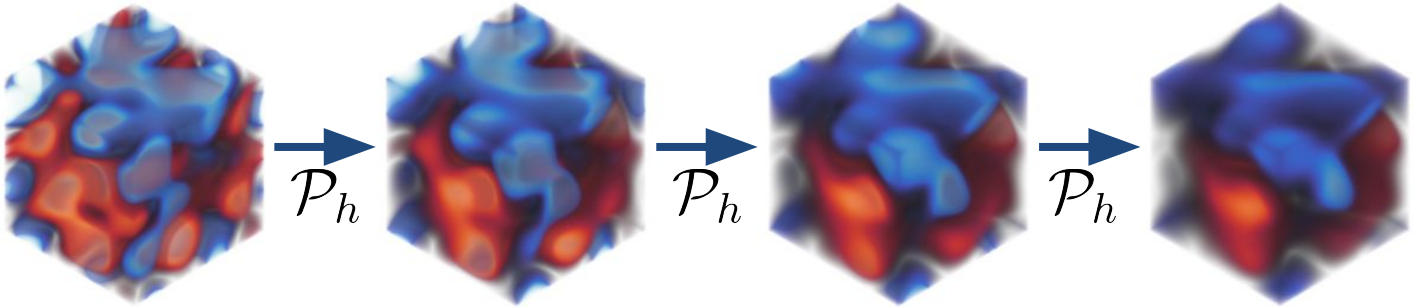


On Autoregressive Neural Emulators for PDEs

13 May 2025 @ IRMA Strasbourg, PDE Team
Felix Koehler
(PhD student with Prof. Thuerey at TUM)

Kuramoto-Sivashinsky 2D

Kolmogorov Flow 2D

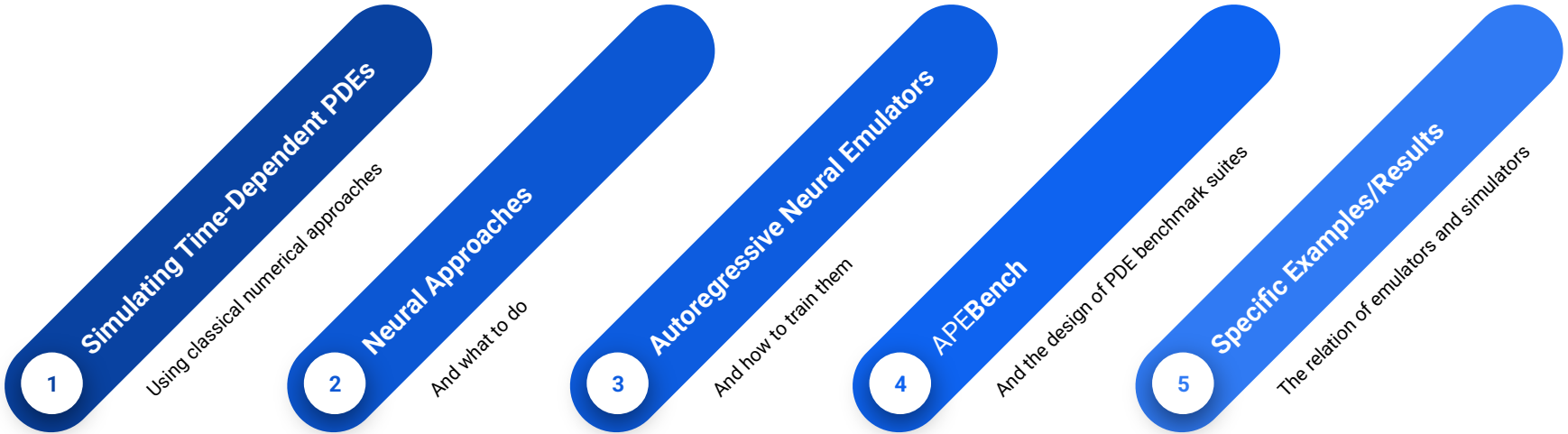
Gray-Scott 2D

Kuramoto-Sivashinsky 3D

Swift-Hohenberg 3D

Gray-Scott 3D

Agenda


1
Simulating Time-Dependent PDEs
Using classical numerical approaches
2
Neural Approaches
And what to do
3
Autoregressive Neural Emulators
And how to train them
4
APEBench
And the design of PDE benchmark suites
5
Specific Examples/Results
The relation of emulators and simulators

Simulating Spatio-Temporal Phenomena

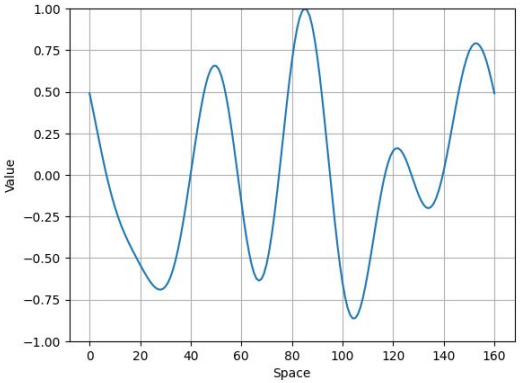
$$\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x}$$

