDEs – Data free approaches
- Hybrid models
  - Incorporating physical knowledge in dynamics models
- Generalization in ML models for dynamics modeling

Context: AI4Science

AI4Science as a new scientific paradigm

- Paradigm shift: from explicit formulation to implicit knowledge discovery
- Emerged in 2018 – rapidly growing field
- Involves many scientific communities
- We are still at the begining of the process

AI4Science paradigm changes

How research is done: From hypothesis generation to data analysis, experimentation, and discovery.

The questions we can ask: Enabling exploration of complexity and scale previously impossible.

The pace of discovery: Accelerating insights in fields like drug discovery, material science, climate modeling, and fundamental physics.

AI for Science as a new scientific paradigm
Worldwide initiatives - examples

Academic research

- Polymathic AI - Foundation Models for Science
  - Simons Foundation NY, Schmidt Future, NYU, Princeton, Berkeley, Cambridge
- Stanford: Center for Research on Foundation Models
  - Interdisciplinary, spans10+ dpts
- Univ. Michigan: Center for Scientific Foundation Models

Industrial labs

- AI Alliance
- Microsoft research: AI for Science
  - How-new-ai-foundation-models-can-speed-up-scientific-discovery
  - C. Bishop Keynote (2024) : The Revolution in Scientific Discovery
- Google Applied Science
  - Combining computer science with physics and biology to create breakthroughs that help the world

AI for Science as a new scientific paradigm

Transformative role of AI

- AI as digital twins/ surrogates
  - Complexity: accelerated simulations: CFD, molecular dynamics
  - Unknown physics
- AI as scientific partner: complementing physical models
  - Hybrid models: numerical solvers + ML
  - Accelerate discovery: biological workflows
- AI as a reasoning engine
  - Theorem proving
  - Code generation

Evolution of AI models

- Surrogate models
  - Solving specific problems: CFD, materials property prediction
- Foundation models
  - Multi-physics, multi-domains: biology, weather forecast
- AI agents
  - Solve high level objectives
  - Requires reasoning, planing, etc
  - Orchestrates various AI tools

Transformative role of AI - Examples

AI as digital twins
Weather forecasting and climate

2022-2024 – Foundation Models for weather prediction (ERA5 dataset 40 years hourly reanalysis data)

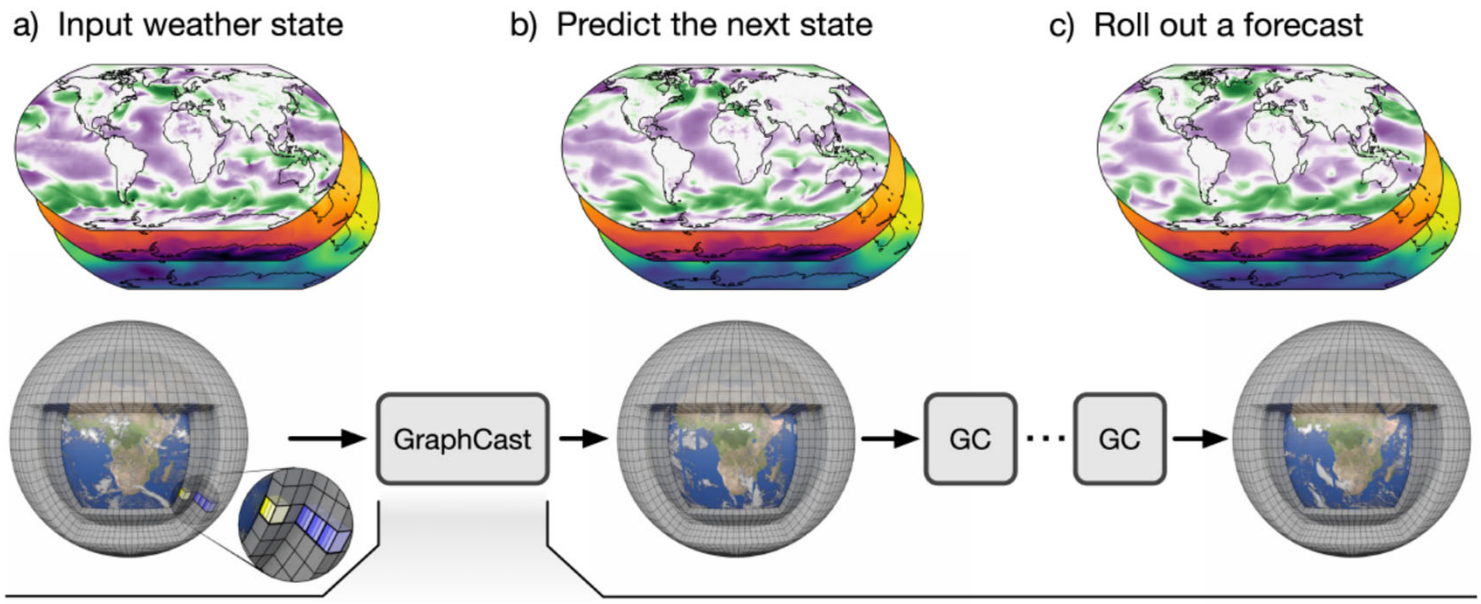
GraphCast – Google & DeepMind 2022
https://arxiv.org/abs/2212.12794
ClimaX – Msoft & UCLA 2023
https://arxiv.org/abs/2301.10343
Pangu-Weather – Huawei 2023
http://arxiv.org/abs/2211.02556
FourCastNet – NVIDIA&Lawrence Berkeley lab.&al.
http://arxiv.org/abs/2202.11214
Neural General Circulation Model – Google 2023
https://arxiv.org/abs/2311.07222
Aurora – Microsoft 2024
https://arxiv.org/abs/2405.13063
AIFS model
https://arxiv.org/abs/2406.01465

Foundation models for weather forecasting: GraphCast (Lam et al. 2023)

https://arxiv.org/abs/2212.12794

- Data-driven approach to weather forecast
- Learn from historical data
  - 39 years (1979-2017) of historical data from ECMWF ERA5 reanalysis archive – petabytes of data
    - ECMWF: European Center for Medium-Range Weather Forecast
  - Test 2018 onward
  - Time step: 6 hours
  - State variables
    - 5 surface variables (temperature, wind speed, etc)
    - 6 athmospheric variables (temp., wind, etc) at 37 pressure levels
    - 0.25° latitude/ longitude grid, 28x28 kilometer resolution, $1M$ points
- Objective
  - Given state variables at t and t-6 hours, predict next state (t + 6)
  - Prediction horizon: 10 days (medium range), auto-regressive model

Foundation models for weather forecasting: GraphCast (Lam et al. 2023) - https://arxiv.org/abs/2212.12794



(Fig. Lam et al 2023)

- Based on Graph Neural Networks
- Performance on weather prediction: on par or better than HRES the SOTA ECMWF model
- Downstream tasks (not trained on)
  - Tropical cyclone tracking
  - Extreme heat and cold, ….

AI as digital twins

Weather forecasting and climate

Aurora (Bodnar et al. 2024), https://arxiv.org/abs/2405.13063

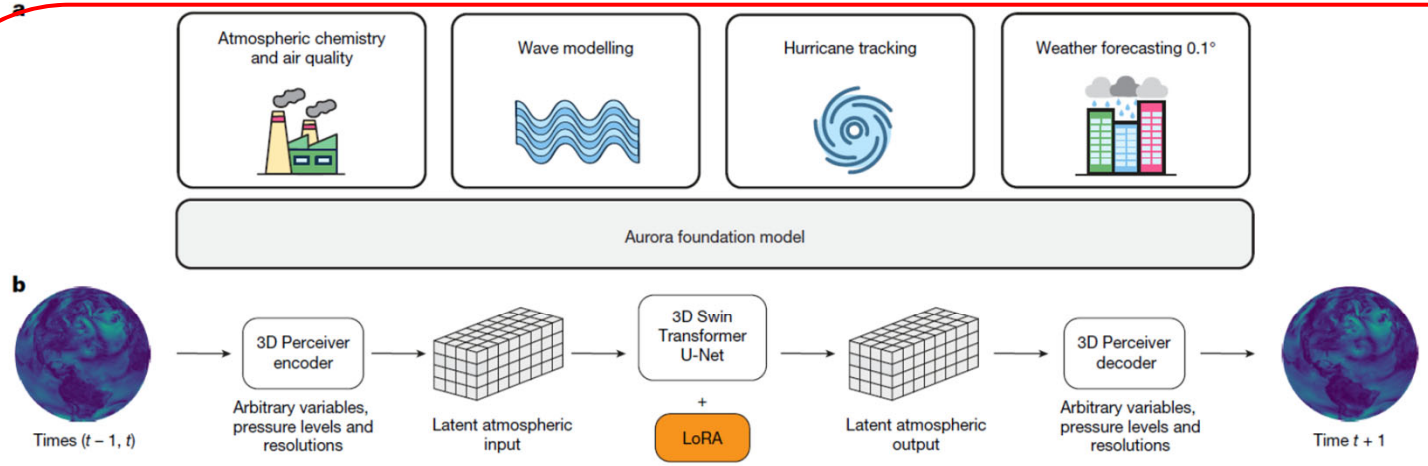


Fig. 1 | Aurora is a 1.3-billion-parameter foundation model for the Earth system. Icons are for illustrative purposes only. **a**, Aurora is pretrained on several heterogeneous datasets with different resolutions, variables and pressure levels. The model is then fine-tuned for several operational forecasting scenarios at different resolutions: atmospheric chemistry and air quality at 0.4°, wave modelling at 0.25°, hurricane tracking at 0.25° and weather forecasting at 0.1°. **b**, Aurora is a flexible 3D Swin Transformer[19] with 3D Perceiver-based[21] atmospheric encoders and decoders. The model is able to ingest inputs with different spatial resolutions, numbers of pressure levels and variables.

- IFS Numerical simulation (ECMWF): 65 minutes on 352 high-end CPU for a 10-day forecast
- Aurora:
  - Inference: less than 1 mn on one A100 GPU roughly a ×5,000 speedup over IFS
  - Pre-training, 2.5 weeks on 32 A100

ECMWF – AIFS ensemble probbailistic model

- Trained using a specific loss function (CRPS) and a noise component

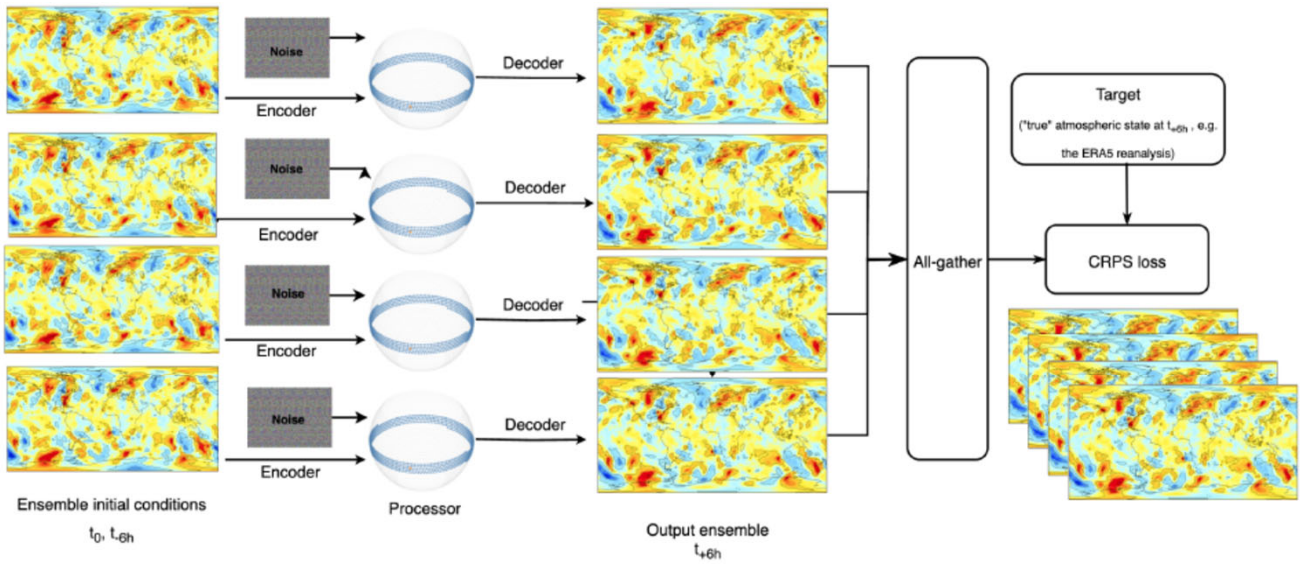


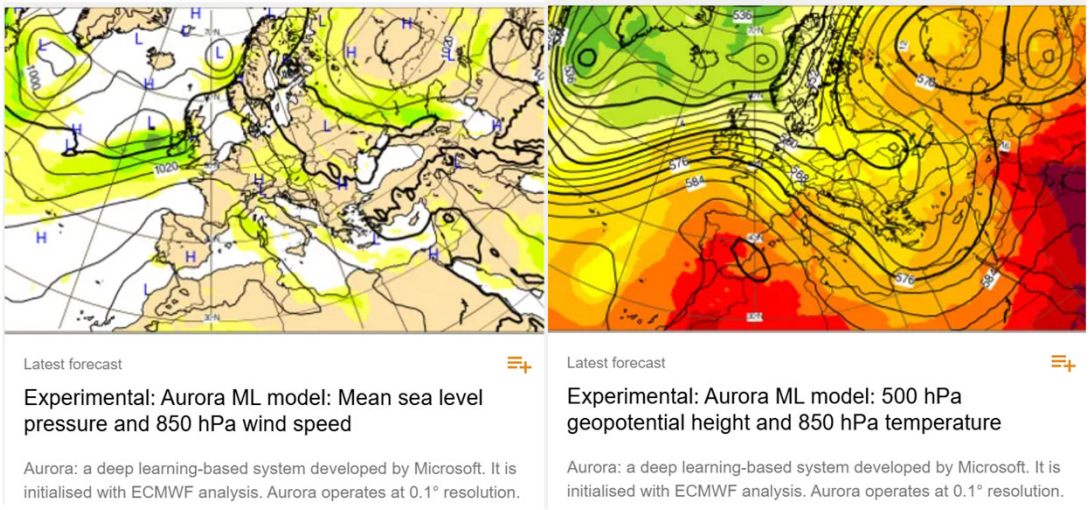
Figure 1: Probabilistic training of AIFS-CRPS. A small ensemble of atmospheric states is propagated forward in time using separate model instances (that share the same weights). With ensemble sharding (see section 2.4), the ensemble forecasts are then gathered across all participating GPU devices using a differentiable all-gather operation. Finally, the (almost fair) CRPS loss is calculated from the AIFS-CRPS forecast ensemble and a deterministic analysis (e.g., ERA5) target.

AI as digital twins
Foundation models for weather forecasting: GraphCast (Lam et al. 2023) - Google

- ECMWF is running a series of data-driven forecasts as part of its experimental suite.
  - Quote: "These ML-based weather forecasts first approached the skill of the IFS (used as the benchmark for high-quality forecasting), then matched IFS skill, and then claimed to surpass our scores. What's more, making a forecast with these models requires only a single GPU, takes less than a minute, and consumes a tiny fraction of the energy required for an IFS forecast."

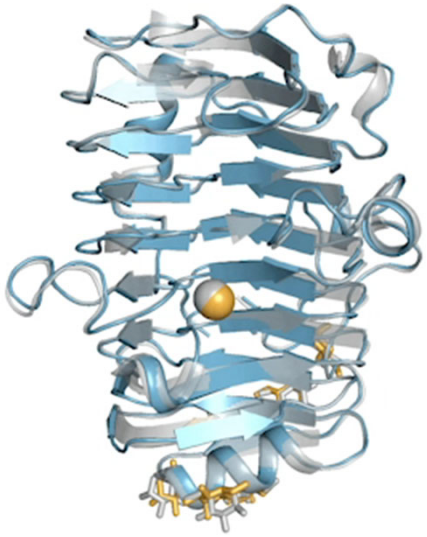
[More ML models]

These models are free and can be downloaded



Biology – Alphafold series

- Alphafold 1
  - 2018, several modules trained separately
- Alphafold 2
  - 2020, tertiary protein structure, end-to-end training, model: « Evoformer » - 20 K citations
- Alphafold 3
  - 2024, structure of protein complexes with DNA/ RNA, ligands, new model: « Pairformer »
  - Input: list of molecules
  - Output: joint 3D structure
  - Applications include drug design
  - Alphafold server
- Classical techniques
  - Cristalograpy, nuclear magnetic resonnance, etc



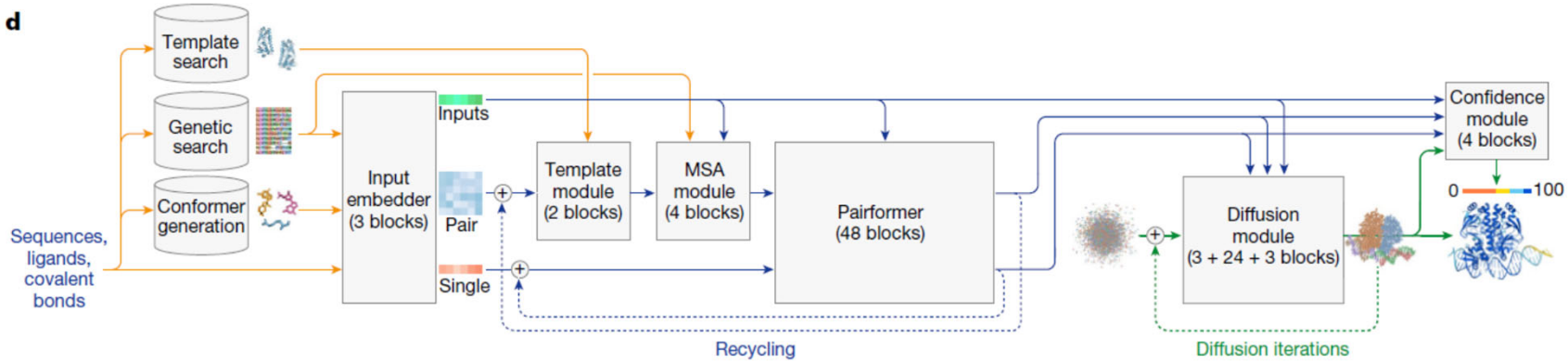
Credit: Google DeepMind
The predicted structure of this enzyme (blue) binding to a calcium ion and several monosaccharides (yellow) matches closely with the experimental structure (gray)

Biology – Alphafold series

- Foundation model

Multiple Sequence Alignment (MSA): compares the target amino acid sequence with similar sequences from other proteins

Diffusion module: predicts raw atomic coordinates of atoms from a cloud of atoms



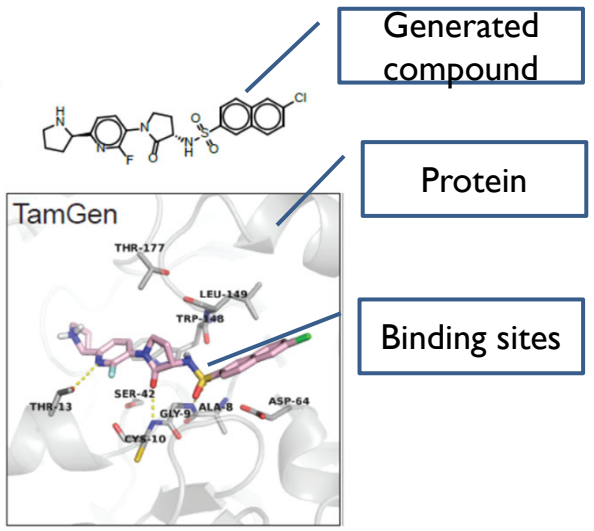
Attention Mechanism: focus on different parts of the data to understand the relationships between amino acids in the sequence and across different sequences in the MSA (Multiple-Sequence-Alignment).

Trained on multiple protein datasets, estimated in the $O(10^8)$ protein sequences and structures

AI as scientific partner: complementing physical models
Drug design and discovery, Wu et al. 2025

https://www.biorxiv.org/content/10.1101/2024.01.08.574635v1.full.pdf

- Objective
  - Drug design: Speed up the drug discovery process
  - Designing molecules/ compounds with high binding affinity to given pathogenic protein targets
- Method
  - GPT like self trained generative model
  - Operates on string representations of molecules (SMILES) + encoding of molecules geometry
  - Pre-trained on 10 millions compounds representations self supervised as for NLP
- Tests
  - Inhibitor compounds against tubercolosis



Drug design and discovery, Xia et al. 2024
https://www.biorxiv.org/content/10.1101/2024.01.08.574635v1.full.pdf

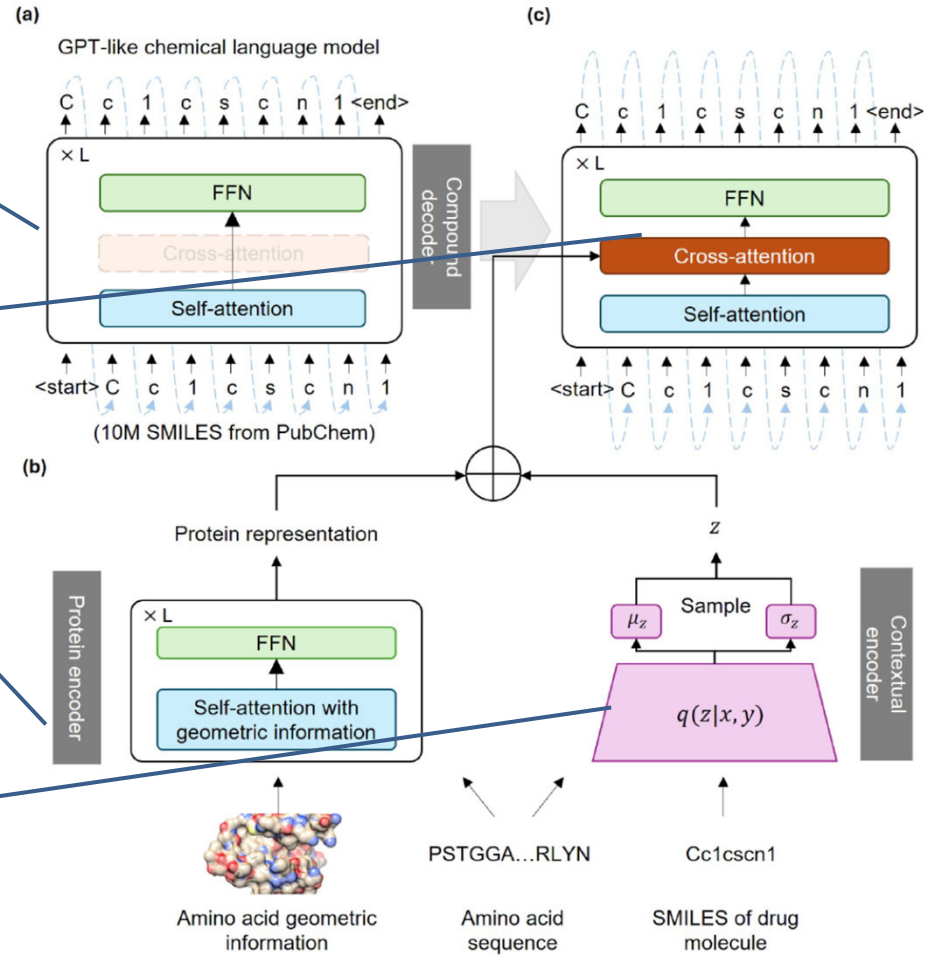
- Main architecture modules

GPT like pretrained transformer – self attention

Cross-attention: conditions generation to the target protein

Transformer - encodes the sequential and geometric data of the target protein

VAE - encodes the compound



AI as scientific partner: complementing physical models
Drug design and discovery, Xia et al. 2024
https://www.biorxiv.org/content/10.1101/2024.01.08.574635v1.full.pdf

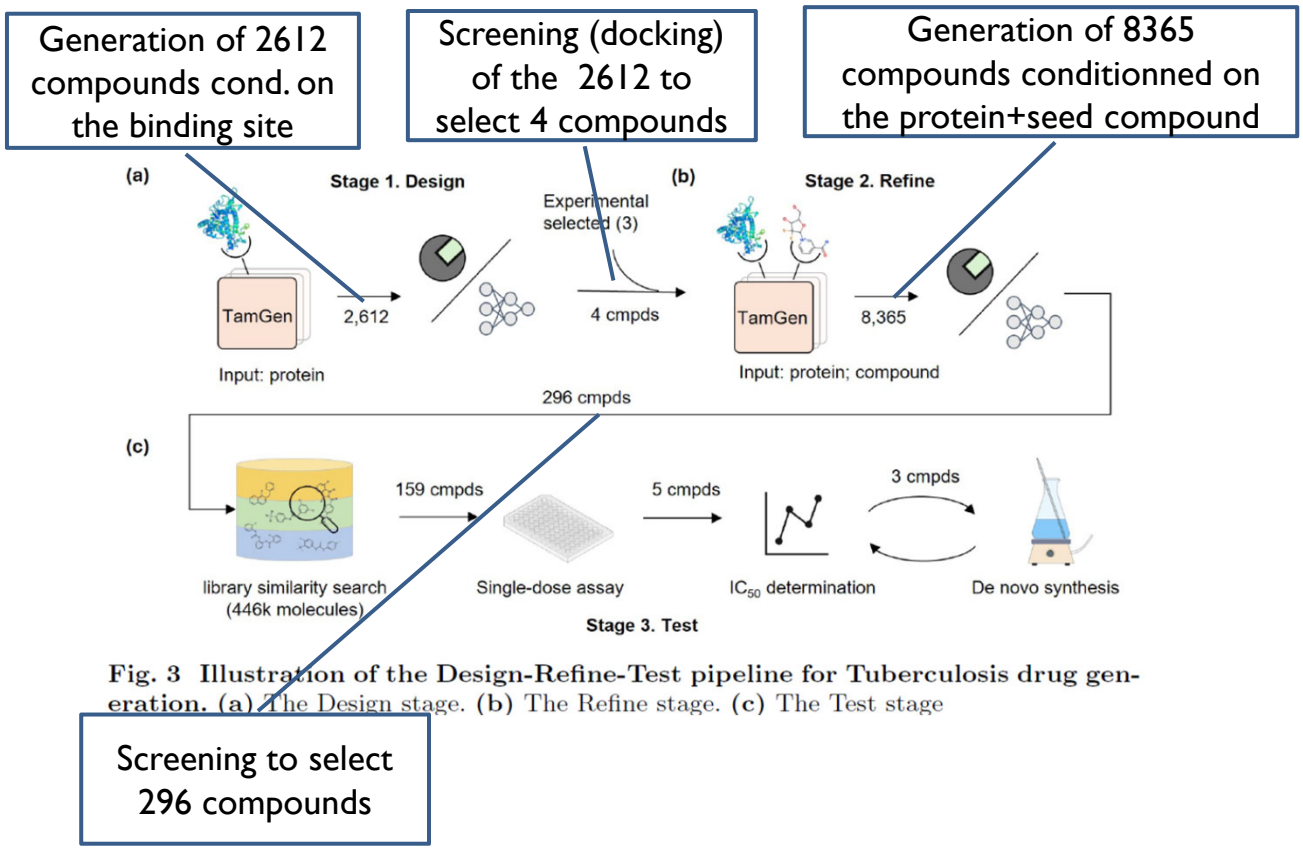


Fig. 3 Illustration of the Design-Refine-Test pipeline for Tuberculosis drug generation. (a) The Design stage. (b) The Refine stage. (c) The Test stage

Material science
A foundation model for atomistic materials chemistry, Batatia et al. 2024 70+ co-authors, https://arxiv.org/abs/2401.00096

- Problem:
  - Scientists use simulations to study materials at the atomic level which is crucial for developing new materials with desired properties.
  - Traditional methods like Density Functional Theory (DFT) are highly accurate but computationally expensive and slow.
- The Role of Machine Learning:
  - ML models can achieve near-DFT accuracy but require significant effort to develop and are often specific to particular materials.
- Objective:
  - Foundation Model: a general-purpose ML model trained on a large dataset of 150,000 inorganic crystal structures.
  - Versatility: this new model can be applied to a wide range of materials and scenarios without needing to be specifically retrained for each new case.

Material science

A foundation model for atomistic materials chemistry, Batatia et al. 2024 70+ co-authors, https://arxiv.org/abs/2401.00096 - Applications

- Material Property Prediction
  - Solids, Liquids, and Gases: Simulate the behavior of materials in various states, helping to predict properties like hardness, melting points, and thermal conductivity.
  - Chemical Reactions: By modeling the interactions between atoms during chemical reactions, the model helps in understanding reaction mechanisms and predicting the outcomes of chemical processes.
- Interface Dynamics:
  - Material Interfaces: The model can simulate the interactions at the boundaries between different materials, which is crucial for understanding and designing composite materials and coatings.
- New Material Discovery:
  - High-Throughput Screening: The model enables high-throughput computational screening of new materials, accelerating the discovery and optimization of materials with specific desired properties, such as superconductors, battery materials, and catalysts.

Context - AI for Science - Material science

A foundation model for atomistic materials chemistry, Batatia et al. 2024, 70+ co-authors, https://arxiv.org/abs/2401.00096

Pre-training
150K inorganic crystals
Predict the potential energy

Generalizes to multiple downstream tasks

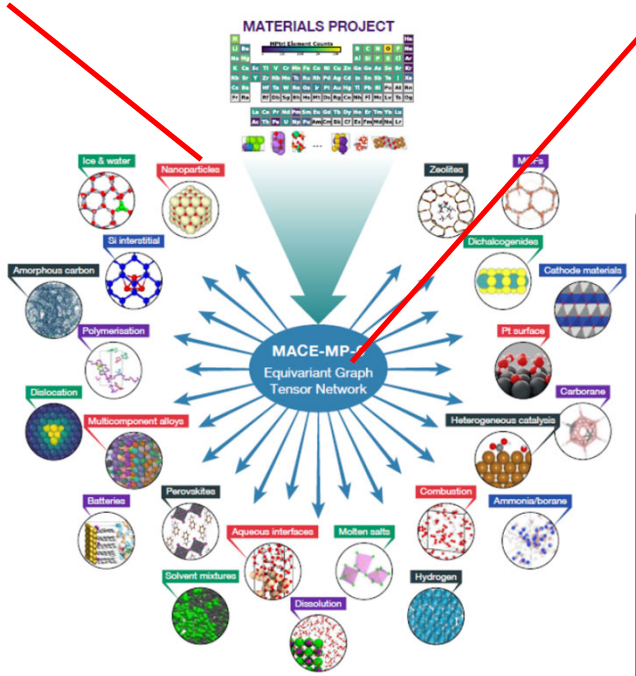


Figure 1: **A foundation model for materials modelling.** Trained only on Materials Project data (19 which consists primarily of inorganic crystals and is skewed heavily towards oxides, MACE-MP-0 is capabl of molecular dynamics simulation across a wide variety of chemistries in the solid, liquid and gaseous phases

Baselines
Density Functional Theory
Intensive and costly

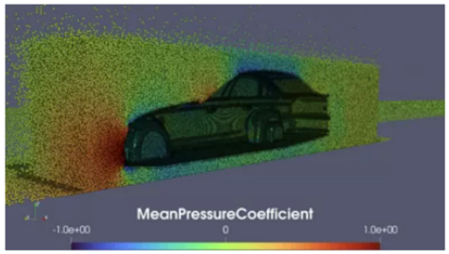
Downstream tasks

Prediction
Material properties
Chemical reactions

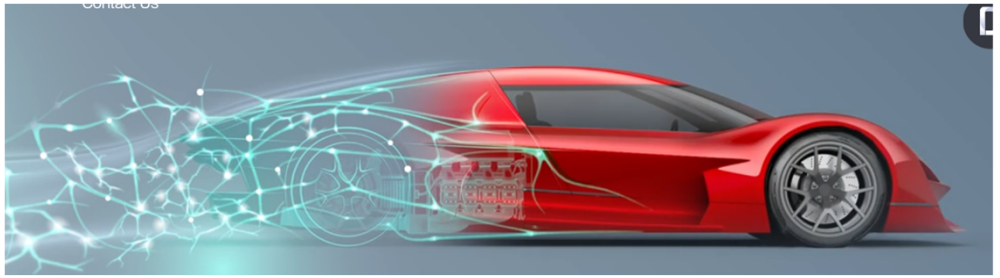
Material discovery
Superconductors
Battery material

Simulation in the industry
Deep NN as emulators for physical processes

- ANSYS Sim-AI
  - https://www.ansys.com/fr-fr/ai



- Neural Concept
  - https://www.neuralconcept.com/



Simulation in academy: the rise of foundation models
DPOT - Hao et al. (ICML 2024) http://arxiv.org/abs/2403.03542

Objective: simulate multiple physics dynamics

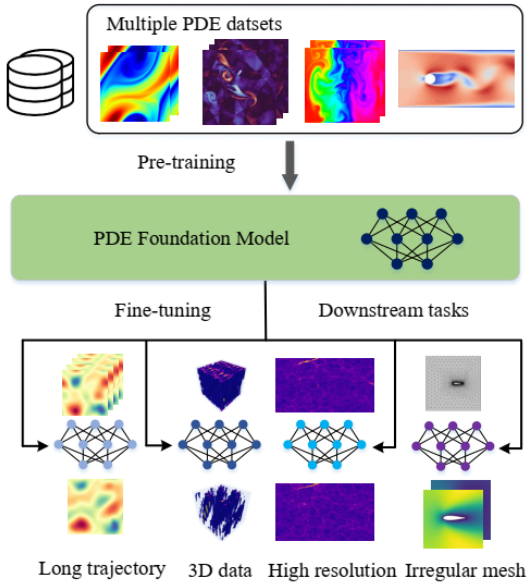


Figure 1. An illustration of pre-training a PDE foundation model using massive data from multiple PDE datasets. The pre-trained model is then used for fine-tuning different downstream operator learning tasks, which can be complex. (Best viewed in color)

- Challenges
  - Trajectories with uneven length and different time frames
  - Different resolutions, number of variables (channels), shape
  - Unbalanced datasets from the different PDE simulations

Simulation in academy: the rise of foundation models
DPOT - Hao et al. (ICML 2024) http://arxiv.org/abs/2403.03542

Model: large scale transformer

Data padding, masking, sampling to get a unified representation

Pretrained operator auto-regressive model

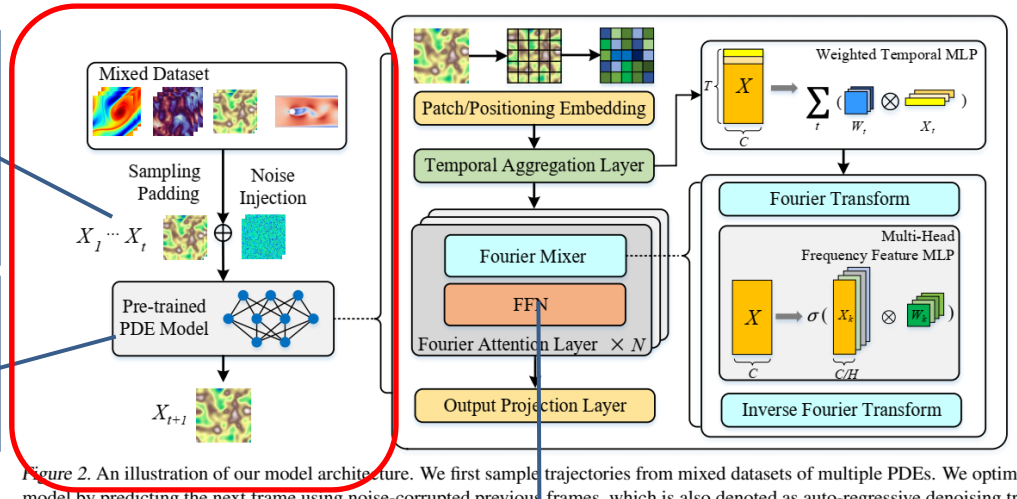


Figure 2. An illustration of our model architecture. We first sample trajectories from mixed datasets of multiple PDEs. We optimize the model by predicting the next frame using noise-corrupted previous frames, which is also denoted as auto-regressive denoising training. We design a new model architecture consisting of a temporal aggregation layer and multiple Fourier attention layers. They can extract spatial-temporal features efficiently and can be easily scaled up to large models. (Best viewed in color)

Tranformer in a spectral space

Simulation in academy: the rise of fundation models
DPOT - Hao et al. (ICML 2024) http://arxiv.org/abs/2403.03542

- Data
  - The model is trained on 12 datasets, with 100k+ trajectories
  - Different model sizes: from 7M to 1 B parameters
  - Claim a 100.000 acceleration factor w.r.t. classical solvers

DPOT: Auto-Regressive Denoising Operator Transformer for Large-Scale PDE Pre-Training

L2RE	Params	FNO-ν			PDEBench CNS,DR,SWE								PDEArena		CFDBench
Subset	–	1e-5	1e-4	1e-3	1,0.1	1,0.01	M1	0.1,0.1	0.1,0.01	M0.1	DR	SWE	NS	NS-cond	–
Small Model															
FNO	0.5M	0.156	0.0834	0.0128	0.098	0.096	0.097	0.360	0.170	0.265	0.12	0.0044	0.0912	0.319	0.00761
UNet	25M	0.198	0.119	0.0245	0.334	0.291	0.313	0.569	0.357	0.463	0.0971	0.0521	0.102	0.337	0.209
FFNO	1.3M	0.121	0.0503	0.0099	0.0212	0.052	0.0366	0.162	0.0452	0.104	0.0571	0.0116	**0.0839**	0.602	0.00714
GK-T	1.6M	0.134	0.0792	0.0098	0.0341	0.0377	0.0359	0.0274	0.0366	0.0320	0.0359	0.00692	0.0952	0.423	0.0105
GNOT	1.8M	0.157	0.0443	0.0125	0.0325	0.0420	0.0373	0.0228	0.0341	0.0285	0.0311	0.00678	0.172	0.325	0.00877
Oformer	1.9M	0.1705	0.0645	0.0104	0.0417	0.0625	0.0521	0.0254	0.0205	0.0229	0.0192	0.00717	0.135	0.332	0.0102
FNO-m	7M	0.116	0.0922	0.0156	0.151	0.108	0.130	0.230	0.076	0.153	0.0321	0.00912	0.210	0.384	0.0274
MPP-Ti	7M	–	–	–	–	–	0.0442	–	–	0.0312	0.0168	0.0066	–	–	–
MPP-S	30M	–	–	–	–	–	0.0319	–	–	0.0213	**0.0112**	**0.0024**	–	–	–
Ours-Ti	7M	0.0976	0.0606	**0.00954**	0.0173	0.0397	0.0285	0.0132	0.0220	0.0176	0.0321	0.00560	0.125	0.384	0.00952
Ours-S	30M	**0.0553**	**0.0442**	0.0131	**0.0153**	**0.0337**	**0.0245**	**0.0119**	**0.0187**	**0.0153**	0.0379	0.00657	0.0991	**0.316**	**0.00696**
Pre-trained															
MPP-L	400M	–	–	–	–	–	0.0208	–	–	0.0147	**0.0098**	**0.00220**	–	–	–
Ours-S	30M	0.0553	0.0442	0.0131	0.0153	0.0337	0.0245	0.0118	0.0188	0.0153	0.0379	0.00657	0.0999	0.316	0.00696
Ours-M	122M	0.0409	0.0285	**0.00474**	0.0116	0.0238	0.0177	**0.00866**	0.0129	0.0108	0.0292	0.00290	0.0812	0.276	0.00752
Ours-L	500M	**0.0550**	**0.0274**	0.00528	**0.0100**	**0.0216**	**0.0158**	0.00872	**0.0115**	**0.0101**	0.0232	0.00233	**0.0798**	**0.240**	**0.00650**
DPOT-FT															
T-200	7M	0.0511	0.0431	0.00729	0.0136	0.0238	0.0187	0.0168	0.0145	0.0157	0.0194	0.00280	0.103	0.313	0.00537
S-200	30M	0.0449	0.0342	0.00680	0.0152	0.0211	0.0182	**0.0150**	0.0151	0.0151	0.0171	0.00224	0.0892	0.290	0.00442
M-200	100M	0.0255	0.0144	0.00427	0.0123	0.0179	0.0151	0.0182	0.0117	0.0149	0.0142	0.00218	0.0329	0.191	0.00452
L-200	400M	0.0235	0.0117	0.00383	0.0114	0.0153	0.0133	0.0171	0.0108	0.0140	0.0158	0.00197	0.0307	0.182	0.00480
T-500	7M	0.0520	0.0367	0.00580	0.0112	0.0195	0.0153	0.0174	0.0138	0.0156	0.0148	0.00241	0.0910	0.280	0.00391
S-500	30M	0.0322	0.0237	0.00437	0.0129	0.0167	0.0148	0.0152	0.0126	0.0139	0.0129	0.00235	0.0867	0.268	0.00382
M-500	100M	0.0229	0.0126	0.00335	**0.00998**	0.0146	0.0123	0.0161	0.00947	0.0128	0.0103	0.00227	0.0294	0.172	0.00373
L-500	400M	**0.0213**	**0.0104**	**0.00323**	0.0108	**0.0131**	**0.0119**	0.0160	**0.00905**	**0.0125**	**0.00739**	**0.00170**	**0.0278**	**0.170**	**0.00322**

AI as a reasoning engine
AI4Math

Datasets and problems

- High school problems
  - MATH benchmark (2021)
  - GSM8K benchmark (2021)
  - AIME benchmark (2024)
- Complex mathematical problems
  - IMO International Mathematical Olympiad (2024)
  - Frontier maths (2025)
  - USAMO (may 2025)

Methods

- LLMs or codeLLMs pretrained and finetuned on step by step exercise solutions
  - e.g. Minerva (2022)
- Combination of LLMs and formal representations s.a. LEAN symbolic component
  - Alphaproof (2024)

AI as a reasoning engine
AI4Math
Example MATH Benchmark

- Dataset
  - 12.5K problems from high school competitions + large pretraining dataset

Problem: The equation $x^2 + 2x = \overline{i}$ has two complex solutions. Determine the product of their real parts.

Solution: Complete the square by adding 1 to each side. Then $(x+1)^2 = 1 + i = e^{\frac{i\pi}{4}}\sqrt{2}$, so $x + 1 = \pm e^{\frac{i\pi}{8}}\sqrt[4]{2}$. The desired product is then $\left(-1 + \cos\left(\frac{\pi}{8}\right)\sqrt[4]{2}\right)\left(-1 - \cos\left(\frac{\pi}{8}\right)\sqrt[4]{2}\right) = 1 - \cos^2\left(\frac{\pi}{8}\right)\sqrt{2} = 1 - \frac{(1+\cos(\frac{\pi}{4}))}{2}\sqrt{2} = \boxed{\frac{1-\sqrt{2}}{2}}$.

- Models: LLMs

Yang et al. 2024

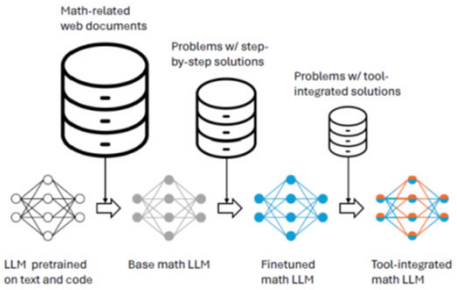


Figure 1: State-of-the-art math LLMs such as NuminaMath [49] typically undergo three stages: math pretraining, finetuning on step-by-step solutions, and further finetuning on tool-integrated solutions that interleave natural language reasoning with Python tool invocation.

Performances

- 2021 Hendriycks et al. 2021

Model	Prealgebra	Algebra	Number Theory	Counting & Probability	Geometry	Intermediate Algebra	Precalculus	Average
GPT-2 0.1B	5.2	5.1	5.0	2.8	5.7	6.5	7.3	5.4 +0%
GPT-2 0.3B	6.7	6.6	5.5	3.8	6.9	6.0	7.1	6.2 +15%
GPT-2 0.7B	6.9	6.1	5.5	5.1	8.2	5.8	7.7	6.4 +19%
GPT-2 1.5B	8.3	6.2	4.8	5.4	8.7	6.1	8.8	6.9 +28%
GPT-3 13B*	4.1	2.4	3.3	4.5	1.0	3.2	2.0	3.0 −44%
GPT-3 13B	6.8	5.3	5.5	4.1	7.1	4.7	5.8	5.6 +4%
GPT-3 175B*	7.7	6.0	4.4	4.7	3.1	4.4	4.0	5.2 −4%

Table 2: MATH accuracies across subjects. '*' indicates that the model is a few-shot model. The character 'B' denotes the number of parameters in billions. The gray text indicates the *relative* improvement over the 0.1B baseline. All GPT-2 models pretrain on AMPS, and all values are percentages. GPT-3 models do not pretrain on AMPS due to API limits. Model accuracy is increasing very slowly, so much future research is needed.