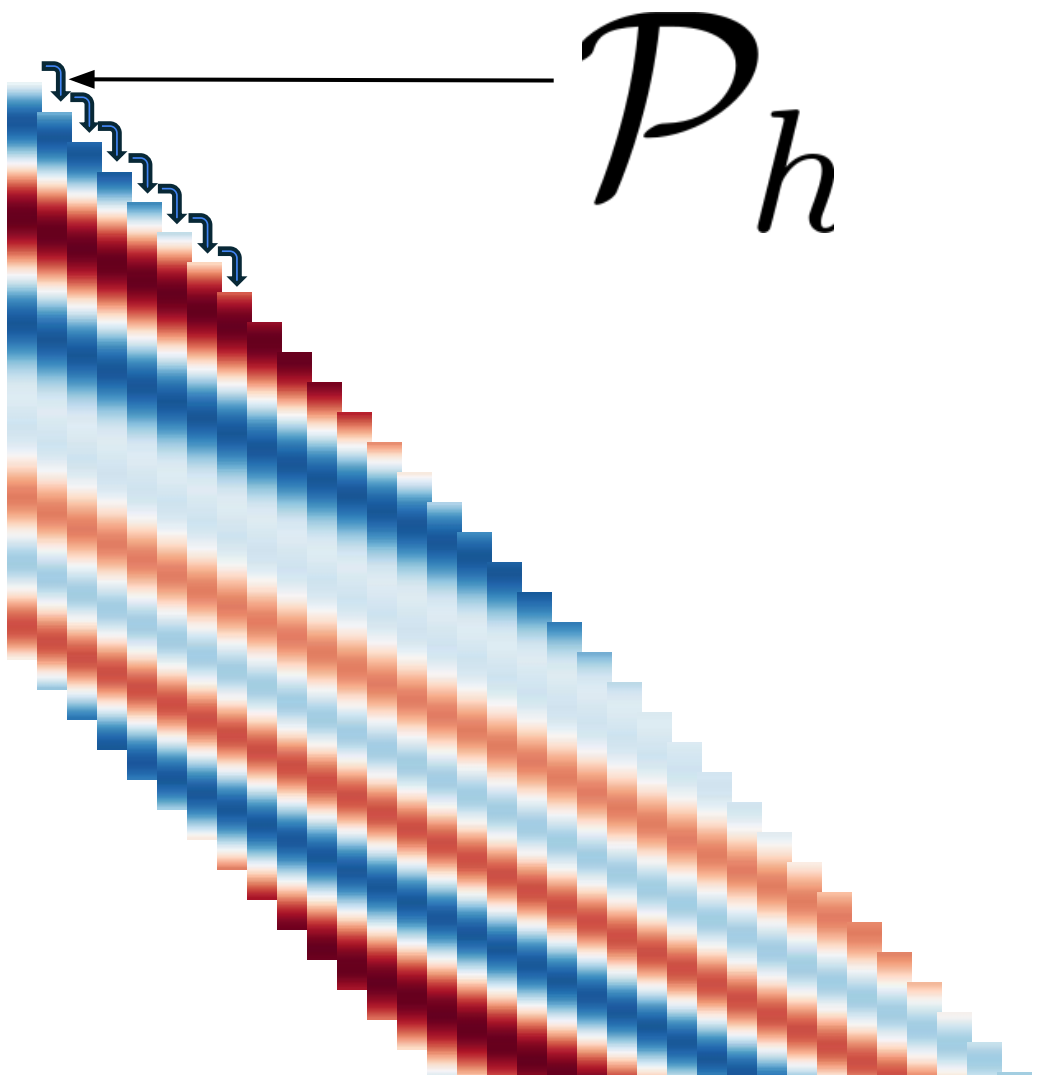
Symbolic Manipulation
+ Implementation

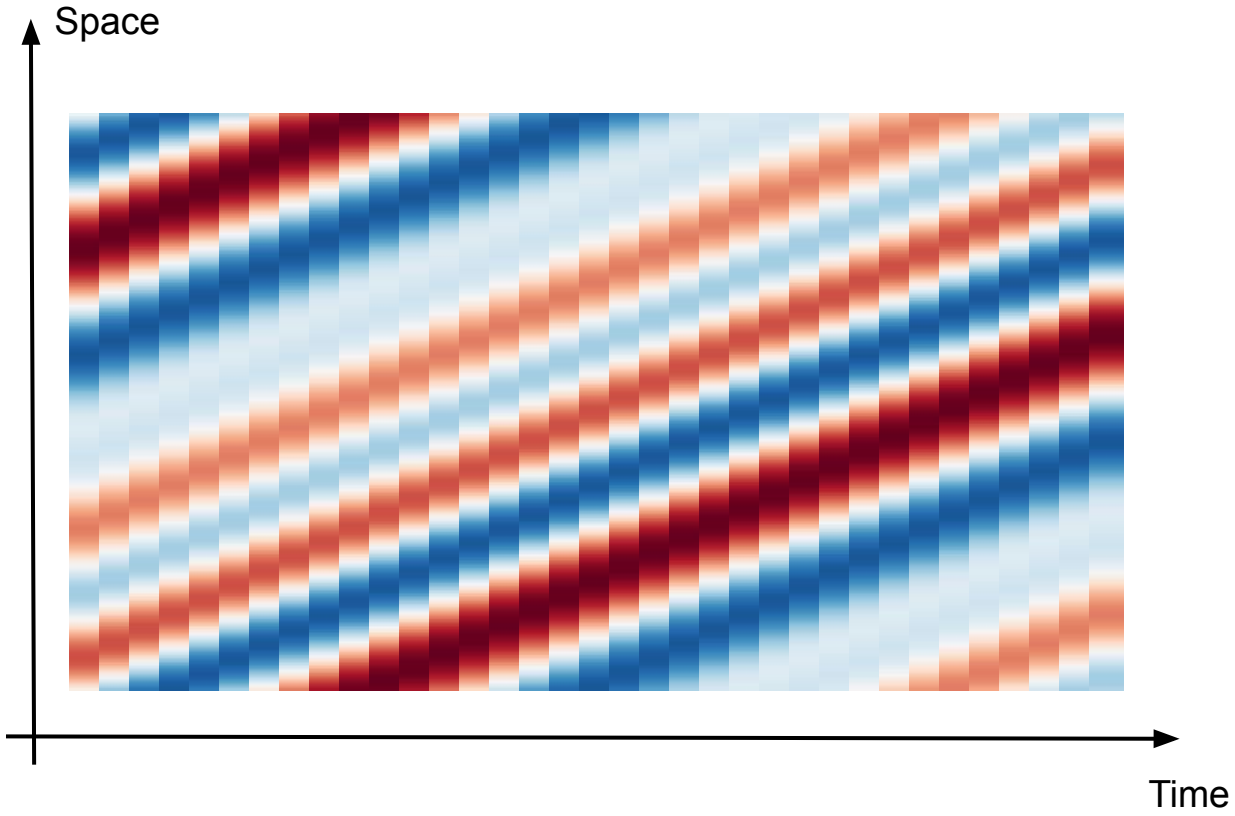
$$\mathcal{P}_h$$

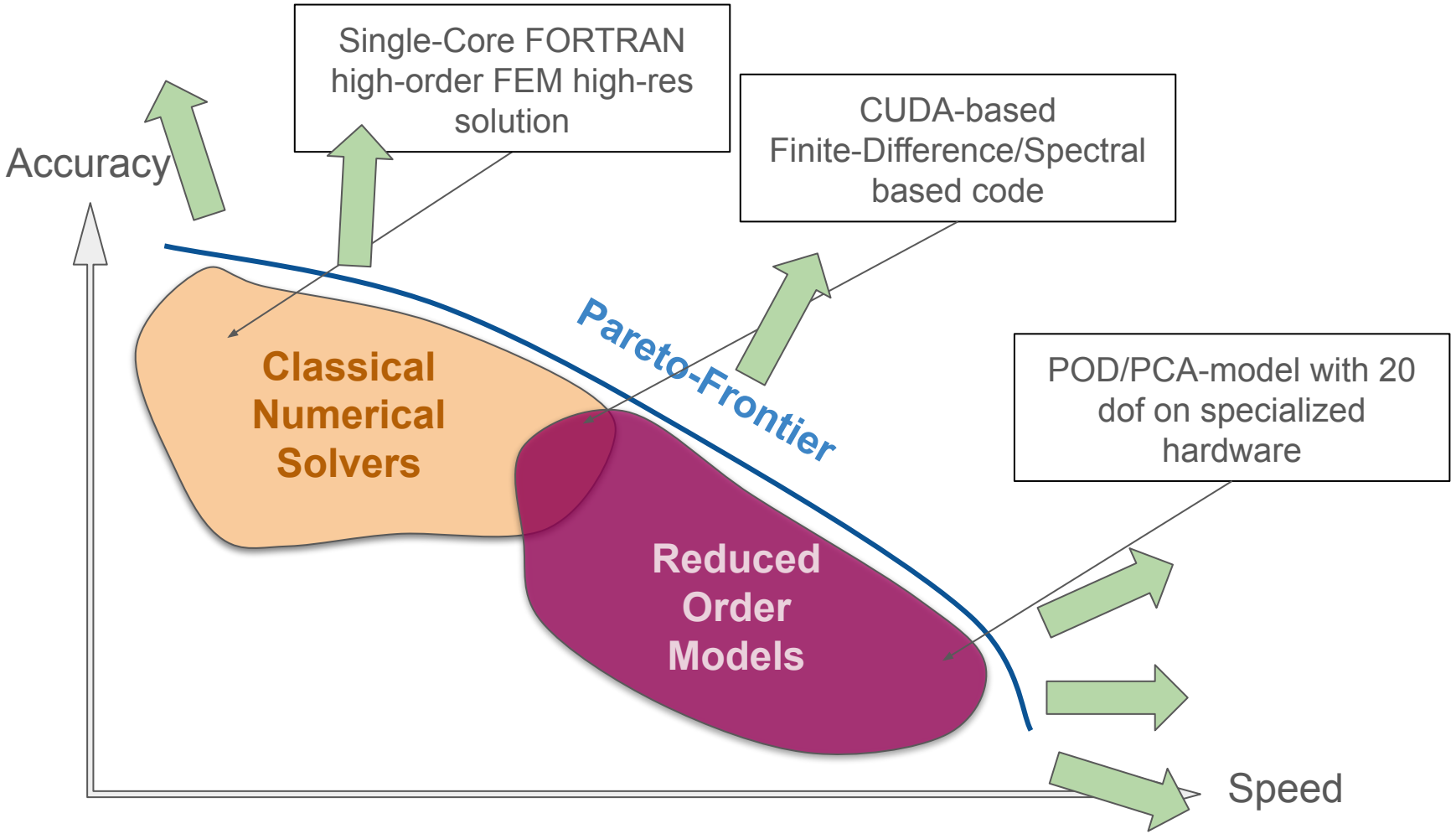
$$u^{[0]}(x) = \sum_i a_i \sin\left(i\frac{2\pi}{L}x - \phi_i\right)$$

Discretization




$\mathcal{P}_h$


Space
Time


Single-Core FORTRAN high-order FEM high-res solution
CUDA-based Finite-Difference/Spectral based code
Accuracy
Pareto-Frontier
Classical Numerical Solvers
POD/PCA-model with 20 dof on specialized hardware
Reduced Order Models
Speed