- 2025 vals.ai/benchmarks

	Model (44)	Accuracy	Cost In / Out	Latency (s)
1	Gemini 2.5 Pro Exp ★	95.2%	$1.25 / $10.00	25.83 s
2	o3	94.6%	$10.00 / $40.00	16.59 s
3	Qwen 3 (235B) $	94.6%	$1.20 / $1.20	142.75 s
4	Grok 3 Mini Fast High Reasoning	94.2%	$0.60 / $4.00	22.77 s
5	o4 Mini ⚡	94.2%	$1.10 / $4.40	12.54 s

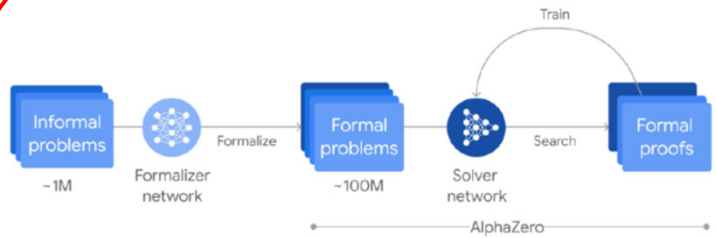
Context: AI as a reasoning engine
AI4Math
Example Alphaproof (DeepMind)

International Mathematical Olympiad

Silver medal 2024

Gold medal 2025

- Algebra + number theory pb
- Combines pretrained LLM with AlphaZero reinforcement learning algorithm
- Prooves math. statements in the formal language « Lean » - formal proovers
- Gemini is used to translate informal statements into formal language
- Trained on millions of problems



Process infographic of AlphaProof's reinforcement learning training loop: Around one million informal math problems are translated into a formal math language by a formalizer network. Then a solver network searches for proofs or disproofs of the problems, progressively training itself via the AlphaZero algorithm to solve more challenging problems.

Neural networks and ordinary differential equations

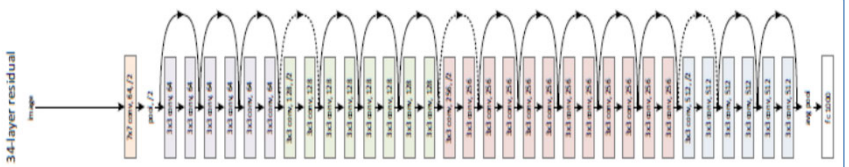
✓ NNs as numerical schemes for solving ODEs

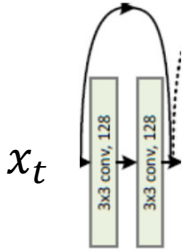
NNs as numerical schemes for solving ODEs - summary

- The dynamics of Neural Networks – explained by ODE
  - NNs with an infinite number of layers can be modeled as ordinary differential equations (ODE)
  - Inference and training can be formulated as solving ODEs
  - NNs interpretation as numerical schemes for solving ODEs
  - Opens the way to the
    - Use of numerical ODE solvers for a variety of ML problems
    - Use of ODE numerical solvers theory for analyzing NNs dynamics – e.g. stability – convergence
  - This helped popularize the use of differentiable numerical solvers in the ML community
    - Implemented in DL libraries, e.g. PyTorch
    - Opens the way to integrating physics and ML:
      - Physics-aware deep learning

NNs as numerical schemes for solving ODEs

- Several NNs use skip connections, e.g. ResNet





Resnet Module

x_t $\quad x_{t+1} = x_t + f(x_t, \theta_t)$

Input x is progressively modified by a residual $f(x, \theta)$

- ODE for initial value problem
  - $\frac{dx}{dt} = f(x(t); \theta(t))$ for $t \in [0, T]$, $x(0) = x_0$
    - What is the value of $x(T)$?
- Equivalent integral formulation
  - $x(T) = x(0) + \int_0^T f\left(t, x(t)\right)dt$
  - $\int_0^T f\left(t, x(t)\right)dt$ is approximated via numerical integration
  - Exemple: Euler numerical scheme
    - $x_{t+1} = x_t + hf(x_t, \theta_t), x(0) = x_0$

Forward pass of ResNet is similar to Euler scheme for solving IVP (E 2017, Haber 2017, Chang 2018, Lu 2018, …)

NNs as numerical schemes for solving ODEs – Learning problem

- Learning problem with ResNets
  - $Min_\theta \;\; L(F(x,\theta),y)$
    $s.t. \quad x_l = x_{l-1} + f_l(x_{l-1}) \,, l = 1 \ldots T, x_0 = x$

    ← The constraint describes the Forward graph of the Resnet
  - x input, y target, θ parameters, x_l layer l activation, T layers
  - Solving this problem requires alternating
    - Forward pass – Euler numerical scheme for solving
      - $\frac{dx}{dt} = f(x(t),\theta(t))$ for $t \in [0,T], x(0) = x_0$
    - Backward pass – differentiation through Euler scheme for solving
      - $\frac{d\theta}{dt} = -\epsilon \frac{\partial L(\theta(t))}{\partial \theta}, \theta(0) = \theta_0$
- Could this idea be generalized?
  - Replace Euler with any numerical integration scheme

NNs as numerical schemes for solving ODEs
Euler derivation

- ODE – IVP problem
  - $\frac{dx}{dt} = f(x(t))$ for $t \in [0, T], x(0) = x_0$
- Continuous to discrete time
  - Divide $[0, T]$ in intervals of size Δt: $t_n = n\Delta t$
  - The objective is to find x_n an approximation of $x(t_n)$ at each t_n
- Taylor expansion
  - $x(t_{n+1}) \approx \mathrm{x}(\mathrm{t_n}) + \Delta t \frac{dx(t_n)}{dt}$
  - $x(t_{n+1}) \approx \mathrm{x}(\mathrm{t_n}) + \Delta t f(x(t_n))$
- Discrete approximation and algorithm
  - $x_0 = x(0)$
  - $x_{n+1} \approx x_n + \Delta t f(x_n)$

NNs as numerical schemes for solving ODEs
ODE formulation of a gradient algorithm

- Steepest gradient descent
  - $\theta_{t+1} = \theta_t - \epsilon_t \nabla L(\theta_t)$, with initial value θ_0
- Continuous formulation
  - Let $\epsilon_t = \epsilon(t)\Delta t$
  - $\theta(t+1) = \theta(t) - \epsilon(t)\Delta t \nabla L(\theta(t))$
  - $\frac{\theta(t+1)-\theta(t)}{\Delta t} = -\epsilon(t)\Delta t \nabla L(\theta(t))$
  - $\frac{\partial \theta(t)}{\partial t} = -\epsilon(t) \nabla L(\theta(t))$
- ODE IVP
  - $\frac{\partial \theta(t)}{\partial t} = -\epsilon(t) \nabla L(\theta(t))$ with $\theta(0) = \theta_0$

NNs as numerical schemes for solving ODEs
Continuous limit

- Continuous limit
  - If we let $h \to 0$ in Euler, the ResNet learning problem becomes
  - $Min_\theta L(F(x,\theta),y)$
    - $s.t. \quad \frac{\partial x}{\partial t} = F\left(x(t),\theta(t)\right), t \in [0,T]\ , x_0 = x$
- Two different families of methods for solving the learning problem:
  - Discretize then Optimize
    - Discretize in time and then solve
    - Leads to back-propagation like algorithms
    - The ResNet derivation described before is an example
    - This is the framework used in this course
  - Optimize then Discretize
    - Solves the continuous optimization problem
    - Advocated by NeuralODE (Chen 2018)
    - Amounts at solving a forward and a backward differential equations
      - See notes in the next slides

NNs as numerical schemes for solving ODEs

- Key ideas
  - Training of NNs can be formulated as solving ODEs with a numerical scheme
  - Different numerical schemes could be used
    - To be implemented with specific NN architectures
  - Allows us using numerical schemes theory for deriving NN properties
  - The link between NNs and differential equations will be most relevant for modeling dynamical systems

  - Note
    - ODE are central in several ML contexts involving dynamical processes such as generative models (e.g. diffusion models, flow matching models)

Interlude
Crash notes on ODEs and PDEs

ODEs

Crash notes on ODEs

- Initial value problem
  - $\begin{cases} \frac{\partial x}{\partial t} = f\ (t, x(t)) \\ x(0) = x_0 \end{cases} \quad (1)$
    - With $f: [0, T] \times R^n \rightarrow R^n$ differentiable and $x_0 \in R^n$ an initial value
  - What is the value of $x(T)$?

- Integral formulation: solution of (1)
  - $x(T) = x(0) + \int_O^T f\ \big(t, x(t)\big) dt$
  - Property: the integral formulation is equivalent to formulation (1)

Example

$$\frac{\partial x}{\partial t} = 2t; x(0) = 1; x(1)?$$

$$x(1) = x(0) + \int_0^1 2t dt$$

$$x(1) = 1 + 1^2 - 0^2 = 2$$

Crash notes on ODEs

- Property (Cauchy- Lipschitz)
  - If f is uniformly Lipschitz w.r.t. t and globally w.r.t. variable x, ($\|f((t,x) - f(t,x')\| \leq L\|x - x'\|$) in a neigborhood of $(0, x_0)$, then a solution exists and is unique
- Corollary
  - If f is continuously differentiable w.r.t. t, x, the solution to the initial value problem is unique
- Geometrical interpretation
  - Solution curves for different solutions (initial values) do not intersect

Crash notes on ODEs

- Trajectories (solution curves)
- Flow of an ODE
  - $\phi : R \times R^n \rightarrow R^n$ of f is defined by $\phi(t, x_0) = x(t)$
- Geometric Interpretation
  - The flow traces the trajectory of the solution in the state space:

    $$\{\phi_t(x_0) : t \in R\}$$
  - These trajectories are solutions of the ODE and follow the vector field f, satisfying:

    $$\frac{d\phi_t}{dt} = f\left(\phi_t(x_0)\right)$$

i.e. it describes all the trajectories for different initial conditions

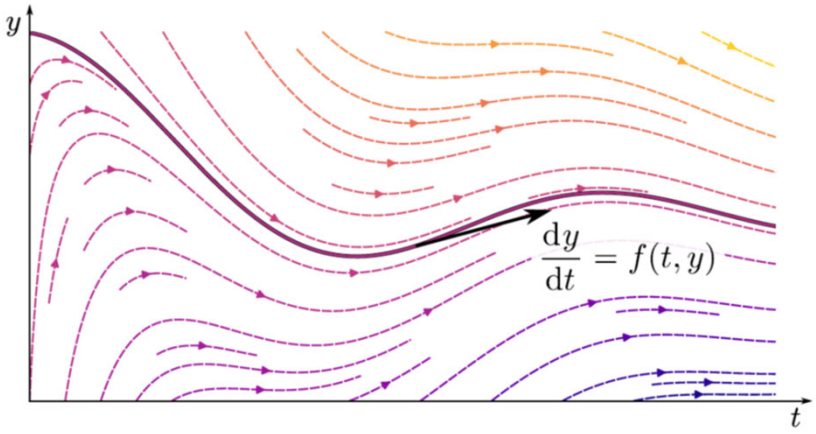
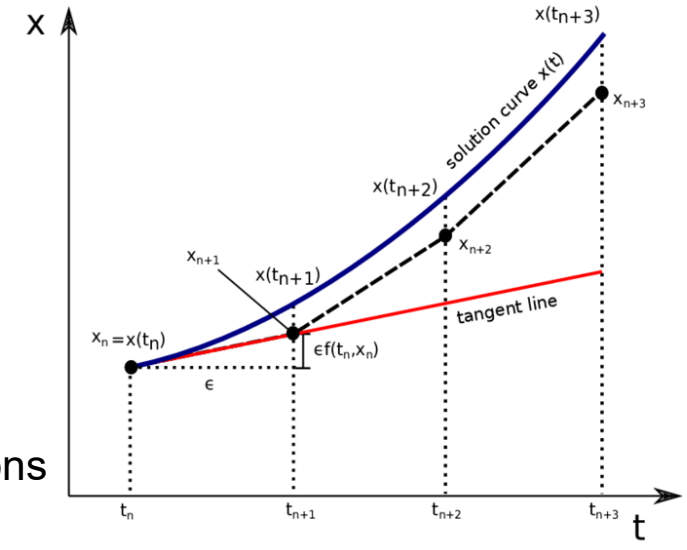


Figure 2.2.: Illustration (in dashed lines) of the continuous flow of an ODE, with a particular solution that is plotted with a solid thicker line. The black arrow represents the tangent to the highlighted solution, which is fully determined by its derivative and initial condition according to variants of the Cauchy-Lipschitz theorem (Demailly, 2006, Chapter V, Section 3.4). In this example, f is defined as $f : (t, y) \mapsto \frac{1}{t}(\cos t - y)$, the ODE admitting as solutions over $I = (0, +\infty)$ functions $y_C : t \mapsto \frac{C}{t} + \operatorname{sinc} t$ for all $C \in \mathbb{R}$.

Crash notes on ODEs

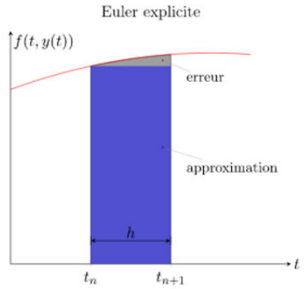
- Numerical solvers
  - $x(T) = x(0) + \int_0^T f\left(t, x(t)\right)dt$
    - What if the integral cannot be analytically integrated?
  - $\int_0^T f\left(t, x(t)\right)dt$ is approximated via numerical integration
    - Objective: build a sequence of values $x_0, x_1, \ldots x_N$ that approximate the solution at the discretization points $x(t_0), x(t_1), \ldots, x(t_N)$
- Exemple: Euler forward
  - Step size h
    - $t_{n+1} = t_n + h$
  - Update using the gradient at $f(t_n)$
    - $x_{n+1} = x_n + hf(x_n, \theta_n)$



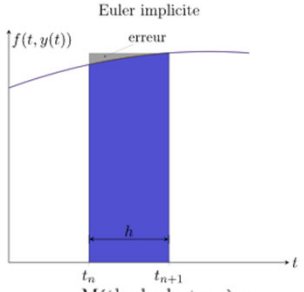
Note: the same solver can be recovered also via the differential formulation through derivative approximations e.g. $\frac{\partial x}{\partial t} \simeq \frac{x(t+h)-x(t)}{h}$ leads to $x_{n+1} = x_n + hf(t_n, x_n)$

Crash notes on ODEs

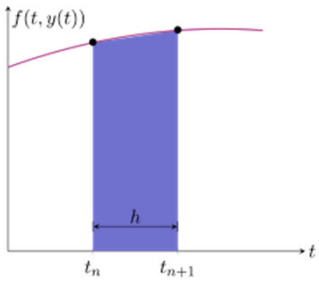
- One step methods - exemples
  - $x_{n+1} = x_n + h_n \phi(t_n, x_n, h_n)$ with ϕ a function depending on f
  - Euler forward (explicit)
    - $x_{n+1} = x_n + hf(t_n, x_n)$



  - Euler backward (implicit)
    - $x_{n+1} = x_n + hf(t_{n+1}, x_{n+1})$
    - Requires solving a fixed point equation



  - Runge Kutta e.g. RK2
    - (explixit, RK4 often used as a default option)
  - $$\begin{cases} x_{n,1} = x_n \\ x_{n,2} = x_n + hf\left(t_n, x_{n,1}\right) \\ x_{n+1} = x_n + \frac{h}{2} f\left(t_n, x_{n,1}\right) + \frac{h}{2} f\left(t_{n+1}, x_{n,2}\right) \end{cases}$$



Crash notes on ODEs
NNs as numerical schemes for solving ODEs

- NN architectures motivated by ODE numerical schemes
  - This link between numerical schemes and NNs has been exploited by some authors
    - Different discretisation methods used in place of Forward Euler
  - Linear multi-step (Lu et al. 2018)
    - $x_{t+1} = (1 - k_t)x_t + k_t x_{t-1} + f(x_t; \theta_t)$, θ_t are the parameters of f
  - Leapfrog Network (Chang et al. 2018)
    - $x_{t+1} = 2x_t - x_{t-1} - h^2 f(x_t, \theta_t)$
  - …
  - Implicit schemes
    - e.g. backward Euler scheme
      - $x_{t+1} = x_t + hf(x_{t+1}; \theta_{t+1})$
      - Note: requires solving a non linear equation at each step
  - Each numerical scheme leads to a specific NN architecture (a la ResNet)

Crash notes on ODEs

- Summary: using numerical solvers for ODEs
  - consider the initial value problem
  - $$\begin{cases} \frac{\partial x}{\partial t} = f\ (t, x(t)) \\ x(t_0) = x_0 \end{cases} \qquad (1)$$
  - What is the value of $x(t_1)$?
  - Note: we introduced here t_0 and t_1, lket us use $t_0 = 0$ and $t_1 = T$
  - Solver call for the forward pass

$$x(t_1) = ODESolve(f(x(t), t, \theta), x(t_0), t_0 = 0, t_1 = T)$$

Numerical solver | Differential | Initial value | Initial time | Final time

Crash notes on ODEs

- Properties
- For simplicity we consider one step methods of the form
  - $x_{n+1} = x_n + h_n \phi(t_n, x_n, h_n)$ (2)
  - Stability
    - Intuition: a perturbation of the initial value and of the ϕ term does not lead to a divergence of the schema
    - Property
      - If there exists $L > 0$ such that $\forall x, x' \in R^m, \forall h \in [0,1], \forall t \in [0, T]$, $\|\phi(t, x, h) - \phi(t, x', h)\| \leq L\|x - x'\|$ then the numerical scheme is stable
      - i.e. ϕ is Lipschitz continuous w.r.t. x, uniformly w.r.t. t and h
    - Note: stability is important e.g. for the robustness of NN to adversarial attacks

Crash notes on ODEs

- Properties
- For simplicity we consider one step methods of the form
  - $x_{n+1} = x_n + h_n \phi(t_n, x_n, h_n)$ (2)
  - Consistency
    - Measures how well the sequence $x_0, x_1, \ldots x_N$ approximates $x(t_0), x(t_1), \ldots, x(t_N)$
    - Truncation error
      - $\epsilon_n = x(t_{n+1}) - x(t_n) - h\phi(t_n, x(t_n), h)$, with $x(t)$ a solution of the ODE (1)
    - A numerical scheme is consistent if the summation of all the truncation errors, for all discretization steps goes to 0 with h
  - Convergence
    - What are the conditions on ϕ for schema (2) to be convergent, i.e. for x_n to converge to x(t_n) when $h \to 0$?
    - If the scheme (2) is stable and consistent then it is convergent, meaning that
    - $\lim_{\substack{h \to 0 \\ x_0 \to x(0)}} \sup_{0 \le n \le N} \left|\left| x_n - x(t_n) \right|\right|^2 = 0$

Interlude
Crash notes on ODEs and PDEs

PDEs

Crash notes on PDEs

- We will consider general PDEs of the form

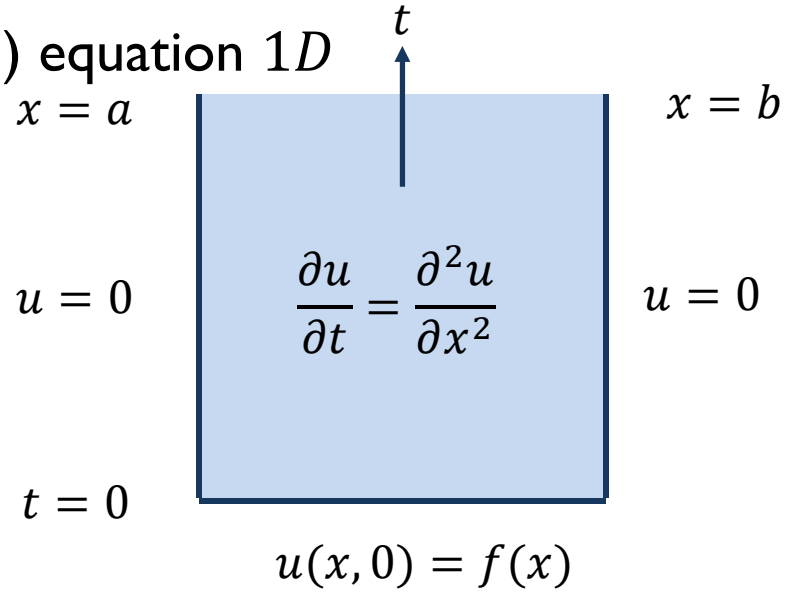
$$\begin{cases} \frac{\partial u(t,x)}{\partial t} + \mathcal{L}u(t,x) = 0, & (t,x) \in [0,T]\text{x}\Omega \\ u(0,x = u_0(x), & x \in \Omega \\ u(t,x) = g(t,x), & x \in [0,T]\text{x}\partial\Omega \end{cases}$$

  - With u: $[0,T]\text{x}\Omega \to R^n$, $\partial\Omega$ the boundary of domain Ω
  - $\mathcal{L}$ is a differentiable operator

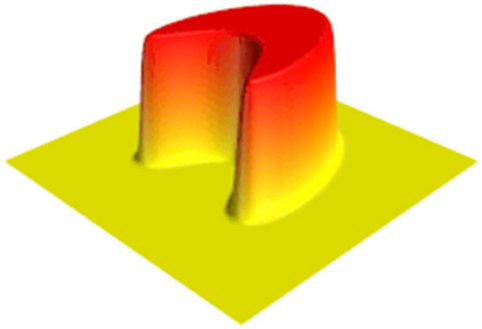
- Different types of boundary conditions, e.g.
  - Dirichlet $u(x) = g(x)$ specifies the value at the boundaries
  - Neuman $\frac{\partial u}{\partial n}(x) = g(x)$ specifies the value of the normal derivative at the boundary
  - Periodic $u(a) = u(b), \frac{\partial u}{\partial x}(a) = \frac{\partial u}{\partial x}(b), \ldots$ for example if $\Omega = [\text{a}, \text{b}]$

Crash notes on PDEs

- Example: IBVP for the Heat (Diffusion) equation $1D$
  - $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad t > 0, x \in [a, b]$
  - $u(a, t) = 0$ and $u(b, t) = 0$ for $t > 0$
  - $u(x, 0) = f(x)$



- Heat equation 2D
  - $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$



Evolution of the temperature in a square metal plate -heat equation. The height and redness indicate the temperature at each point. The initial state has a uniformly hot hoof-shaped region (red) surrounded by uniformly cold region (yellow). As time passes the heat diffuses into the cold region.

Crash notes on PDEs – method of lines

- Many applications of NN to PDE proceed as in the method of line (Hamdi et al. 2007)
  - Replace spatial derivatives in the PDE with algebraic approximations.
  - The spatial derivatives don't appear anymore and the only remaining independent variable is the time.
  - This transforms the PDE into a system of ODE that can be solved with classical ODE solvers using a time stepping scheme
  - This is the scheme used for most discrete space ML solvers

Crash notes on PDEs – method of lines

- Example Diffusion (heat) PDE in one dimension
  - $\frac{\partial u}{\partial t} = \mathrm{c}\frac{\partial^2 u}{\partial x^2}$ with boundary conditions $u(-L,t) = u(L,t)$
  - Spatial discretization of $u(x,t)$:
    - Denote $u(-L,t) = u_1, u(-L+\Delta x,t) = u_2, \ldots, u(L,t) = u_{n+1}$
- Replace spatial derivatives in the PDE with algebraic approximations.
  - Discretization of the spatial derivative with a second order scheme :

$$\frac{du_i}{dt} = \frac{c}{\Delta x^2}\left[u_{i+1} - 2u_i + u_{i-1}\right], \qquad \forall\, i \in \{1,n\}$$

- The spatial derivative do not appear anymore
  - We are left with a system of n ODEs

Crash notes on PDEs – method of lines

- This system of PDE can be solved using appropriate classical ODE solvers.
  - Let us derive the ODE integration using an Euler discretization scheme for the temporal component: $\frac{\mathrm{du_i}}{\mathrm{dt}} \approx \frac{u_i^{n+1} - u_i^n}{\Delta t}$
  - The integration scheme for the system of ODEs can be rewritten as:

$$u_i^{m+1} = u_i^m + \lambda \left(u_{i+1}^m - 2u_i^m + u_{i-1}^m\right) \quad \forall i \in \{1, n\}$$

    with $\lambda = c\frac{\Delta t}{\Delta \mathrm{x}^2}$ the CFL (Courant-Friedrichs-Lewy) number that conditions the stability of the ODE
    - The values of u at time step $m+1$ can be obtained from the values at time step m using neighbourhood points at space index u_i
    - Two dimensions, is slightly more complex, but the PDE can be reduced to a system of linear ODEs as in the example above
- End Interlude PDEs

Modeling Spatio-temporal dynamics with Neural Networks

NNs as surrogate models for solving PDEs – Discrete space models
NNs as surrogate models for solving PDEs – Continuous space models
NNs as surrogate models for solving PDEs – Data free approaches

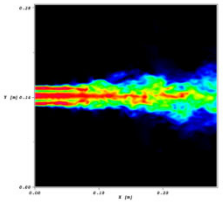
Modeling Spatio-temporal dynamics with Neural Networks Motivations

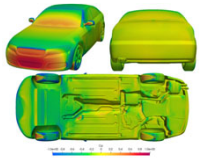
- Modeling the complex dynamics arising in natural/ physical processes
  - Objective: understanding, predicting, controling
- Physical models
  - Mathematical equations of dynamical systems
  - Often take the form of PDEs and associated numerical models
  - Stem from a deep understanding of the underlying physics
- Data driven modeling
  - In many cases data are plentiful (climate, simulations, etc)
  - Can we leverage ML for modeling these complex systems?
    - Way more complex than current ML successes (vision, language)
- Challenge: Interaction between the physical model based and the statistical paradigms

Modeling Spatio-temporal dynamics with Neural Networks

- Applications domains - examples

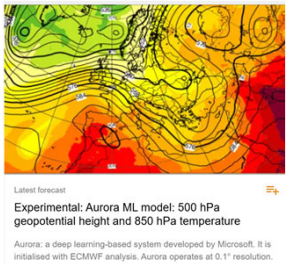
Computational Fluid Dynamics





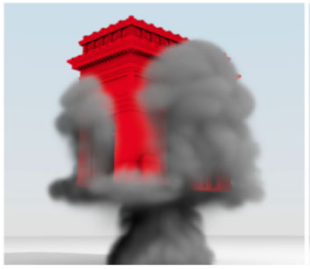
DrivaerNet 2025

Earth System Science – Weather prediction/ Climate



ECMWF 2025

Graphical design



Tompson et al. 2017

- Potential major application field for ML
- Objectives
  - Fast alternative to numerical solvers
    - Reduce computational cost and facilitate the design and deployment
    - Complement physical models
  - Focus of the presentation: pure data-driven models

Explicit modeling of physical dynamics: Partial Differential Equations

Spatio temporal dynamics are usually modeled through partial differential equations
We consider a class of spatio-temporal partial differential equations dependent on **parameters**

Explicit modeling

$$\frac{\partial u}{\partial t} = g(c, f; x, t, u, \nabla u, \nabla^2 u, \dots) \qquad \forall (x,t) \in \Omega \times \mathbb{R}_*^+$$

Partial Differential Equation

Initial constraint → $u(x, t=0) = u^0(x)$ $\quad \forall x \in \Omega$

Boundary constraint → $B(b; u, \nabla u, x, t) = 0$ $\quad \forall (x,t) \in \partial\Omega \times \mathbb{R}_*^+$

Parameters that influence the dynamics: PDE coefficients (c), forcing terms (f), boundary conditions (b), initial conditions (u^0), any physical parameter.

Implicit modeling of physical dynamics: Neural surrogates

Neural surrogates are **trained** on trajectories sampled from an underlying dynamics
We consider pure data driven approaches: no prior on the PDE form

Explicit modeling

$$\frac{\partial u}{\partial t} = g(c, f; x, t, u, \nabla u, \nabla^2 u, \dots) \qquad \forall (x,t) \in \Omega \times \mathbb{R}_*^+$$

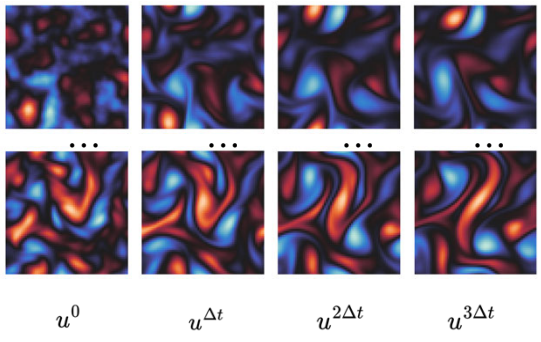
Partial Differential Equation

Initial constraint → $u(x, t=0) = u^0(x) \qquad \forall x \in \Omega$

Boundary constraint → $B(b; u, \nabla u, x, t) = 0 \qquad \forall (x,t) \in \partial\Omega \times \mathbb{R}_*^+$

Implicit modeling

Training dataset of N trajectories



→ Surrogate Model

Tackling the generalization problem for physical dynamics
Generalizing to changes in discretization and domain

Existing neural surrogates face **several challenges** for emulating the evolutions of systems governed by PDEs

Explicit modeling

$$\frac{\partial u}{\partial t} = g(c, f; x, t, u, \nabla u, \nabla^2 u, \dots) \quad \forall (x,t) \in \Omega \times \mathbb{R}_*^+$$

Initial condition

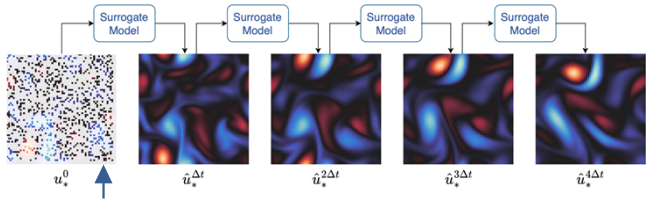
$$u(x, t=0) = u^0(x) \quad \forall x \in \Omega$$

Domain & Grid

$$B(b; u, \nabla u, x, t) = 0 \quad \forall (x,t) \in \partial\Omega \times \mathbb{R}_*^+$$

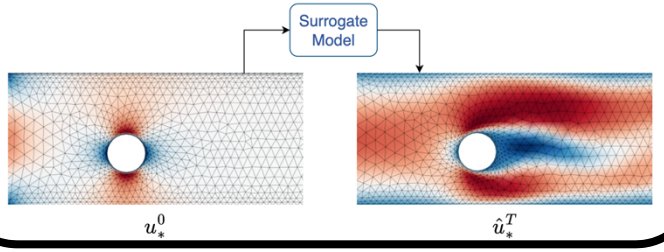
Implicit modeling

2) Neural Surrogates should generalize to changes in the discretization grid $\mathcal{X}$



Sparse grid

And to changes of domain Ω



Tackling the generalization problem for physical dynamics

Generalizing to changes in the parameters: solving parametric PDEs

Existing neural surrogates face **several challenges** for emulating the evolutions of systems governed by PDEs

PDE coefficients

Forcing terms

Explicit modeling

$$\frac{\partial u}{\partial t} = g(c, f; x, t, u, \nabla u, \nabla^2 u, \dots)$$

Initial condition

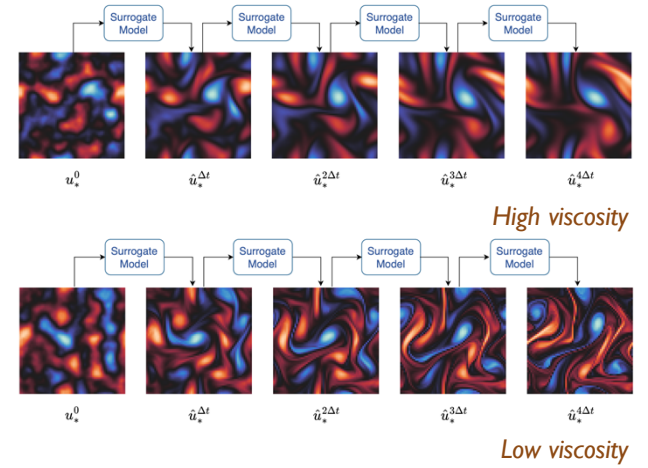
$$u(x, t = 0) = u^0(x)$$

$$B(b; u, \nabla u, x, t) = 0 \qquad \forall (x, t) \in \partial\Omega \times \mathbb{R}_+$$

Boundary type

Implicit modeling

Neural Surrogates should generalize to changes in new initial conditions and PDE parameters like forcing terms, coefficients, boundary conditions



Modeling Spatio-temporal dynamics with Neural Networks

✓ NNs as surrogate models for solving PDEs – Discrete space models
NNs as surrogate models for solving PDEs – Continuous space models
NNs as surrogate models for solving PDEs – Data free approaches

Basic NN architectures
Used on discrete grids

Convolutional Neural Networks

Unet

Graph Neural Networks

Convolutional Neural Networks

- Convolution of function $u(x), x \in R^n$ with kernel $g(x)$
  - $(u * g)(x) = \int_{Rd} u(y)g(y - x)dy$
  - $(u * g)(x) = \int_{Rd} u(y - \mathrm{x})g(y)dy$

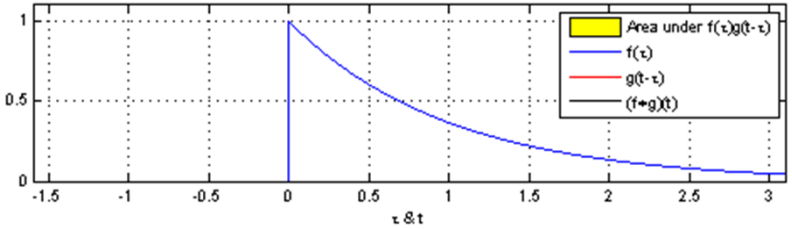
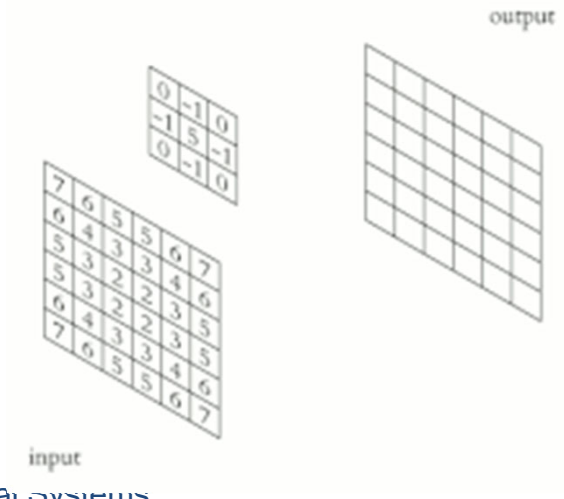


Fig. https://en.wikipedia.org/wiki/Convolution

- Discrete convolution in 2 D of $u[i, j]$ with $g[i, j]$
  - Let us suppose that g has a finite support set $\{-N, -N + 1, \ldots, N\}^2$
    - $(u * g)[i, j] = \sum_{m=i-N}^{i+N} \sum_{n=i-N}^{i+N} u[m, n]g[m - i, n - j]$
    - $\boxed{(u * g)[i, j] = \sum_{m=-N}^{N} \sum_{n=-N}^{N} u[i + m, \mathrm{j} + n]g[m, n]}$
      - The latter is the convolution used in CNNs



Convolutional Neural Networks

- $(u * g)[i,j] = \sum_{m=-N}^{N} \sum_{n=-N}^{N} u[i+m, \mathrm{j}+n] g[m,n]$

Image u

u_1	u_2	u_3	u_4
u_5	u_6	u_7	u_8
u_9	u_{10}	u_{11}	u_{12}
u_{13}	u_{14}	u_{15}	u_{16}

Filter g
indication of the indexing positions

-1,1	0,1	1,1
-1,0	0,0	1,0
-1,-1	0,-1	1,-1

y_1	y_2
y_3	y_4

$$y_1 = (u * g)[2,2] \sum_{m=-1}^{1} \sum_{n=-1}^{1} u[2+m, 2+n] g[m,n]$$

Convolutional Neural Networks

- Relations with finite difference
  - Classical stencils used for finite differences can be implemented via NN convolution operators
  - NNs operating in discrete spaces (CNN, Unet, ResNet, …) have the potential to learn differential operators
  - Example
    - Let $u: \mathrm{R}^2 \rightarrow \mathrm{R}$ be a function, and $x \in R^2$, and h the grid size of a discretized representation for x
    - Central difference operator for approximating ((i,j) respectively denote the row and the column)
    - $\frac{\partial}{\partial x_1} u(x)$ is $\frac{u[i+1,j]-u[i-1,j]}{2h}$ implemented as $\frac{1}{2\mathrm{h}}\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$
    - $\frac{\partial^2}{\partial x_1^2} u(x)$ is $\frac{u[i+1,j]-2u[i,j]+u[i-1,j]}{h^2}$ implemented as $\frac{1}{\mathrm{h}^2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$
  - Note, more on that in Long et al. 2019

UNets

- Introduced for image to image transformations (initially image segmentation, could be used to associate a source to the solution of a PDE)
- Encoder-decoder type architecture, V- Cycle:
  - First, upscale the image resolution and increase the number of channels
  - Then downscale to the initial resolution and reduce the number of channels
- Close to Multigrid numerical methods
  - Makes use of
    - Convolutions
    - skip connections combine information at the same resolution
  - Recent versions incorporate Attention mechanisms

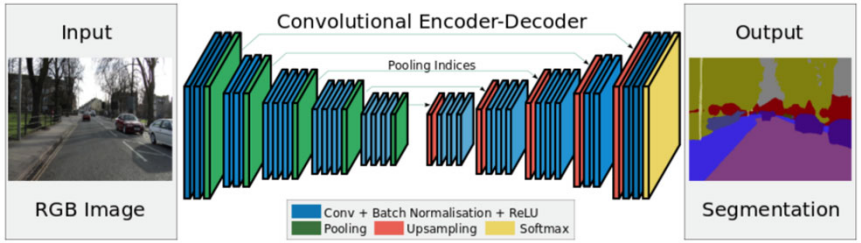


Fig. Badrinarayanan et al. 2015

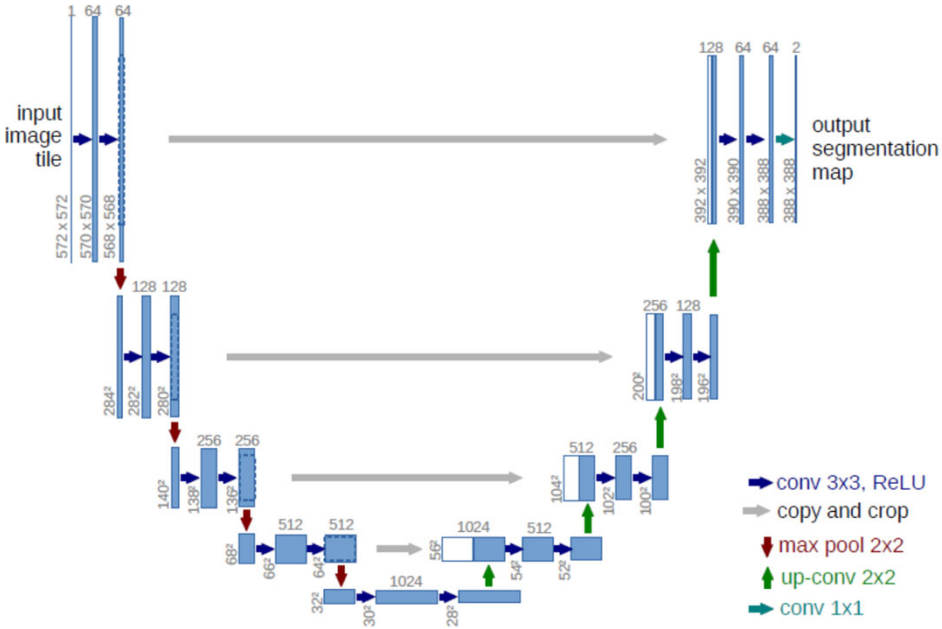
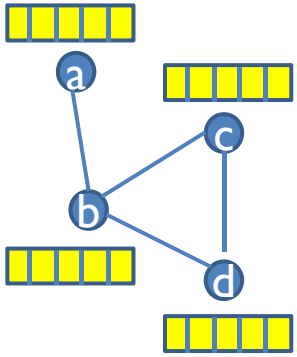


Fig. Ronneberger net al. 2015

Graph Neural Networks

- Extends the CNN ideas to irregular grids (graphs)
  - Better adapted to irregular meshes used e.g. in fluid mechanics
  - Computational cost high compared to CNNs
    - Needs to compute the neighbours for several operations – not so adapted to GPUs
  - Notations
    - $G = (V, E)$ a graph
    - To each $v \in V$, is attached a set of node features $x \in R^d$:
    - $\mathcal{N}(v)$ is the neighborhood of note v in the graph
    - Alternatively, G can be described by its adjacency matrix A



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Graph Neural Networks

- GNNs are multilayer NNs where each layer has a graph structure
  - As for CNNs, nodes in layer $k+1$ compute their activation from activations in layer k.
  - The regular CNN convolution is replaced by a message passing operation
- Message passing
  - Let x_v^k the embedding vector associated to node v at layer k
  - Message passing operation for nodes has the following general form
    - Update v state: $\boxed{x_v^{k+1} = \sigma(w^k \sum_{u \in \mathcal{N}(v) \cup v} \frac{x_u^k}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}})}$
      - $\mathcal{N}(v)$ is the neighborhood of node v in the graph
      - $\frac{1}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}}$ is a normalization factor
      - $\sum_{u \in \mathcal{N}(v) \cup v} \frac{x_u^k}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}})$ is an aggregation operator over $\mathcal{N}(v)$
      - w^k is a matrix of the appropriate dimensions

Graph Neural Networks

- Message passing operation has the following general form
  - Update v state: $\boxed{x_v^{k+1} = \sigma(w^k \sum_{u \in \mathcal{N}(v) \cup v} \frac{x_u^k}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}})}$
    - Normalization factor $\frac{1}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}}$ introduces a relative independence w.r.t. the nodes (v) degree compared to a basic aggregation rule s.a. $\sum_{u \in \mathcal{N}(v) \cup v} x_u^k$ where nodes with high degree would dominate
    - Many variants, this one was proposed in the graph convolutional network (GCN) by (Kipf et al 2017) – which remains popular GCN approach
    - Note: any aggregation operator must be permutation invariant, i.e. independent of the node order
    - Note: aggregagtion can also be performed on edges
    - Note: more on GNNs in Hamilton, 2020

Graph Neural Networks

- Attention with GCN
- A popular set aggregation rule relies on neighborhood attention
  - Attention was popularized with transformers in NLP
- Example
  - Aggregation rule $\sum_{u \in \mathcal{N}(v) \cup v} \alpha_{v,u} x_u^k$
  - With coefficients $\alpha_{v,u}$ denoting the attention on neighbor $u \in \mathcal{N}(v)$
    - $\alpha_{v,u} = \frac{\exp(a^T(Wx_v \oplus Wx_u))}{\sum_{u' \in \mathcal{N}(v)} \exp(a^T(Wx_v \oplus Wx_{u'}))}$, $\oplus$ concatenation operation
    - W a matrix of the appropriate size
    - And possible extensions to multiple head attention as in Transformers

NNs as surrogate models for solving PDEs
Discrete space models

✓ Regular grid: Learning from partial observations – ResNets - UNets

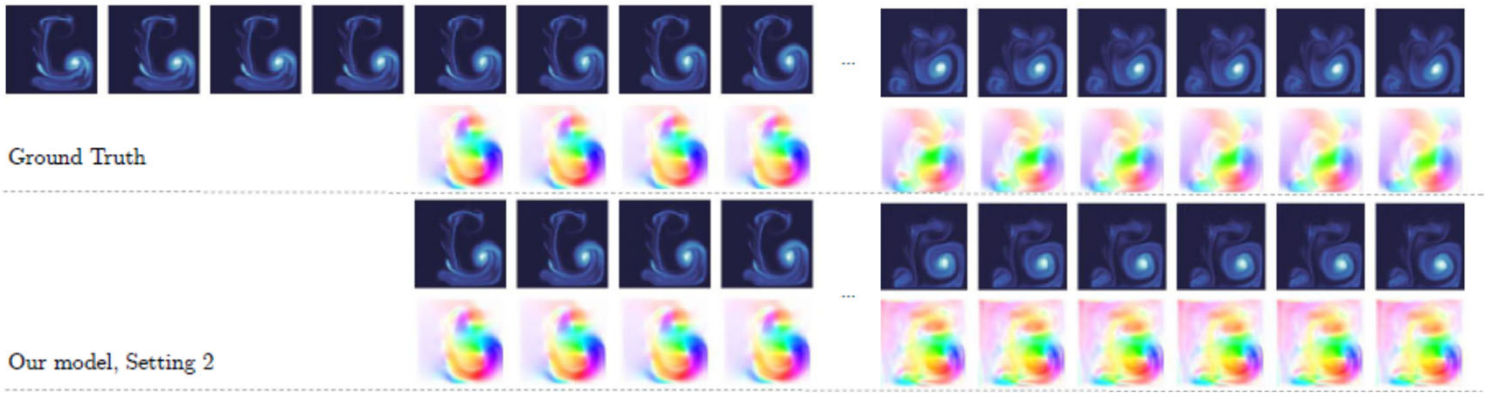
Irregular mesh: passing PDE solvers – Graph NNs