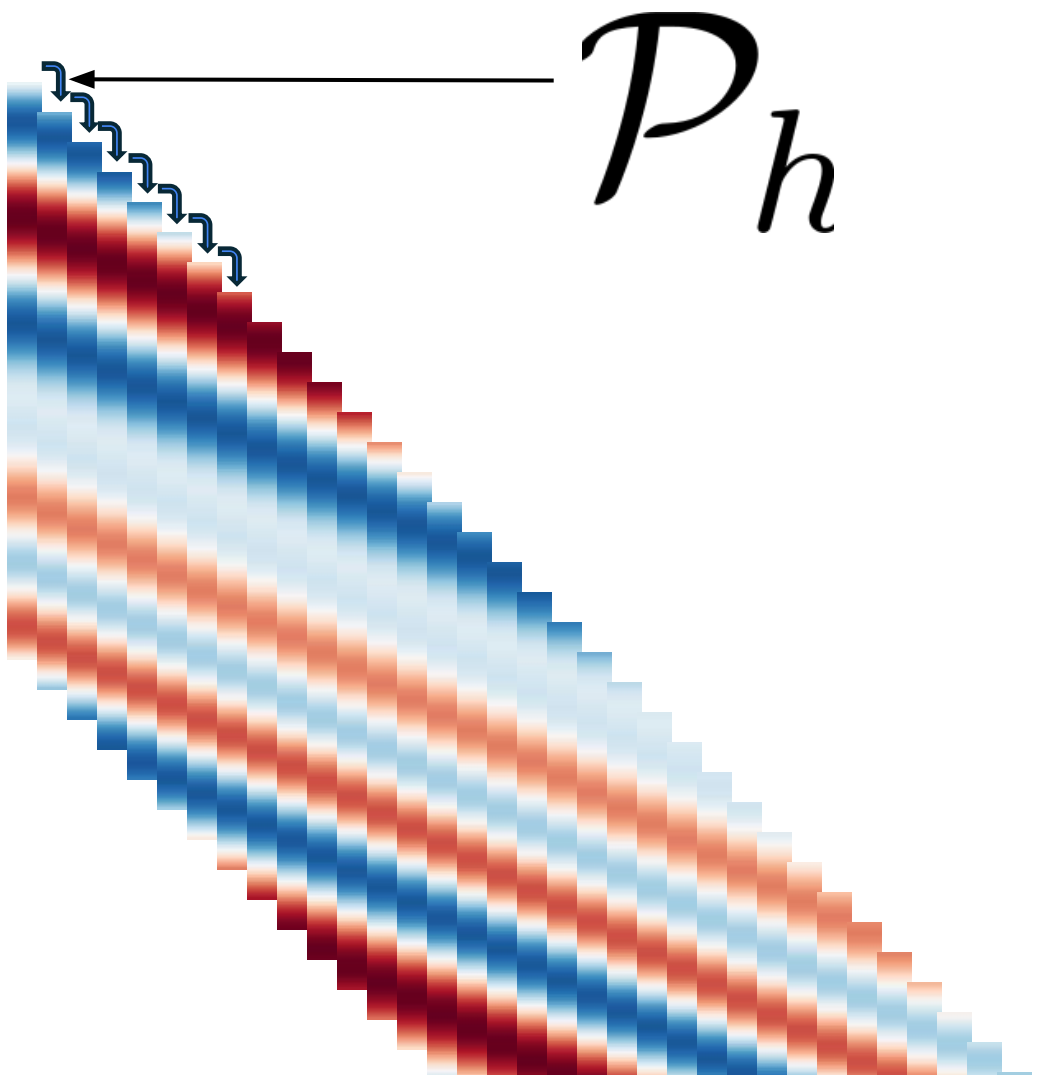
What do?

- **Physics-Informed Neural Networks** aka continuous solution functions
- **Smaller Components** for parts that are heuristic anyway:
  - In Modelling: Multi-scale models like turbulence models
  - In Numerics: Analyzers like flux limiters or step-size controllers
- **Augment** existing solvers on state-discrete representation (solver-in-the-loop)
- Fully **replace numerical solver** with neural models
- …

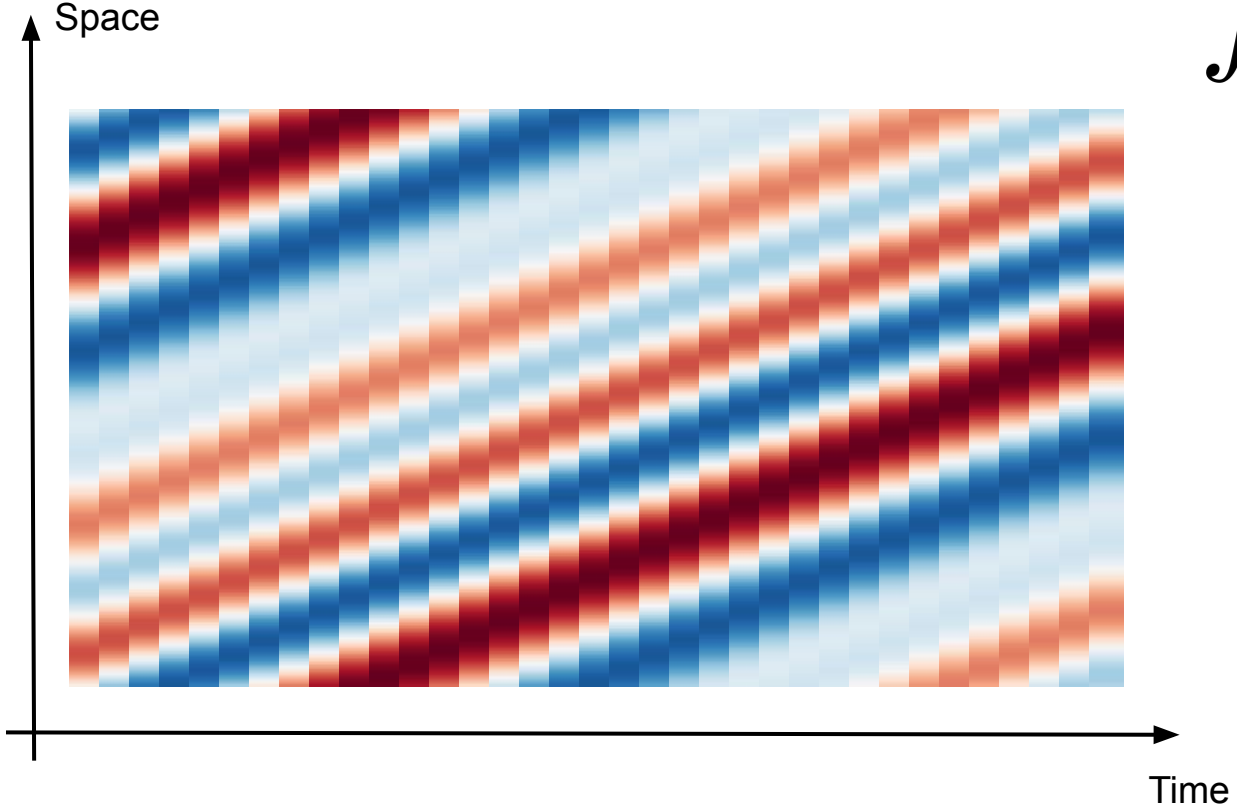
Why fully replace the numerical solver?

- Find sth more-efficient (pushes the Pareto frontier)
- “Imitation Learning”:
  - Bring existing numerical solvers:
    - to the GPU
    - into deep learning frameworks (for downstream tasks like control)
  - get features like differentiability or vectorized execution
- Understanding the learning process:
  - kind of “automating the numerical analysis of deriving a new scheme”
- Combine different sources of data (from experiments, existing numerical simulations, etc.)