Learning from partial observations (Ayed et al. 2019- 2022)

- Forecasting non linear dynamical systems from observations only
  - Data-driven approach: without any knowledge of the physics
    - The form of the PDE is unknown
  - Only assumption
    - The underlying system follows a differential equation, but the PDE is unknown
- Objective
  - Learn the evolution of this system (observations and state) from scratch with a NN
  - Discover automatically the relation between states (dynamics)

Learning from partial observations (Ayed et al. 2019- 2022)

- Illustration
  - Navier Stokes equations
    - Discretised on a spatial 64x64 grid
    - State: fluid particle density + 2D velocity field
    - Observations: density only Y
    - Initial state: true full state X_0



Forecasting NS – horizon prediction $T = 30$ (in blue density Y, in color 2D velocity field with color code

Learning from partial observations (Ayed et al. 2019- 2022)

- Assume an underlying dynamical system with initial conditions
  - $$\begin{cases} X_0 & \text{Initial state of the system} \\ \frac{dX_t}{dt} = F^*(X_t) & \text{State dynamics} \\ Y_t = H(X_t) & \text{Observations} \end{cases}$$
- Variables
  - $X_t \in R^d$: state of the system at time t
    - function of time and space, partially observed
    - e.g. 3 D dynamics of the Ocean: velocity, pressure of the ocean
  - Y_t : observation, i.e. only available data for training $\{Y_t, 0 \leq t \leq T\}$
    - e.g. satellite observations: temperature, salinity, ocean color, waves height, …
  - H: measurement process linking state to observation is known
  - F^* describes the evolution of the state and is unknown

Learning from partial observations (Ayed et al. 2019- 2022)

- Objective
  - Learn the evolution of the system (observations and state) from scratch with a NN
- Learning problem
  - $\underset{F_\theta, g_\theta}{minimize}\ E_Y[\sum_{t=0}^{T}\left\|Y_t - H(\hat{X}_t)\right\|_2^2$ learn **trajectories** from **observations**
  - Subject to $\forall t, \frac{d\hat{X}_t}{dt} = F_\theta(\hat{X}_t)$, learn the **state** dynamics
  - $\hat{X}_0 = g_\theta(Y_{-k}, 0 < k \leq K)$ learn **initial state** from previous observations
- Implementation
  - F_θ is implemented as a ResNet, similar to forward Euler for ODEs
    - Solves $\frac{d\hat{X}_t}{dt} = F_\theta(\hat{X}_t)$
      - $X_{t+\delta t}^\theta = X_t^\theta + \delta t F_\theta(X_t^\theta)$
  - g_θ is a Unet or a ResNet

Learning from partial observations (Ayed et al. 2019- 2022) Examples

- NEMO – Nucleus for European Modelling of the Ocean Engine
  - State: 7 variables, we make use only of 2 variables corresponding to the velocity field
  - Observations: Sea Surface Temperature
  - Initial state: interpolated from previous observations
- For all test, data are partitioned into a training and a test set
  - Horizon of 6 time steps used for the target sequence for training

Learning from partial observations (Ayed et al. 2019- 2022)
NEMO – Global Ocean Physics Reanalysis

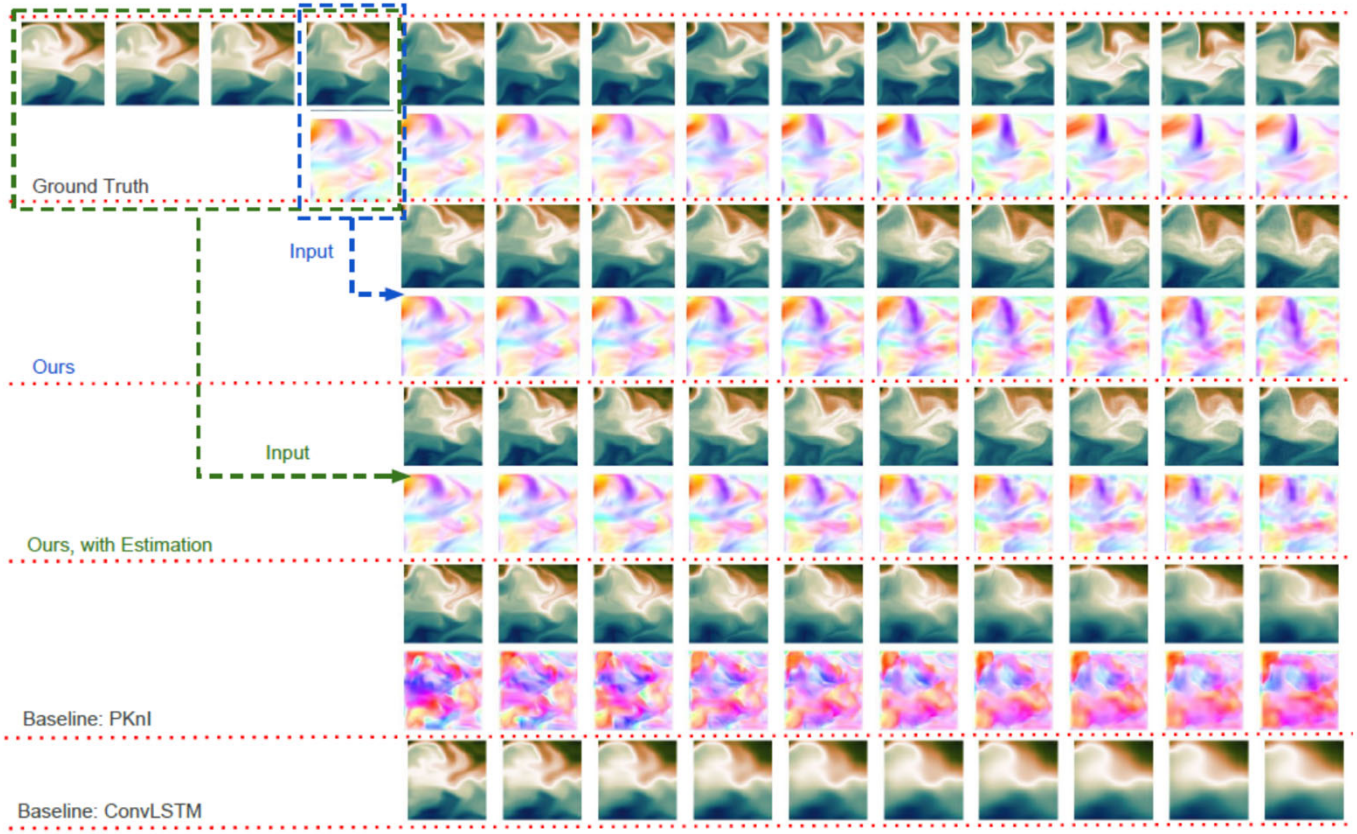


Figure 4. Forecasting Glorys2v4. From top to bottom: input and target observations, along with the associated ground truth partial hidden state, our model's outputs, our model variant when the initial conditions are estimated from the observations, outputs from the PKnI baseline, and from the ConvLSTM.

NNs as surrogate models for solving PDEs
Discrete space models

Regular grids: Learning from partial observations – ResNets – Unets

✓ Irregular meshes: Message passing PDE solvers – Graph NNs

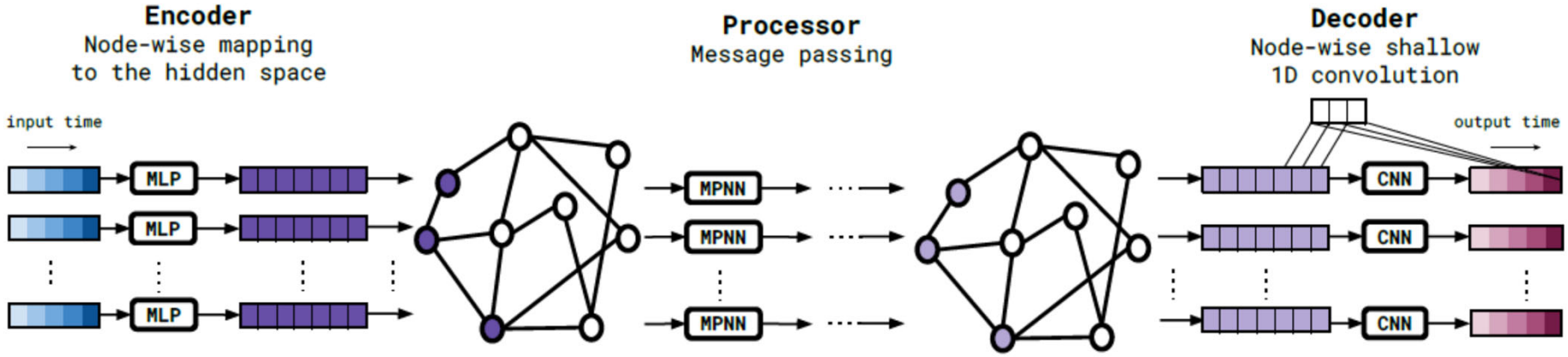
Graph Neural Networks

- GNN are well adapted to handle irregular meshes
  - Several efforts for developing PDE solvers based on graphs
    - (Sanchez-Gonzales et al. 2020, Belbute-Peres et al. 2020, Pfaff et al. 2021, …)
  - Grid cells/ nodes are mapped to a graph which is processed via message passing
- Example used in the course:
  - Brandstetter et al. 2022: Message Passing Neural PDE Solvers
    - Representative GNN solver
      - Handle multiple situations:
        - Multiple resolutions, boundary problems, parametric PDEs, etc
      - New improvements for training w.r.t. previous GNN PDE solvers
  - Inference
    - The mesh (precomputed) is mapped onto a graph
    - Objective: forecast spatio-temporal dynamics
    - Auto-regressive model $u(x,t) \rightarrow u(x,t+\Delta t) \rightarrow u(x,t+2\Delta t)$ ...

Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

Fig. Bransdtetter et al. 2022

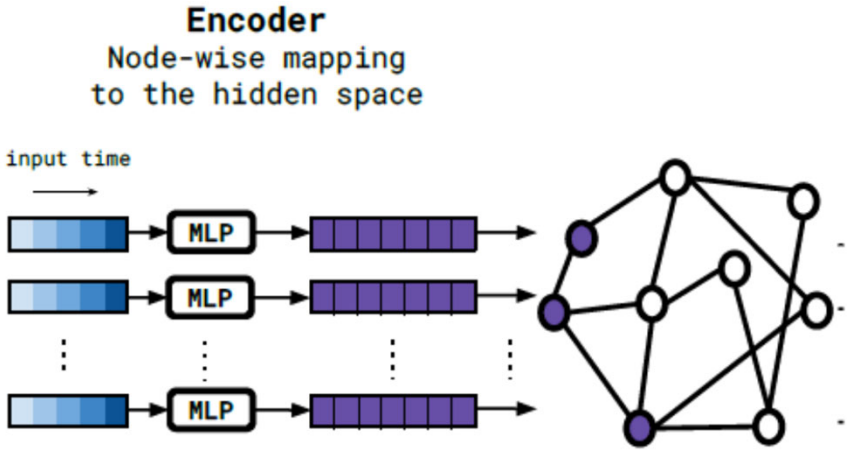
- General framework
  - Framework: Encode-Process-Decode (Sanchez-Gomzales 2020)
    - Process: message passing on the graph node embeddings



Node i, step k		
Encoding	**Process**	**Decode**
Input: last K values at each node i $f_i^0 = \text{Encode}(u_i^{k-K}, \dots, u_i^k)$	M message passing steps $f_i^m, m = 1 \dots M$	Output: next K values $u_i^{k+1}, \dots, u_i^{k+K}$

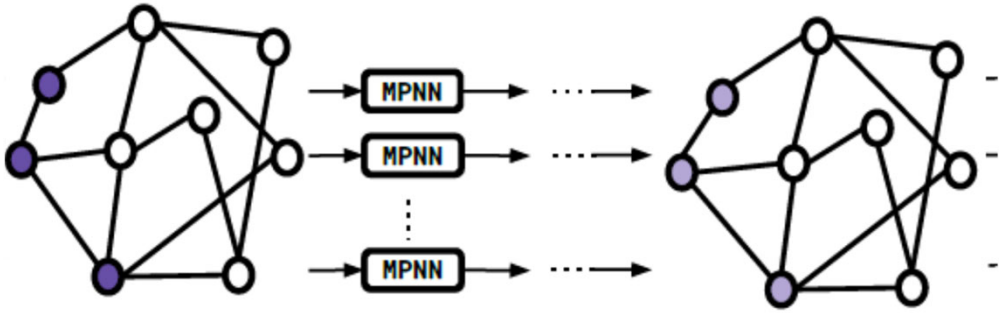
Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

- Encode
  - Compute node embeddings for each node i
  - $f_i^0 = embed\left(u_i^{k-K:k}, x_i, t_k, \theta_{PDE}\right)$ (vector)
    - $u_i^{k-K:k}$: K last values, x_i: node position, t_k: time step k
    - θ_{PDE}: parameters of the equation, e.g. PDE coefficient values, boundary condition indicators etc



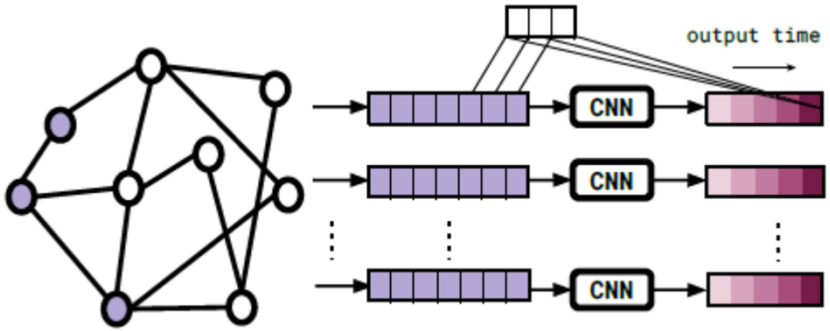
Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

- Process
  - Compute M steps of node update, $f_i^1, \ldots, f_i^M$ for all nodes i via message passing
  - Step m
    - Message for edge $j \to i$ $\quad m_{ij}^m = \Phi(f_i^m, f_j^m, u_i^{k-K:k} - u_j^{k-K:k}, x_i - x_j, \theta_{PDE})$
    - Update for node i $\quad f_i^{m+1} = \psi\left(f_i^m, \sum_{j \in \mathcal{N}(i)} m_{ij}^m, \theta_{PDE}\right)$
    - Φ, ψ are MLPs



Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

- Decode
  - From the last node embedding f_i^M , compute next K values $u_i^{k+1}, \ldots, u_i^{k+K}$ for all nodes i
  - f_i^M is a vector and is considered as a time contiguous signal and processed through a $1D$ CNN to compute the next K predictions $u_i^{k+1}, \ldots, u_i^{k+K}$



Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

- Claim
  - Able to handle
    - parametric PDEs, with the θ_{PDE} coefficients
    - Multiple resolutions, message passing allows for multiple resolutions in the GNN
    - Multiple boundary conditions (Dirichlet, Neuman, mixture)

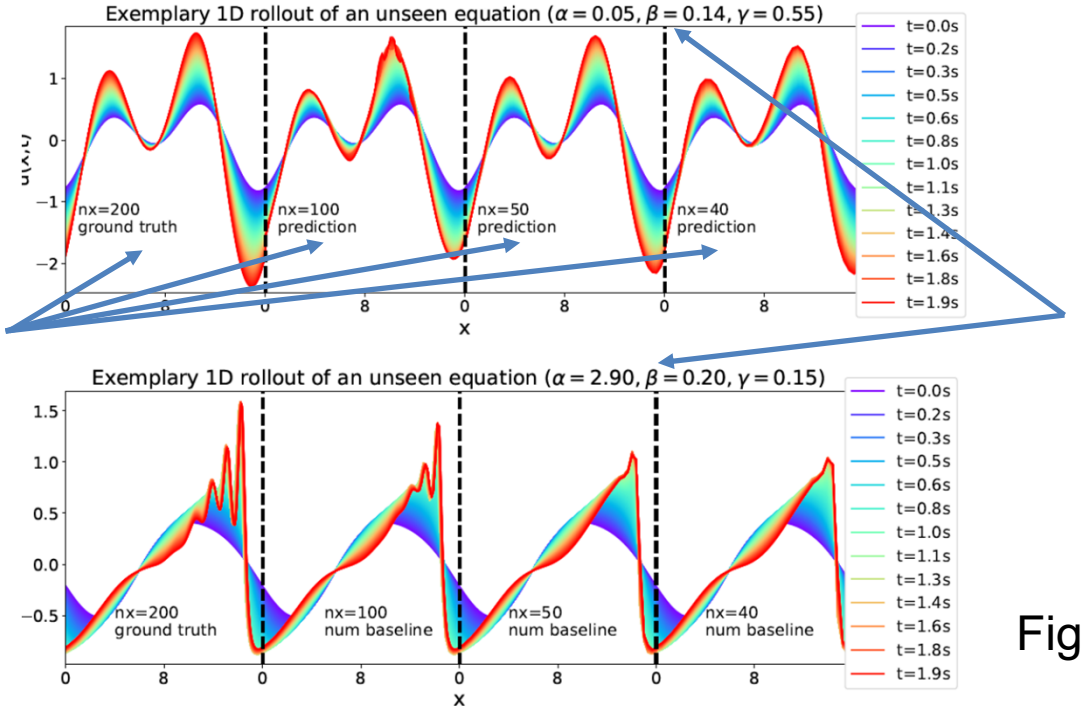
Message Passing Neural PDE Solvers (Brandstetter et al. 2022)

- Example
  - Generalization : family of PDE equations with different parameters and different resolutions
  - $\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\alpha u^2 - \beta \frac{\partial u}{\partial x} + \gamma \frac{\partial^2 u}{\partial x^2}\right) = f(x,t)$
  - $u(0,x) = u_0(x)$
  - Encompasses several classical equations
    - $(\alpha, \beta, \gamma) = (0, \eta, 0)$ Heat equation
    - $(\alpha, \beta, \gamma) = (0.5, \eta, 0)$ Burgers equation (simplified equation for fluid flows)
    - Etc
  - $\theta_{PDE} = (\alpha, \beta, \gamma)$

Message Passing Neural PDE Solvers (Brandstetter et al. 2022) Example

- Generalization : to PDE equations with different parameters values for (α, β, γ) and different resolutions
- Not real generalization but interpolation in the range of training values for (α, β, γ)
- Colours correspond to different times

resolutions



Parameter values (α, β, γ)

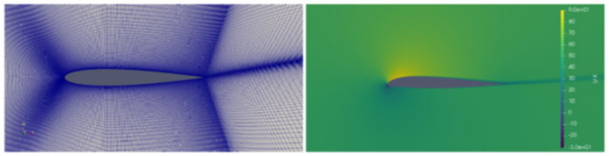
Fig. Brandstetter et al. 2022

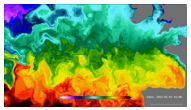
Modeling Spatio-temporal dynamics with Neural Networks

NNs as surrogate models for solving PDEs – Discrete space models
✓ NNs as surrogate models for solving PDEs – Continuous space models
NNs as surrogate models for solving PDEs – Data free approaches

Neural surrogate models: Neural operators

- Classical numerical solvers operate on grids or meshes (finite differences, finite elements, finite volumes)



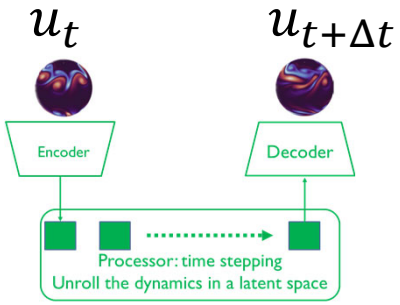


- Early neural solvers operate on tensors (grids) or on graphs (irregular meshes)

- Neural operators is a relatively recent topic aiming at learning maps between function spaces instead of vector spaces
  - e.g. images are considered as continuous functions
- Potential benefits
  - Functions and operators are mesh/ resolution invariant
  - Handle different geometries, multiple resolutions
  - Query at any space-time coordinate

Neural surrogate models; Neural operators implementations

- Encode-Process-Decode has become the standard framework for spatio-temporal forecasting problems



  - Encoding: maps physical inputs to a fixed-size small dimensional latent space
  - Processing: model the dynamics into this small latent space
  - Decoding: maps back to the physical space

- Implementations adapt recent developments and concepts from NLP, vision etc to the field lof physics
  - Attention/ NLP Transformers or Vision Transformers
  - Generative models: diffusion, flow matching etc

NNs as surrogate models for solving PDEs – Continuous space models

- ✓ Fourier Neural Operators

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- We consider
  - $\mathcal{V} = \mathcal{V}(\Omega \subset R^d; R^n), \mathcal{U} = U(\Omega' \subset R^{d'}; \mathrm{R^m})$ two function spaces
  - $\mathcal{G}: \mathcal{V} \to \mathcal{U}$ a non linear unknown mapping between the two function spaces
    - FNO considers mappings $\mathcal{G}$ that correspond to the solution operator of a parametric PDE
    - $v \in \mathcal{V}$ and $\mathrm{u} \in \mathcal{U}$ could correspond respectively to
      - an initial condition and a solution for a time dependent PDE
      - A parameter function and a solution for a time independent PDE
- Objective
  - Learn $\mathcal{G}_\theta$ an approximation of $\mathcal{G}$ from a finite set of samples
  - Samples are provided as p-points discretization of functions $v \in \mathcal{V}$ and $\mathrm{u} \in \mathcal{U}$
    - i.e. in practice we learn from discrete spaces, the representation of the continuous functions $v \in \mathcal{V}$ and $u \in \mathcal{U}$

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- FNO considers mappings $\mathcal{G}$ that correspond to the solution operator of a parametric PDE
  - $v \in \mathcal{V}$ and $u \in \mathcal{U}$ could correspond respectively to
    - An initial condition and a solution for a time dependent PDE
      - E.g. Advection-diffusion eq. (Sea Surface Temperature)

$u(t)$  $u(t + \Delta t)$

    - A parameter function and a solution for a time independent PDE
      - e.g. elliptic equation (Darcy Flow)
      - $-\nabla.(a(x)\nabla u(x) = f(x), x \in \Omega, u(x) = 0, x \in \partial\Omega, f$ piecewise constant

$a(x)$ diffusion coefficient  $u(x)$ steady state solution

Fig Li et al. 2022

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- Classical neural network

$$u = (K_T \circ \sigma_T \circ \cdots \circ \sigma_t \circ K_t \circ \cdots \circ \sigma_1 \circ K_0)v$$

  - With K_t a linear operator, σ_t a non linearity, u, v vectors
- Neural operators (simplified)
  - Follow a similar framework but u and v are no more vectors but functions

$$v_{t+1}(x) = \sigma_{t+1}\left(K_t(v_t)(x)\right)$$

  - With $K_t(v_t)$ an integral operator

$$K_t(v_t)(x) = \int_\Omega \varkappa_t(x, y)v_t(y)dy$$

  - $\varkappa_t(x, y)$is a kernel function
  - $v_t: \Omega \to R^n, v_{t+1}: \Omega \to R^m, \Omega \subset R^d$ a bounded space

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- How to learn the kernel function κ_t?
- We consider the simplified update rule

$$\boxed{u(x) = K(v)(x) = \int_{\Omega} \kappa(x, y)v(y)dy}$$

  - with $v, u: \Omega \rightarrow R^n$
- FNO works in Fourier space
  - Let us make $\kappa(x, y) = \kappa(x - y)$
    - $u(x) = \int_{\Omega} \kappa(x - y)v(y)dy$
    - $u(x) = \int_{-\infty}^{+\infty} \kappa(x - y)v'(y)dy$ with $v'(y) = \mathbb{1}_{\Omega}(y)v(y)$ <<<<<< convolution

$$\boxed{u(x) = (\kappa * v')(x)}$$

  - Convolution theorem:

$$\boxed{u(x) = \mathcal{F}^{-1}(\mathcal{F}(\kappa).\mathcal{F}(v'))(x)}$$

  - Convolution in space is equivalent to pointwise multiplication in Fourier domain
  - $\mathcal{F}(\kappa)$ is a linear transformation

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- Fourier transform – Linear Transform – Inverse Fourier

$$\boxed{u(x) = \mathcal{F}^{-1}(\mathcal{F}(\varkappa).\mathcal{F}(v'))(x)}$$

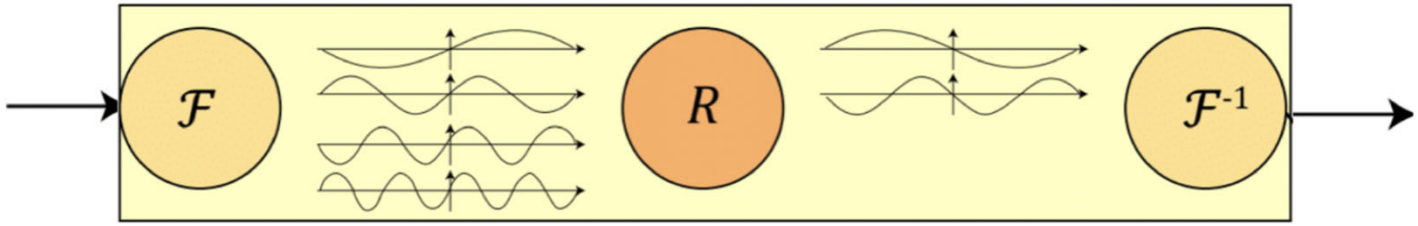


Fig Li et al., 2021

- Findings
  - In practice, it is sufficient to take the lower frequency modes
  - Fourier filters operate at the global level, different from CNN filters operating at a local level
  - R is a linear operator – implemented as a tensor

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- Whole module

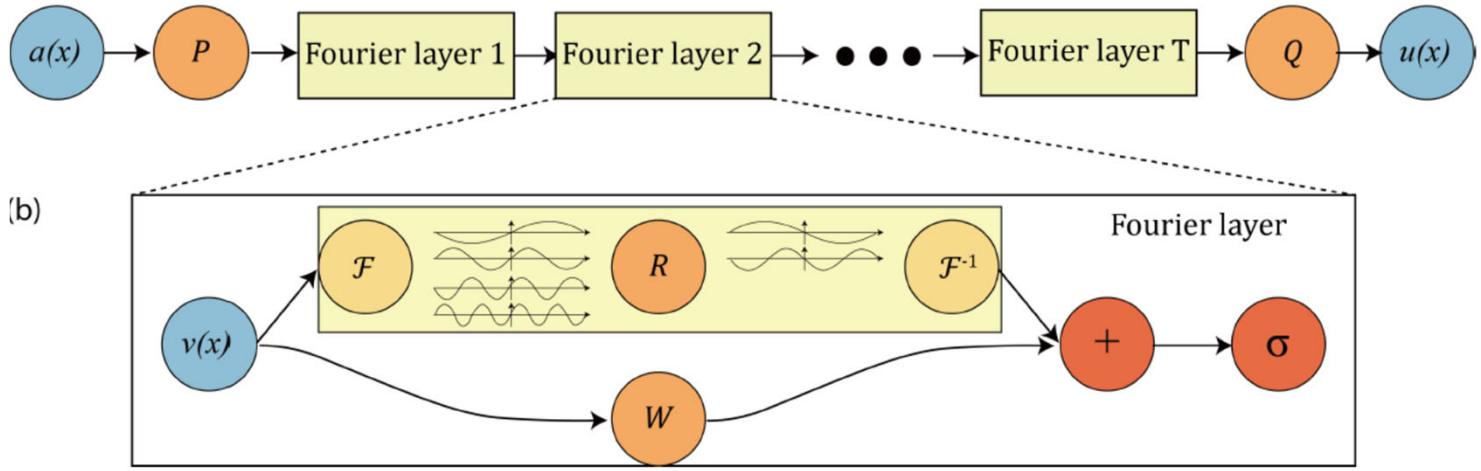


Fig Li et al. 2021

  - Fourier Layer works only with periodic conditions
  - W captures non periodic conditions
  - σ transforms are performed in the spatial domain
- In practice
  - $\mathcal{F}$ is implemented via a Fast Fourier Transform (complexity $nlogn, n$ nb of spatial points)
  - Operates on regular grids only
  - But FFT is independent of the grid size
    - Could be used on resolutions different from the training ones

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- Example: zero shot super-resolution
  - 2 D Navier Stokes, vorticity form, viscuous incompressible fluid
    - $\frac{\partial}{\partial t} w(x,t) + u(x,t).\nabla w(x,t) = \nu \Delta w(x,t) + f(x), x \in (0,1)^2, t \in (O,T]$
    - $\nabla . u(x,t) = 0, x \in (0,1)^2, t \in (0,T)$
    - $u(x,t)$ velocity field, $w(x,t)$ vorticity, characterizes local rotation of the fluid
  - Fig. Illustrates super-resolution: trained at 64x64, test on 256x256

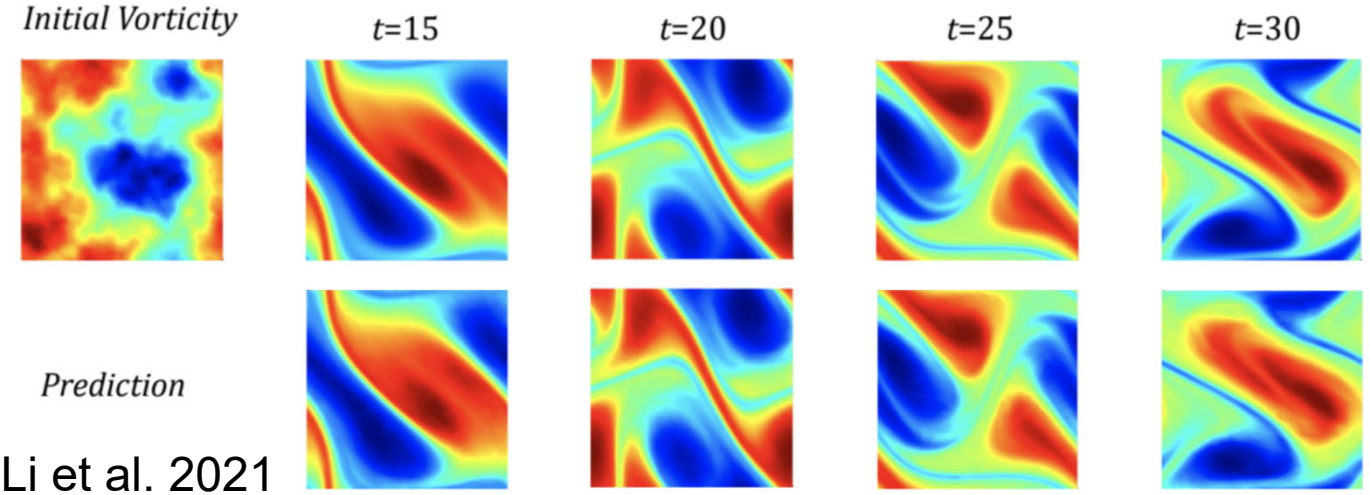


Fig Li et al. 2021

Zero-shot super-resolution: Navier-Stokes Equation with viscosity $\nu = 1e{-}4$; Ground truth on top and prediction on bottom; trained on $64 \times 64 \times 20$ dataset; evaluated on $256 \times 256 \times 80$ (see Section 5.4).

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- Details on N-S example
  - 2 D Navier Stokes, vorticity form, viscuous incompressible fluid
    - $\frac{\partial}{\partial t} w(x,t) + u(x,t).\nabla w(x,t) = \nu \Delta w(x,t) + f(x), x \in (0,1)^2, t \in (0,T)$
    - $\nabla . u(x,t) = 0, x \in (0,1)^2, t \in (0,T)$
    - $w(x,0) = w_0(x), x \in (0,1)^2$
      - $\nabla u = (\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y})$
      - $\Delta u = \nabla . \nabla \mathrm{u}$
      - $\nabla . \mathrm{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}$ for a 2 D vector v – divergence operator
      - $w = \nabla \times u$ with x the curl operator

NNs as surrogate models for solving PDEs – Continuous space models
Fourier Neural Operator (Li et al. 2021)

- Many extensions/ variants
  - Irregular grids, Physics informed FNO (Li et al. 2022) Transformer FNO, large size application to weather forecasting (Pathak 2022)
- Approximation theorem
  - Universal property approximation of operator classes by (F)NO e.g. Kowachki 2022

NNs as surrogate models for solving PDEs – Continuous space models

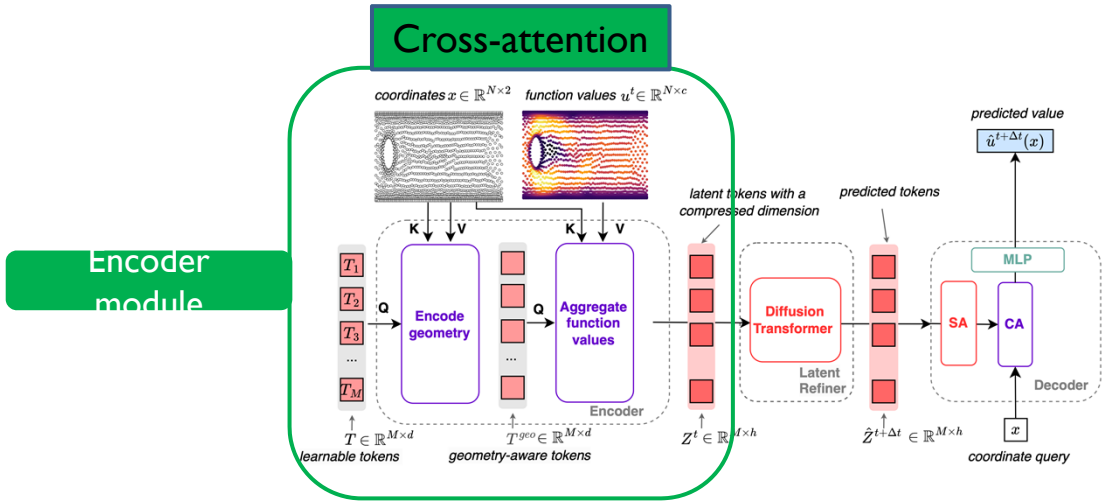
Learning grid free models

✓ AROMA: Preserving Spatial Structure for Latent PDE Modeling with Local Neural Fields, Serrano et al. Neurips 2024

Neural operators
AROMA: Attentive Reduced Order Model with Attention (Serrano et al. 2024)

- Principled Encode/ Process/ Decode framework
  - Properties
    - Handle diverse geometries: inputs and outputs may consist in point sets, grids, meshes, with different formats
    - Handle variable size inputs, e.g.. no fixed resolution, irregular samples
    - Can be queried at any spatial position within the spatial dynamics' domain

AROMA: Attentive Reduced Order Model with Attention (Serrano et al. 2024)
General framework



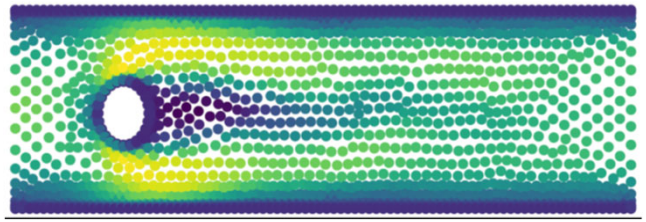
Cross-attention encoder: $u^t \rightarrow Z^t$

- Maps variable size discretized input u (point set, mesh, grid) into a fixed size & small dimensional sequence of latent embedding tokens Z
- Latent space encodes local spatial information on problem geometry + variable values

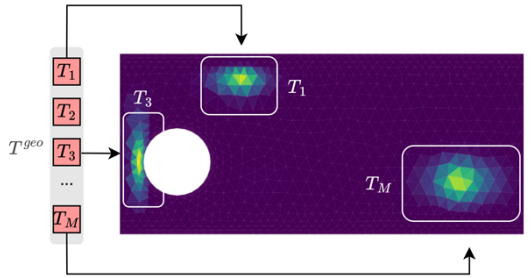
AROMA: Attentive Reduced Order Model with Attention
Cross-attention encoder captures spatial attention

Example: Navier Stokes – cylinder flow
Cross attention illustration

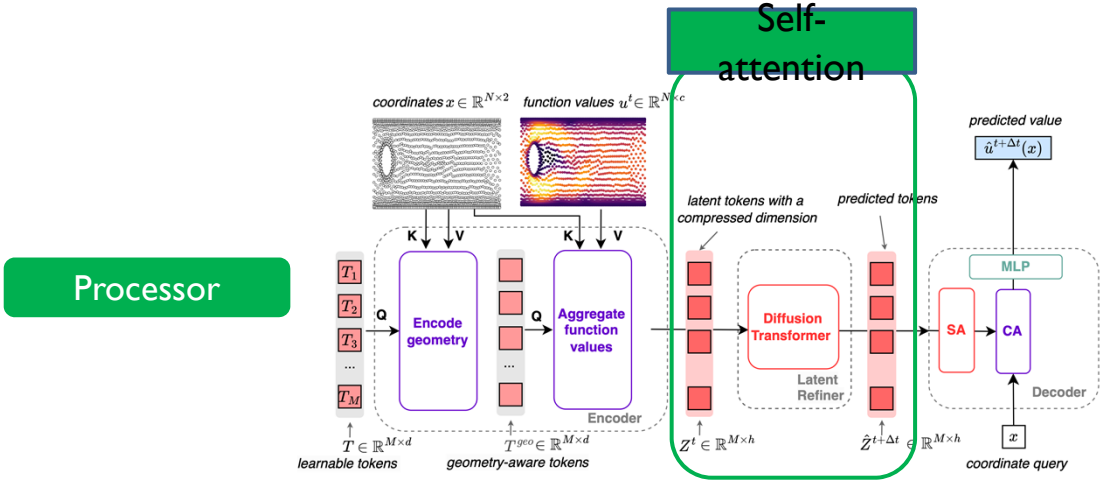
- Cylinder flow ground truth



- Tokens capture and encode local spatial information – cross attention between T^{geo} tokens and "x"



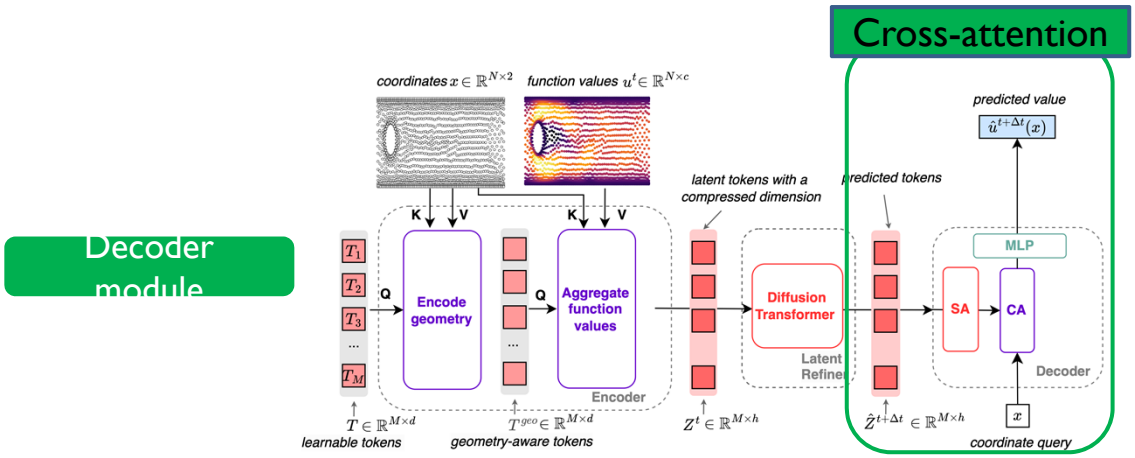
C3: Neural operators
AROMA: Attentive Reduced Order Model with Attention (Serrano et al. 2024)
General framework



Dynamics is modeled by a time stepping diffusion transformer: $Z^t \rightarrow Z^{t+\Delta t}$

- Learns the dynamics in the small dimensional latent space
- Self attention models relations between spatial latent tokens
- Inference: dynamics is enrolled in the latent space starting from an initial condition – low complexity
- Diffusion: introduces a stochastic component

AROMA: Attentive Reduced Order Model with Attention (Serrano et al. 2024)
General framework



Decoder: cross-attention neural fields decoder: $Z^{t+\Delta t} \rightarrow u^{t+\Delta t}$

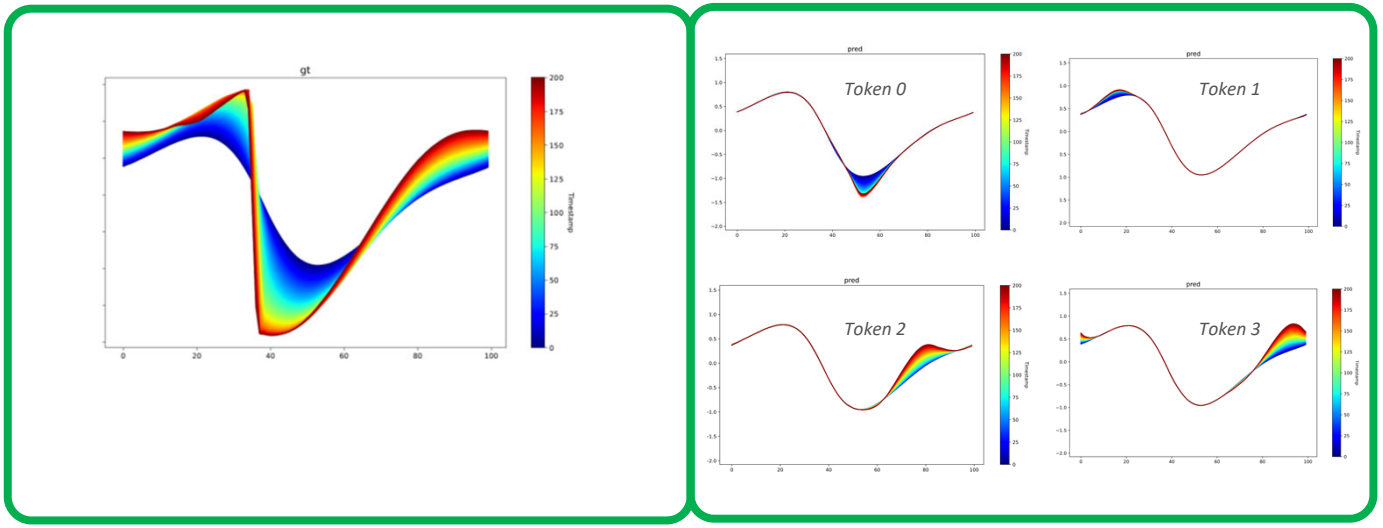
- Maps the latent representation forecast $Z^{t+\Delta t}$ to the original physical space
- Can be queried at any position x of the physical space

AROMA: Attentive Reduced Order Model with Attention (Serrano et al. 2024)
Cross-attention encoder captures spatial attention

- Burgers equation ground truth

Example: Burgers equation – perturbation analysis on the tokens

- Tokens encode local spatial information



AROMA: Attentive Reduced Order Model with Attention (Serrano et al. 2024)
Stability on long rollouts

Burgers equation
Trained to predict next step on 50 time steps trajectories
Unrolled for 200 steps

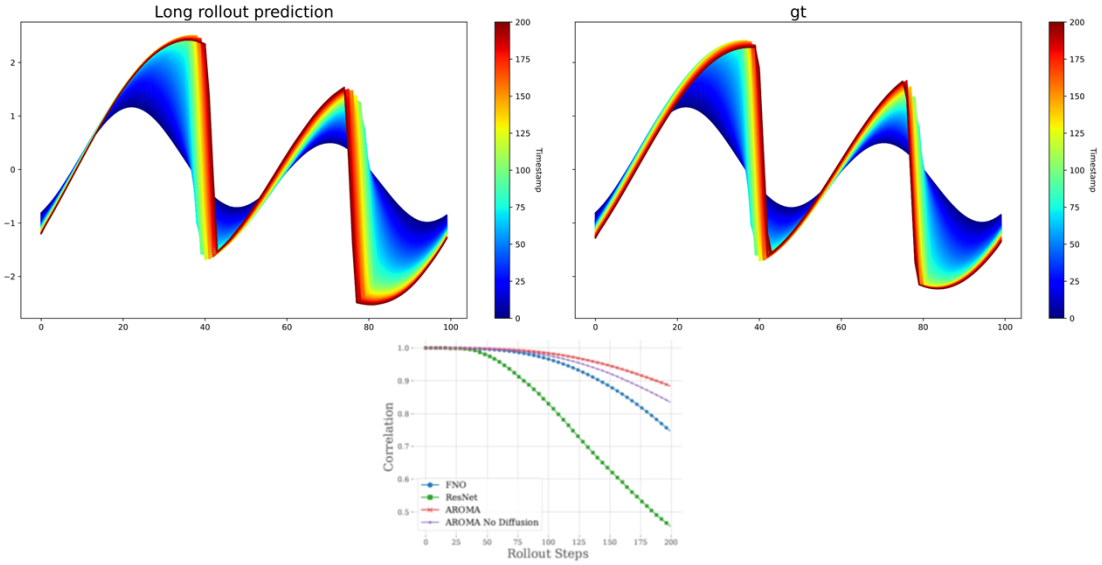


Figure 3: Correlation over time for long rollouts with different methods on *Burgers*

NNs as surrogate models for solving PDEs – Continuous space models

In-context generative pretraining – discrete transformers

✓ Zebra: In-context and generative pretraining for solving parametric pdes (Serrano et al. ICML 2025)

ZEBRA - In-context generative pretraining (Serrano et al. 2025)

https://arxiv.org/abs/2410.03437

- Inspired by In-context learning in NLP decoders (LLMs)

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```

Fig. Brown et al. 2020 (GPT3)

In-Context examples ($u_{n+1} = 3u_n - 1$)

Prompt

Sequence 1: 1, 2, 5, 14, …
Sequence 2: 2, 5, 14, 41, …
Sequence 3: 3, 8, 23, 68, …
Sequence 4: 4, ?