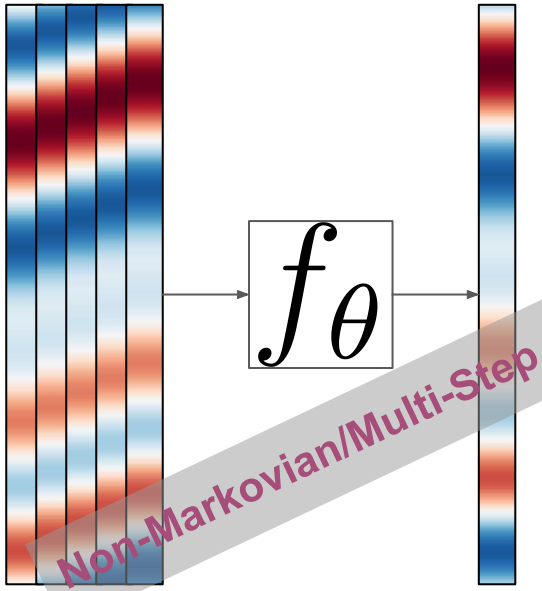
Autoregressive Emulators


$\mathcal{P}_h$

$$f_\theta \approx \mathcal{P}_h$$

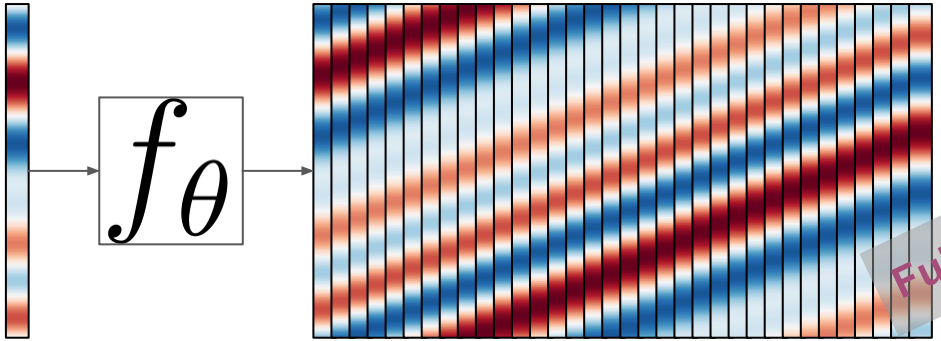


First-Order Markovian



Non-Markovian/Multi-Step

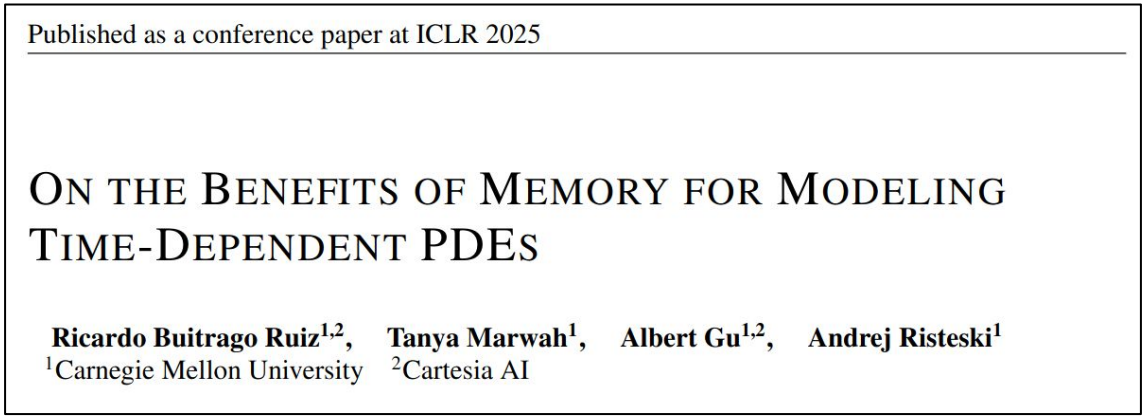
Temporal Bundling/Chunked



Full Trajectory Prediction

Which approach on time?

- Non-Markovian: Mori-Zwanzig formalism and Adams-Bashforth methods
- Chunked Prediction: Efficiency and inductive bias?
- Full Trajectory Prediction: Like video-diffusion models (shortcut integration)
- **First-Order Markovian: Simplicity and so is Physics?!**