Query new initial condition

Answer

Sequence 4: 4, 11, 32, 95, etc..

Without optimization LLMs are somehow capable of inferring the correct pattern in the sequences, i.e the underlying parameters: $u_{n+1} = 3u_n - 1$

Fig. Serrano 2025

No gradient update – only context

ZEBRA - In-context generative pretraining (Serrano et al. 2025)

How does in-context learning works: Two main interpretations

Gradient update interpretation

Dai et al. ACL 2023. Why can, GPT learn in context ?
Transformer attention has a dual interpretation as gradient descent in the linear attention case.

Interpret LLM as meta-optimizers that perform implicit fine tuning for in-context examples.

Bayesian interpretation

Xie et al. https://ai.stanford.edu/blog/understanding-incontext/

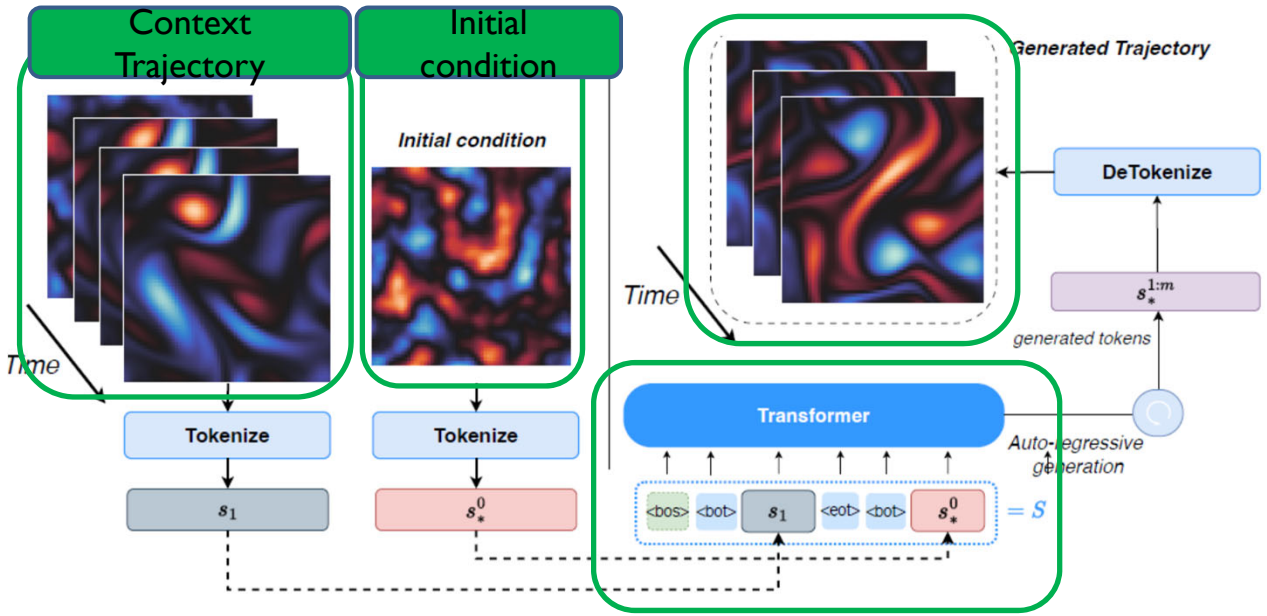
Pretraining learns latent concept distributions, inference identifies the prompt latent concept

$$p(\text{output}|\text{prompt}) = \int_{\text{concept}} p(\text{output}|\text{concept}, \text{prompt}) p(\text{concept}|\text{prompt}) d(\text{concept})$$

ZEBRA - In-context generative pretraining (Serrano et al. 2025)

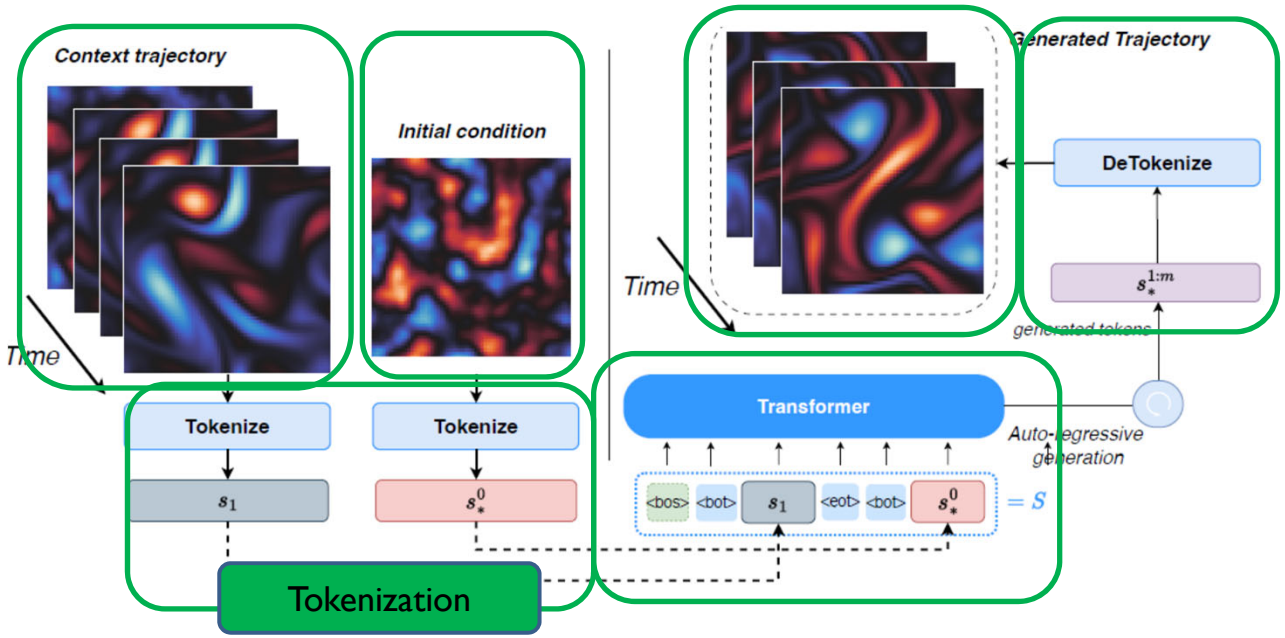
https://arxiv.org/abs/2410.03437

Inference



Context trajectory: trajectory from the same PDEs starting from another Initial Condition

ZEBRA - In-context generative pretraining (Serrano et al. 2025)
Inference: **discrete model** i.e. LLM structure

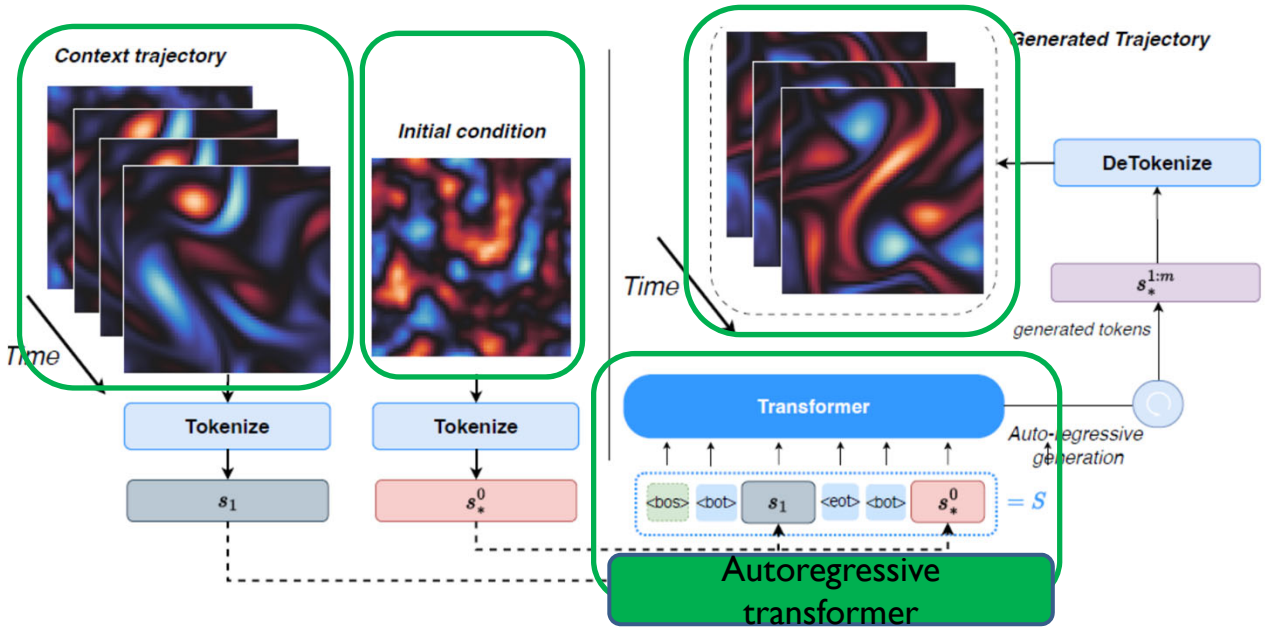


Context trajectory and IC are compressed into sequences of discrete tokens (similar to word encoding)

ZEBRA - In-context generative pretraining (Serrano et al. 2025)

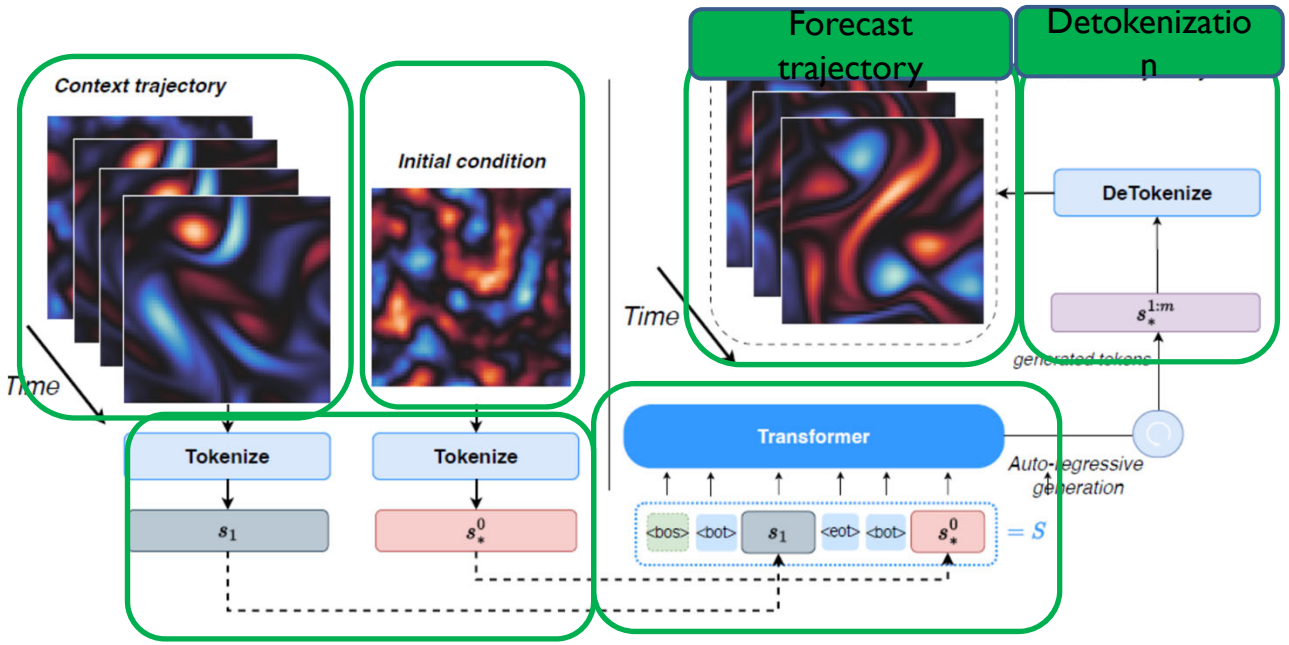
https://arxiv.org/abs/2410.03437

Inference



The transformer operates on the discrete tokenized representations of the data to predict next tokens sequence autoregressively – similar to LLMs

ZEBRA - In-context generative pretraining (Serrano et al. 2025)
Inference: **discrete model** i.e. LLM structure

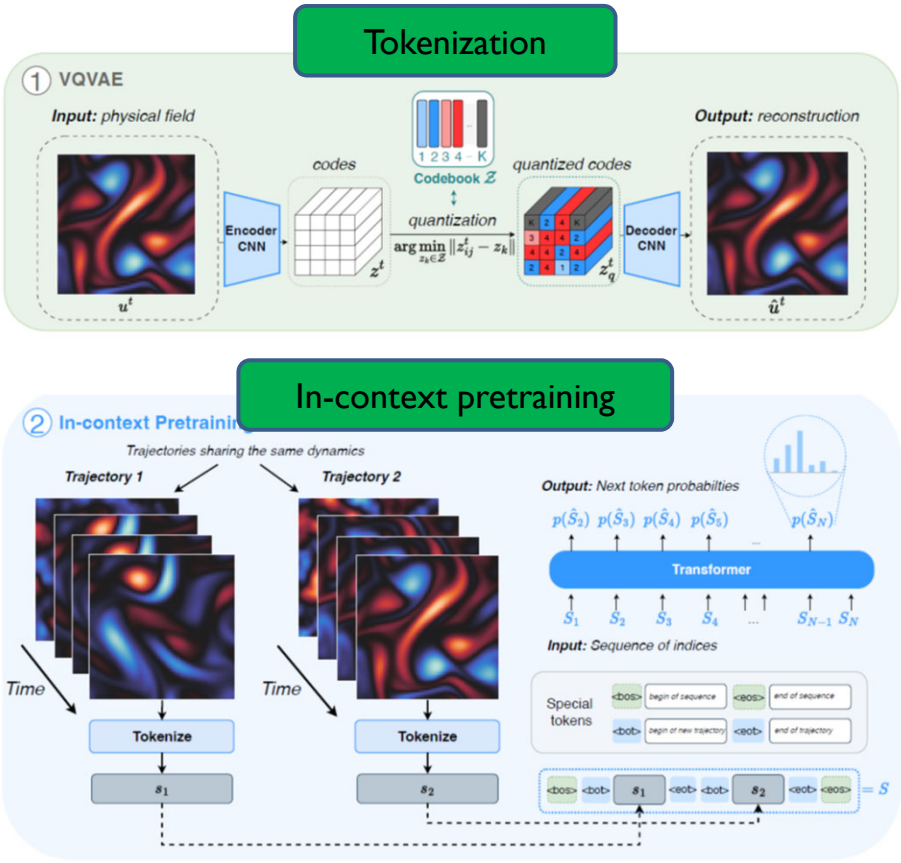


The predicted token sequence is mapped back to the physical space

ZEBRA - In-context generative pretraining (Serrano et al. 2025)
Training: 2 steps

Encode-decode:VQVAE
Maps a frame to a finite set of tokens

Processor: LLM
Predicts discrete distribution of **tokens**
Training loss: cross entropy



ZEBRA - In-context generative pretraining (Serrano et al. 2025)

https://arxiv.org/abs/2410.03437

Examples

Heat equation

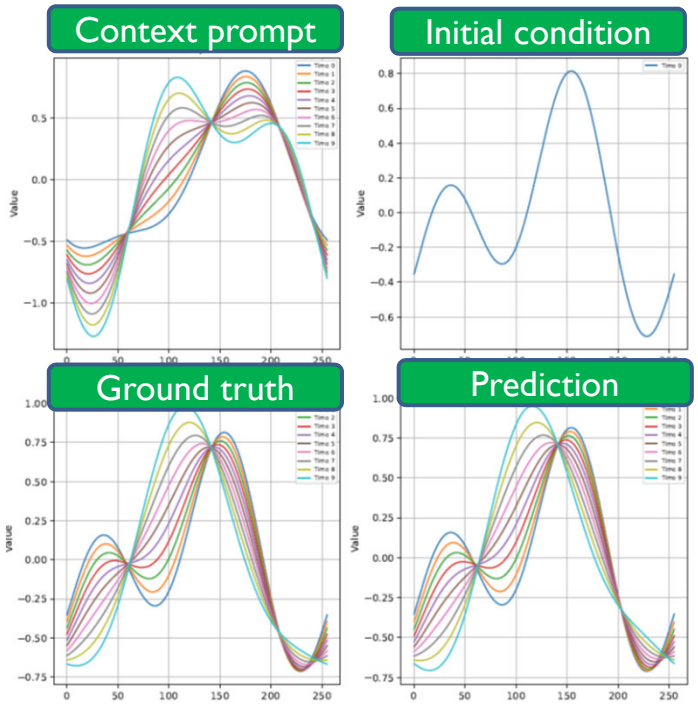


Figure 26: **One-shot** adaptation on Heat

Combined equation

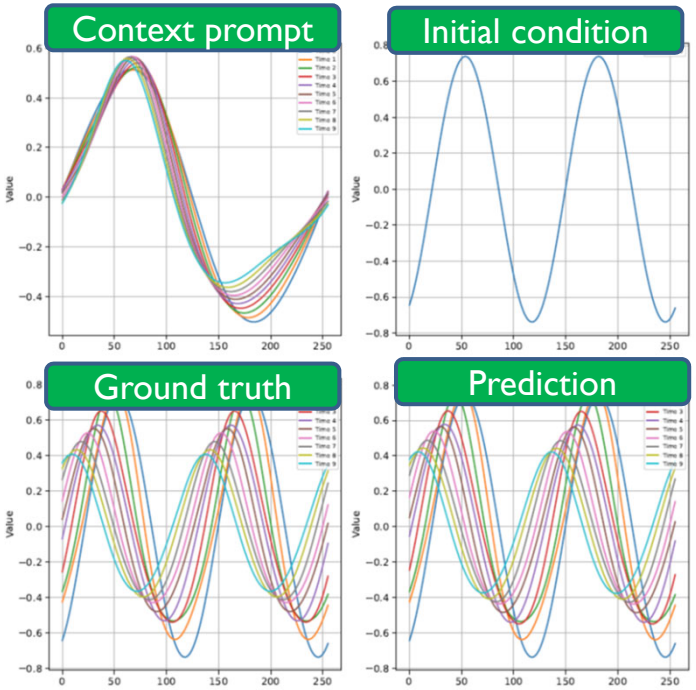


Figure 30: **One-shot** adaptation on Combined

ZEBRA - In-context generative pretraining (Serrano et al. 2025)

https://arxiv.org/abs/2410.03437

Examples

Vorticity

Context prompt

Ground truth

Prediction

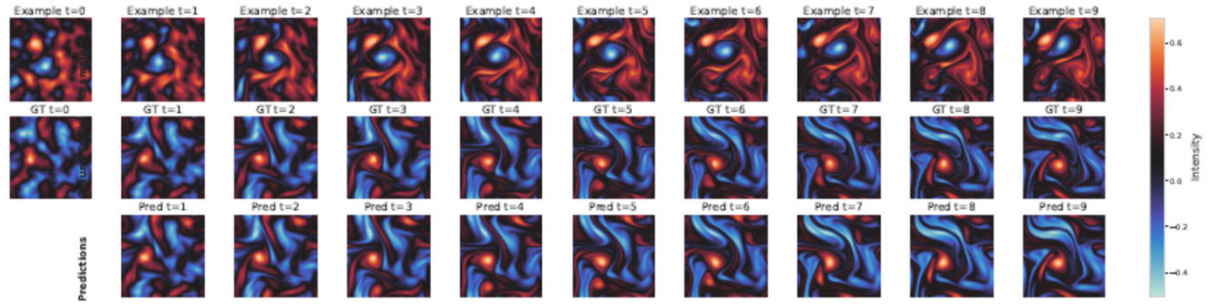


Figure 35: **One-shot OoD** adaptation on Vorticity. Example 1.

NNs as surrogate models for solving PDEs – Continuous space models

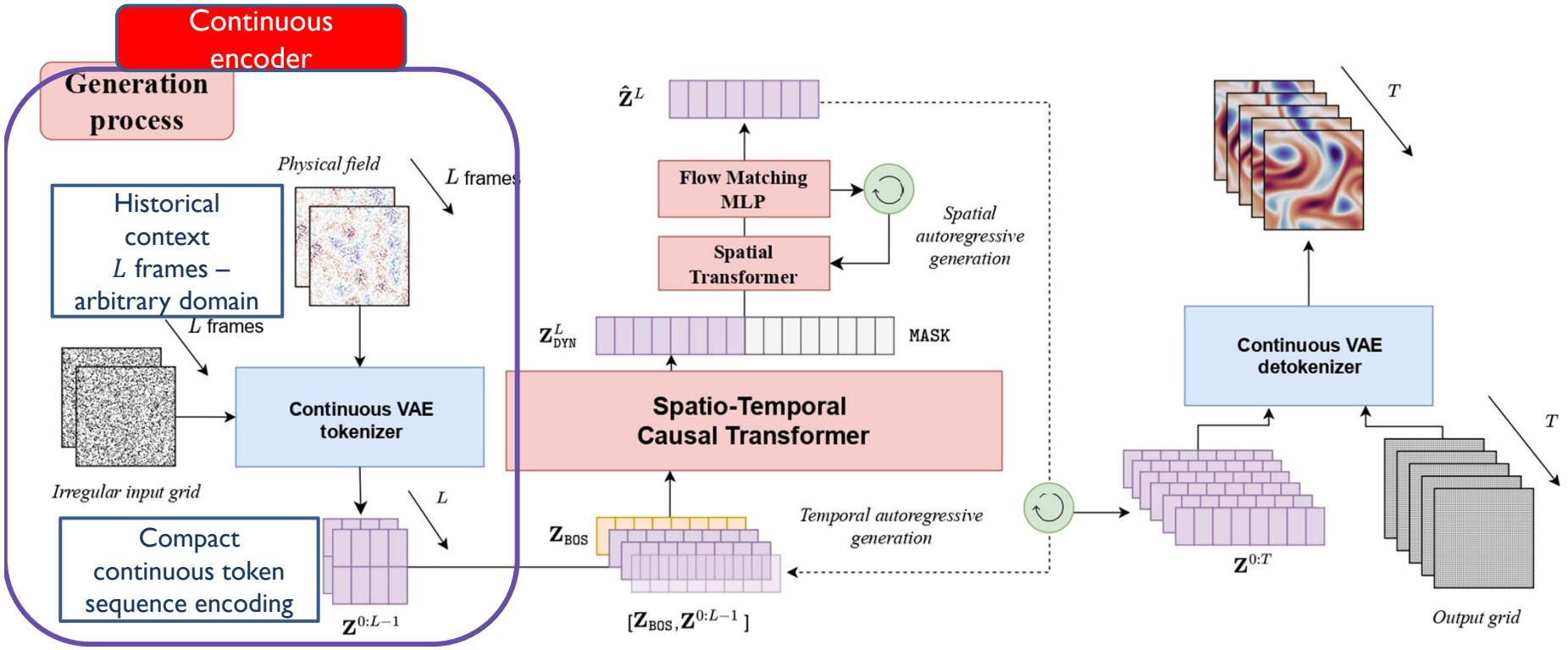
In-context generative pretraining: continuous transformers

✓ ENMA: Tokenwise Autoregression for Continuous Neural PDE Operators (Kassai et al. Neurips 2025)

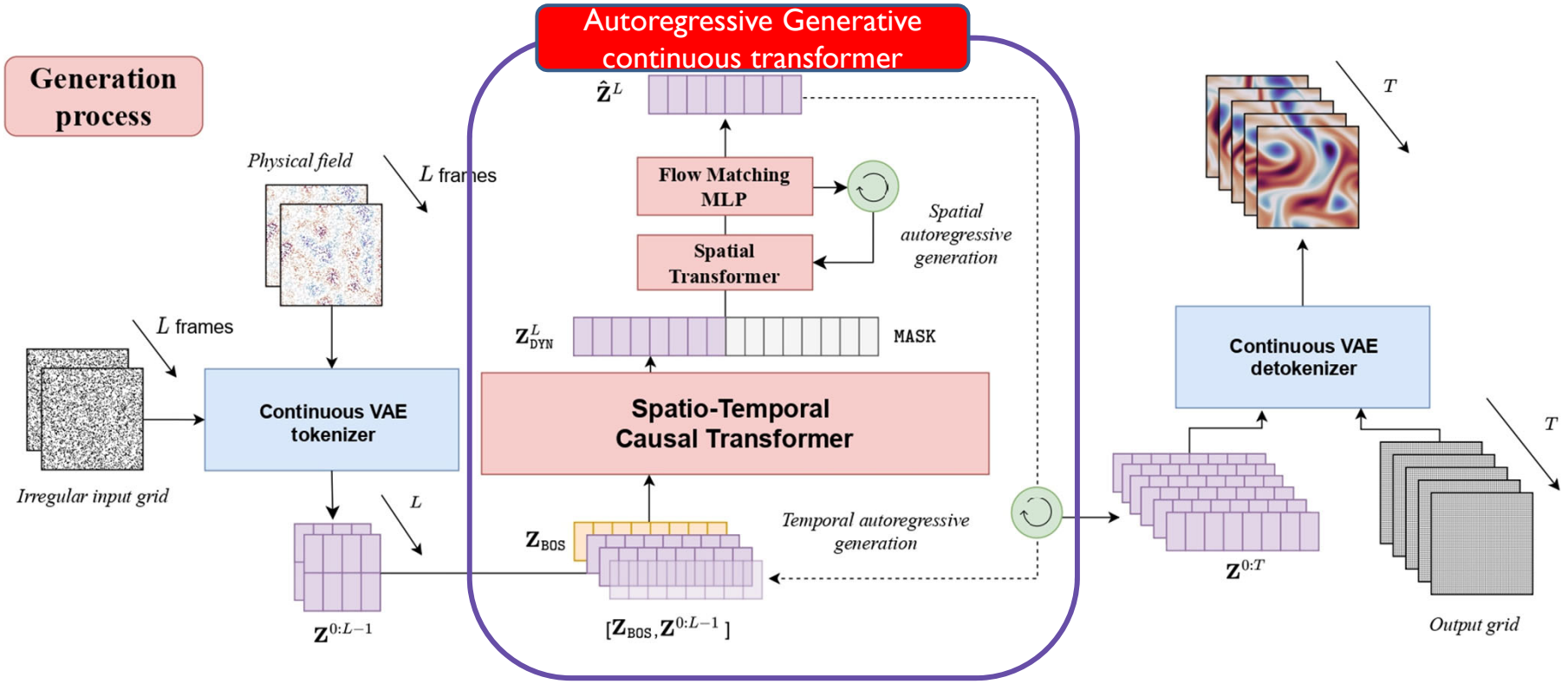
ENMA: Tokenwise Autoregression for Generative Neural PDE Operators (Kassai et al. 2025)

- Follows an encode-process-decide framework
- Extends the in context learning ideas of ZEBRA to a fully continuous token space instead of a discrete space
  - More natural for physical phenomena
  - No more token quantization with the corresponding information loss
  - Trained as a generative model using a flow matching loss
  - Properties
    - Handle irregularly positioned and possibly faulty sensors
    - Allows performing in context learning and well as forecasting from past history
    - Adapt to changes in the discretisation grid
    - Models uncertainty

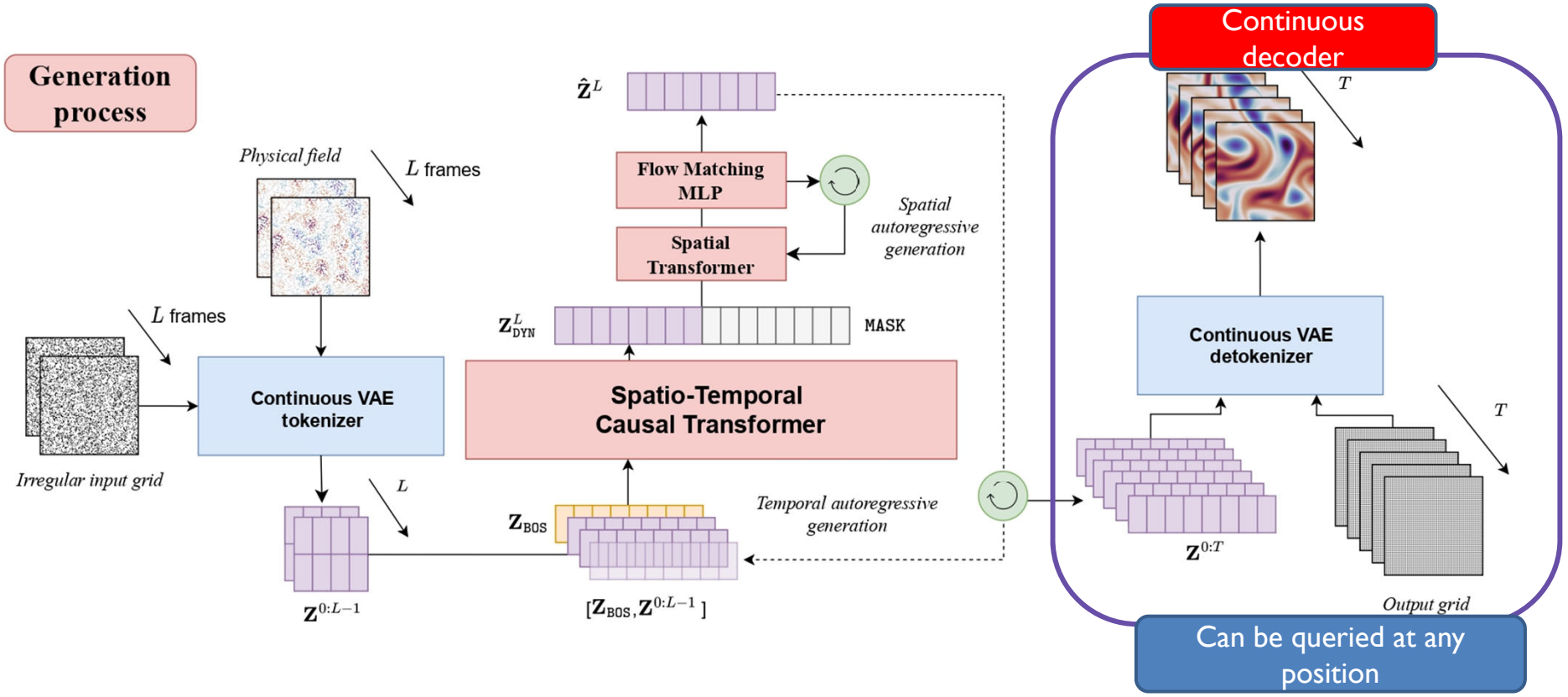
ENMA: Tokenwise Autoregression for Generative Neural PDE Operators (Kassai et al. 2025)



ENMA: Tokenwise Autoregression for Generative Neural PDE Operators (Kassai et al. 2025)



ENMA: Tokenwise Autoregression for Generative Neural PDE Operators (Kassai et al. 2025)



ENMA: Tokenwise Autoregression for Generative Neural PDE Operators (Kassai et al. 2025)

Examples: Gray Scott

In-distribution

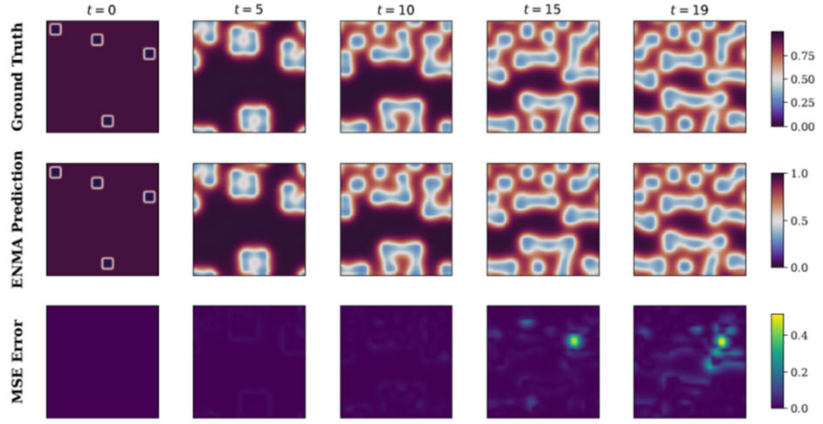


Figure 37: Qualitative comparison between ENMA prediction and ground truth for an in-distribution sample from the Gray-Scott dataset ($F = 0.0323$, $k = 0.0606$).

Out-of-distribution

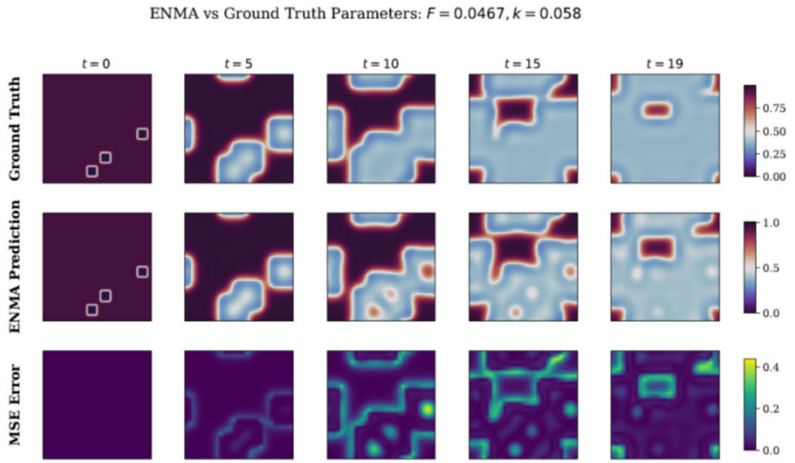


Figure 39: Out-of-distribution (OOD) generalization for the Gray-Scott equation. ENMA prediction remains consistent despite being evaluated at unseen parameters ($F = 0.0467$, $k = 0.058$).

ENMA: Tokenwise Autoregression for Generative Neural PDE Operators (Kassai et al. 2025)

Examples: vorticity at different viscosities

In-distribution

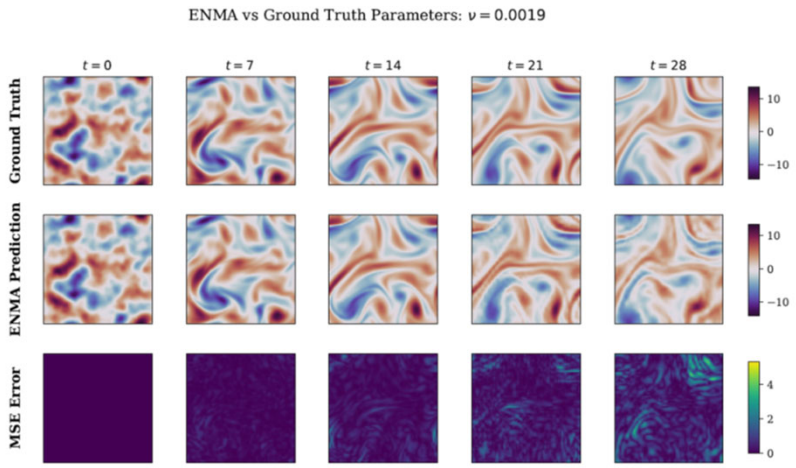


Figure 43: Qualitative comparison between ENMA prediction and ground truth for an in-distribution sample from the Vorticity dataset ($\nu = 0.0019$).

Out-of-distribution

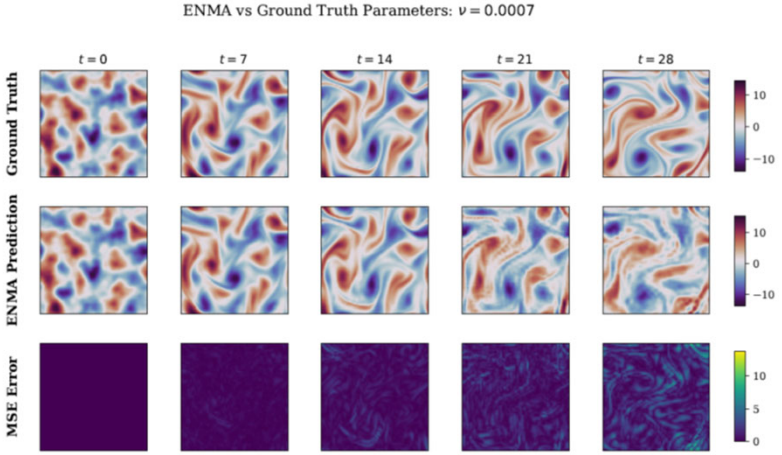


Figure 45: Out-of-distribution example from the Vorticity dataset ($\nu = 0.0007$), highlighting ENMA's robustness in extrapolating vortex dynamics.

Modeling Spatio-temporal dynamics with Neural Networks

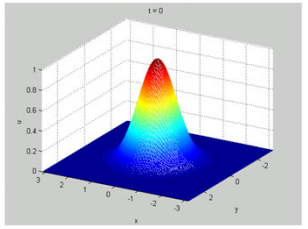
NNs as surrogate models for solving PDEs – Discrete space models
NNs as surrogate models for solving PDEs – Continuous space models
✓ NNs as surrogate models for solving PDEs – Data free approaches

NNs as surrogate models for solving PDEs – Data free approaches (Lagaris 1998, Sirignano 2018, Raissi 2019)

- Objective
  - Build a reduced order (parametric) model, implemented by a NN, to offer a cheap approximate solution of a PDE
  - Assumption: the form of the PDE is known as for classical solvers
  - Data free approach: no need for simulated data/ observations as required by all the other approaches seen before
- Results
  - The algorithm solves the PDE using a single parametric function, for all space and time conditions
  - The original algorithm solves a unique IBVP – and shall be re-trained for a new IBVP

NNs as surrogate models for solving PDEs – Data free approaches (Lagaris 1998 , Sirignano 2018, Raissi 2019)

- Problem
  - Parabolic PDE with d spatial dimensions
  - $$\begin{cases} \frac{\partial u(t,x)}{\partial t} + \mathcal{L}u(t,x) = 0, (t,x) \in [0,T] \times \Omega, \Omega \subset R^d & \text{PDE} \\ u(t=0,x) = u_0(x), x \in \Omega & \text{Initial conditions} \\ u(t,x) = g(t,x), x \in \partial\Omega & \text{Boundary conditions} \end{cases}$$
  - $u(t,x)$ is the spatio-temporal quantity of interest
  - $\mathcal{L}u(t,x)$ is the differential term of the PDE
    - e.g. Burgers $1D$: $\frac{\partial u(t,x)}{\partial t} = -\mathrm{u}\frac{\partial u(t,x)}{\partial x} + \nu\frac{\partial^2 u(t,x)}{\partial x^2}$



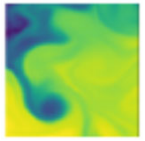
- Objective
  - Approximate $u(t,x)$ with a NN $f(t,x;\theta), \theta \in R^K$ are the network parameters

NNs as surrogate models for solving PDEs – Data free approaches (Lagaris 1998 , Sirignano 2018, Raissi 2019)

- Formulate the problem as minimizing an objective function
  - Foe simplification we only consider initial conditions (remove BC)

Initial condition loss | PDE loss: constraint

- $J(f) = \|f(0,x;\theta) - u_0(x)\|^2_{\Omega,\nu_1} + \left\|\frac{\partial f(t,x;\theta)}{\partial t} - \mathcal{L}f(t,x;\theta)\right\|^2_{[0,T]\times\Omega,\nu_2}$

$u_0()$: 

- Learn $f(0,x;\theta)$ by sampling from Ω, the initial condition
- This is a regression problem
- This provides a parametric approximation of target $u(x,t=0)$

- Constrains $f(t,x;\theta)$ to follow the PDE expression by sampling uniformly from $[0,T]\times\Omega$
- $\frac{\partial f}{\partial t}$ and $\mathcal{L}u(t,x)$ computed by automatic differentiation

  - Solved using stochastic gradient descent
  - Several extensions, e.g. sampling from data from the PDE loss, etc

NNs as surrogate models for solving PDEs – Data free approaches Sirignano 2018, Raissi 2019

- Algorithm
  - Iterate
    - Sample (t_n, x_n) from $[0, T] \times \Omega, \nu_2$; sample the initial condition point z_n from Ω, ν_1
    - Calculate the squared error $G(\theta_n, s_n)$ at the sampled points $s_n = \{(t_n, x_n), (\tau_n, y_n), z_n\}$ with:
      - $G(\theta_n, s_n) = (\frac{\partial f(t_n, x_n; \theta_n)}{\partial t} - \mathcal{L} f(t_n, x_n; \theta_n))^2 + \left(f(0, z_n; \theta_n) - u_0(z_n)\right)^2$
    - Take a gradient step
      - $\theta_{n+1} = \theta_n - \epsilon_n \nabla_\theta G(\theta_n, s_n)$

NNs as surrogate models for solving PDEs – Data free approaches Sirignano 2018, Raissi 2019

- Comments
  - Mesh free approach, similar in that to INR
  - Several extensions (Karniadakis et al. 2021)
  - Popularized the idea of approximatin a differential equation via a parametric function
  - Still much slower than classical solvers
  - Requires learning a solver for each specific equation/ initial & boundary conditions
    - More on that later
  - No extrapolation in time

NNs as surrogate models for solving PDEs – Data free approaches (Sirignano 2018, Raissi 2019)

- Example: Burger equation

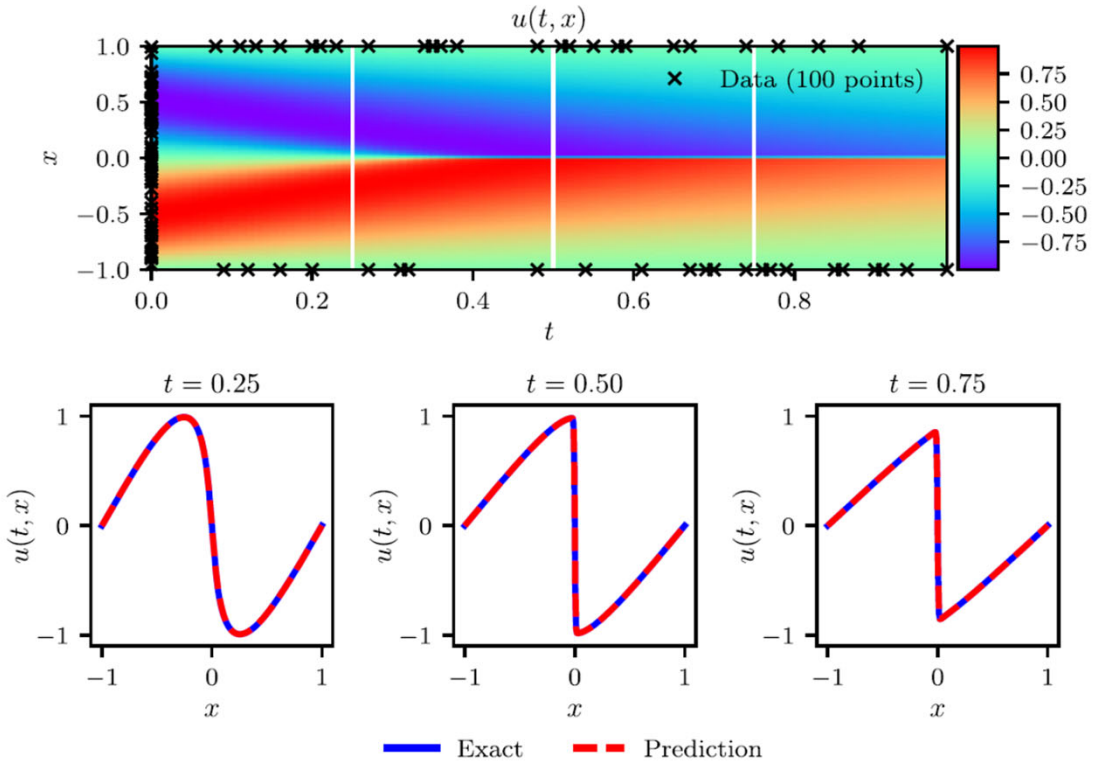


Fig. A.6. *Burgers' equation: Top:* Predicted solution $u(t,x)$ along with the initial and boundary training data. In addition we are using 10,000 collocation points generated using a Latin Hypercube Sampling strategy. *Bottom:* Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the white vertical lines in the top panel. The relative $\mathbb{L}_2$ error for this case is $6.7 \cdot 10^{-4}$. Model training took approximately 60 seconds on a single NVIDIA Titan X GPU card.

Fig: Raissy 2019