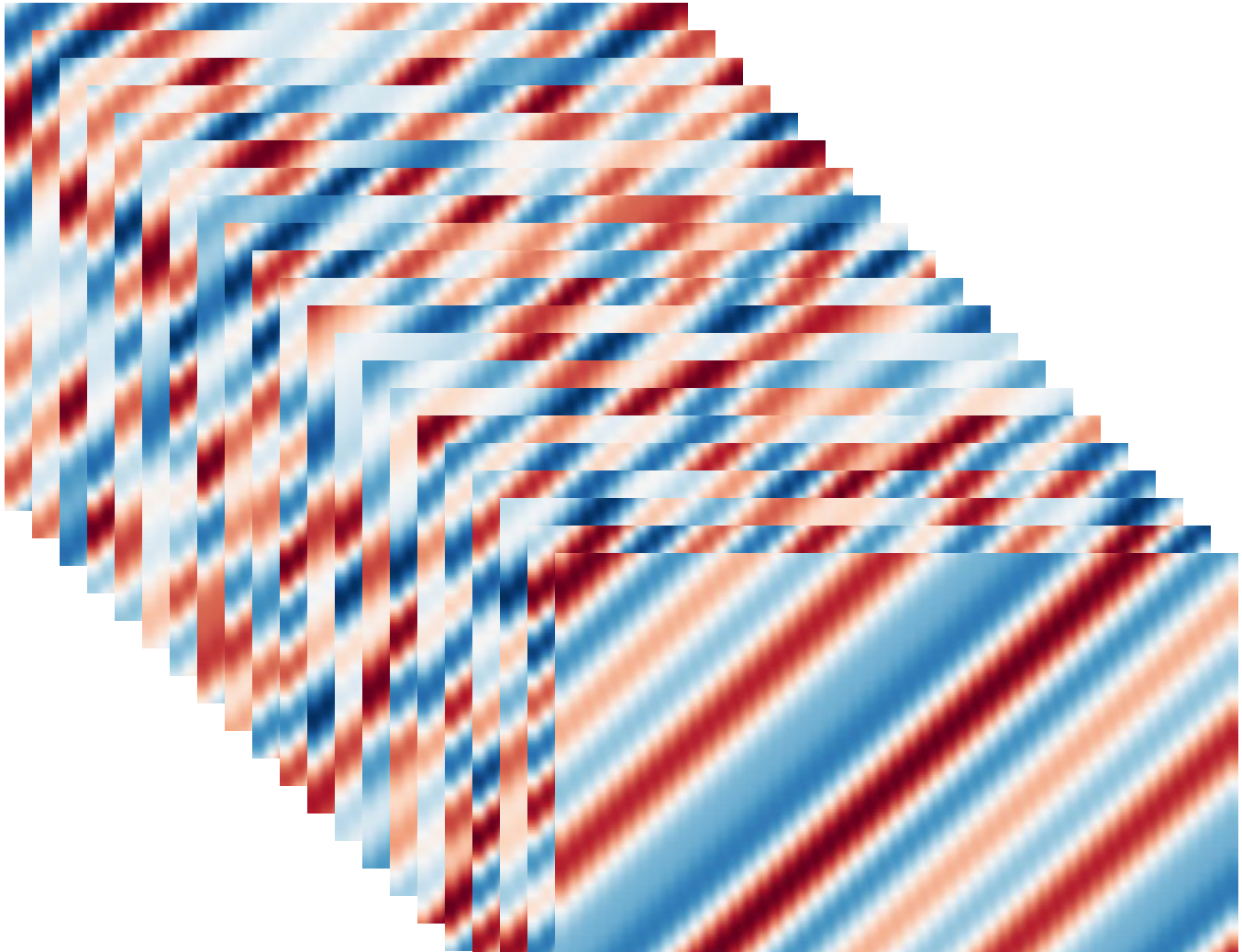


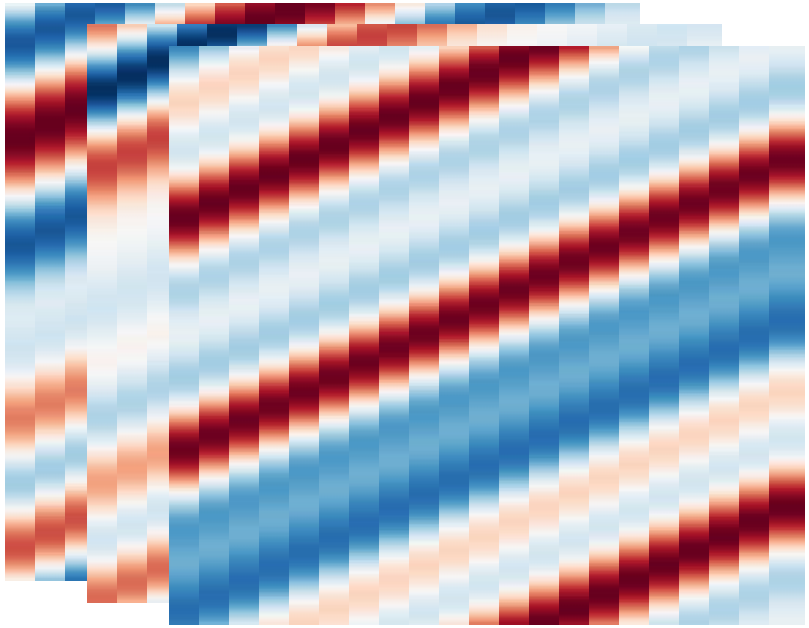
Published as a conference paper at ICLR 2025

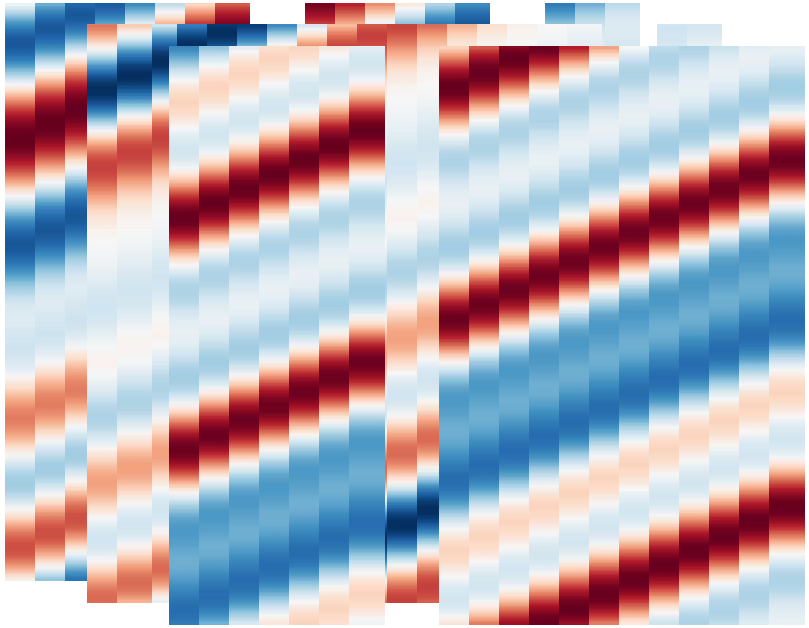
ON THE BENEFITS OF MEMORY FOR MODELING TIME-DEPENDENT PDES

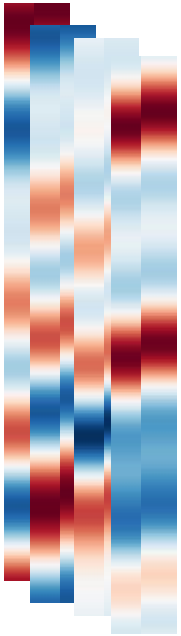
Ricardo Buitrago Ruiz[1,2], **Tanya Marwah**[1], **Albert Gu**[1,2], **Andrej Risteski**[1]
[1]Carnegie Mellon University [2]Cartesia AI

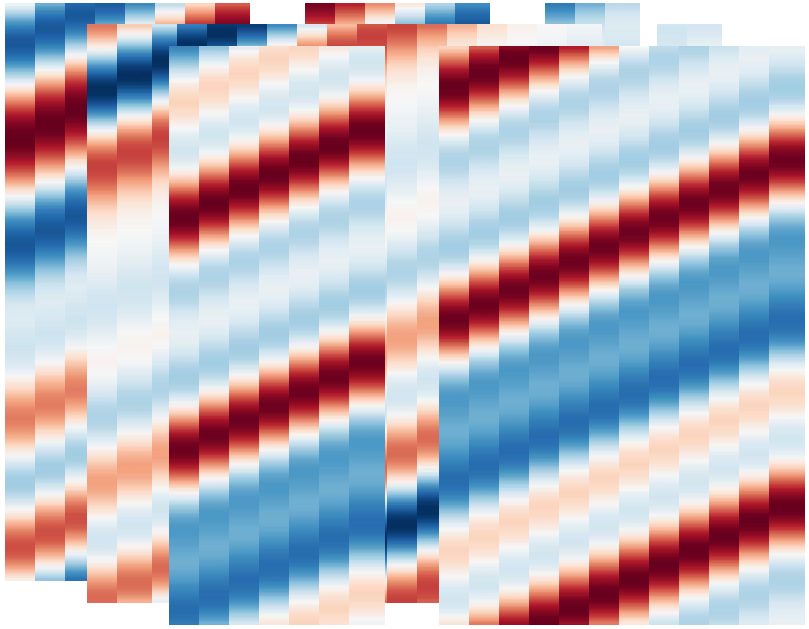
How to train (Markovian) Autoregressive Emulators

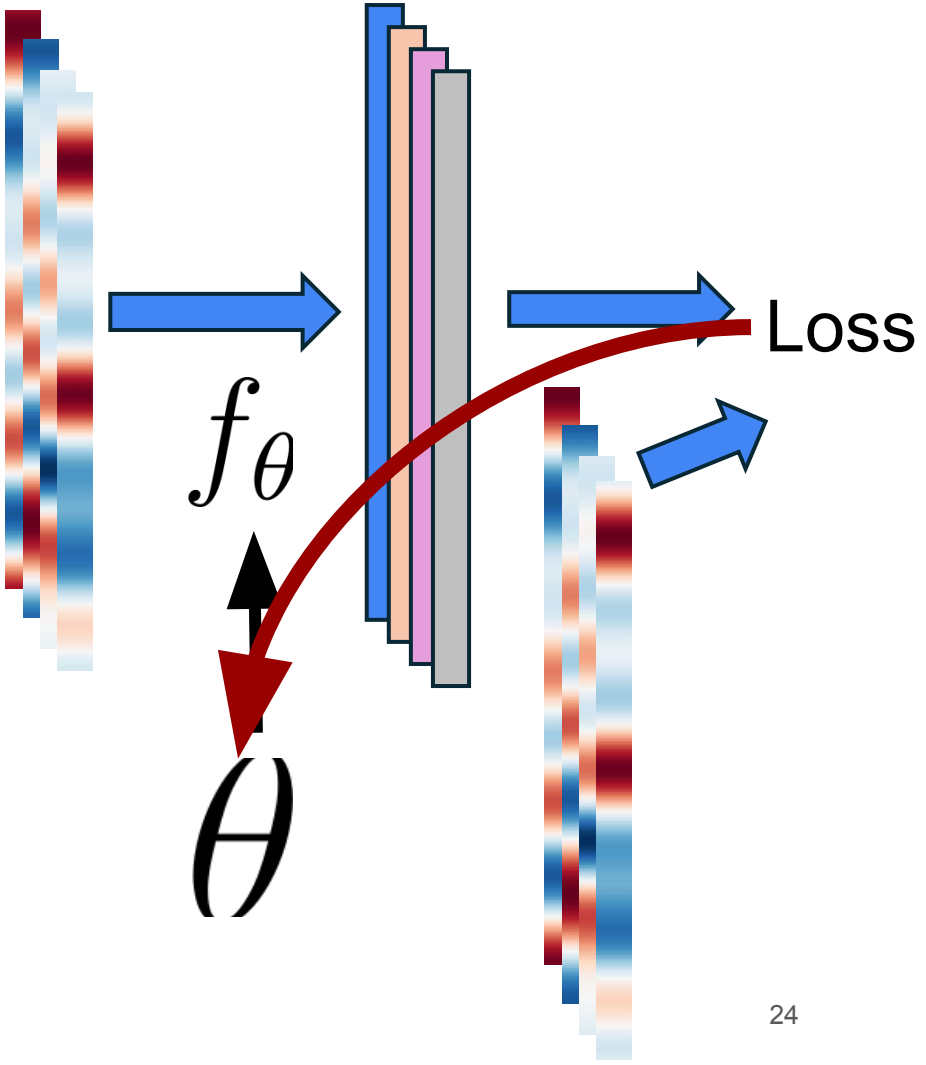


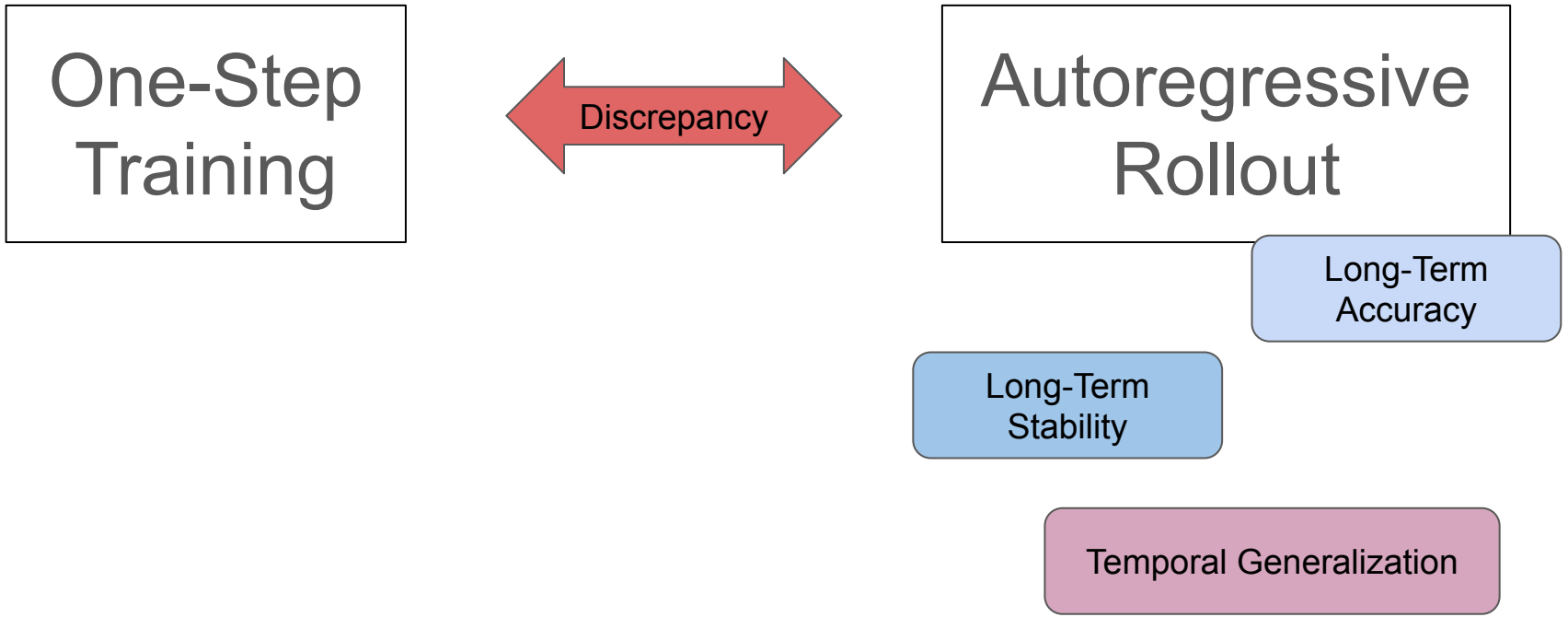


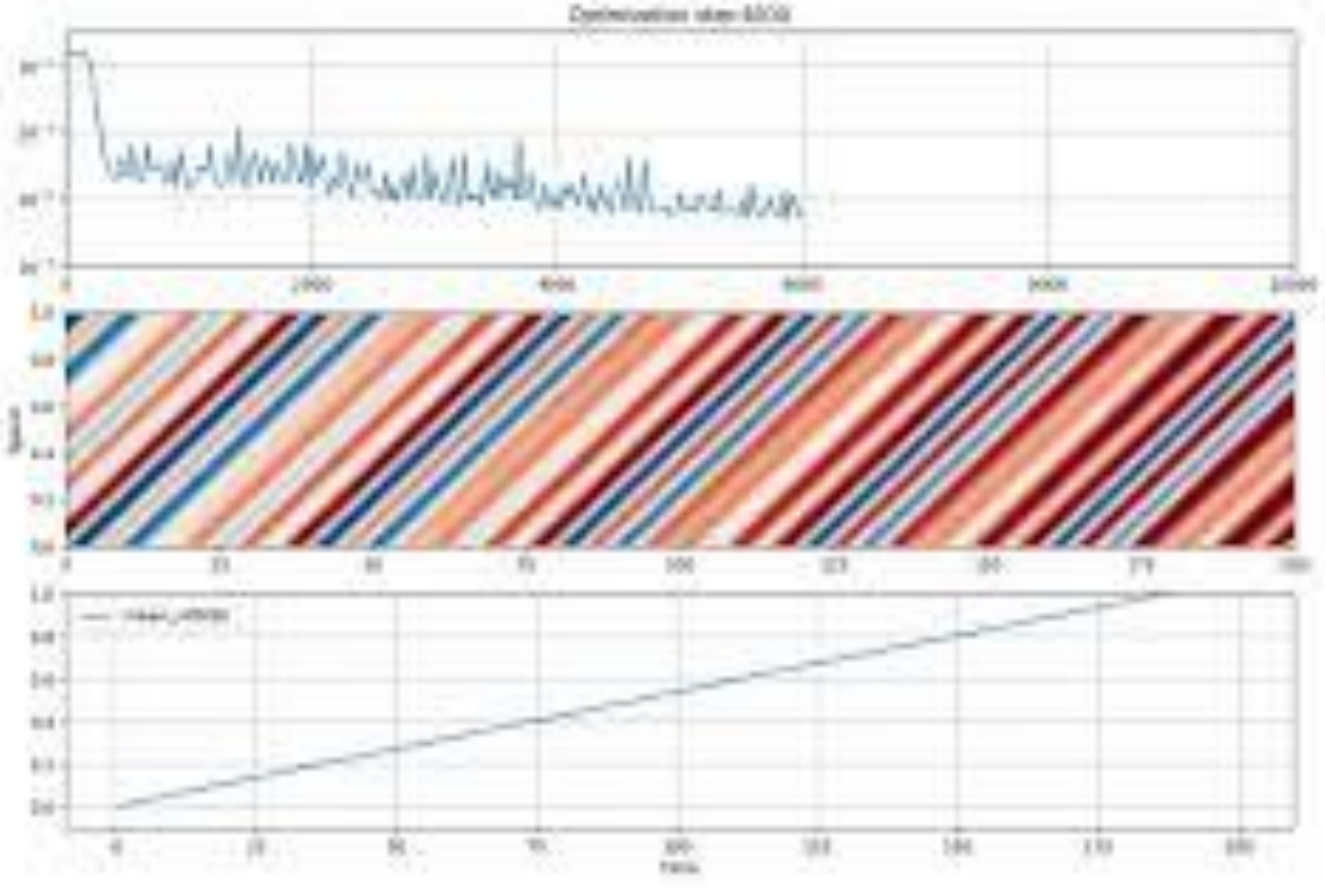







Loss
f_θ
θ


One-Step Training
Discrepancy
Autoregressive Rollout
Long-Term Accuracy
Long-Term Stability
Temporal Generalization



APEBench

PDE

TimeStepper

IC Distribution

IC State

Trajectory

Data Trajectories

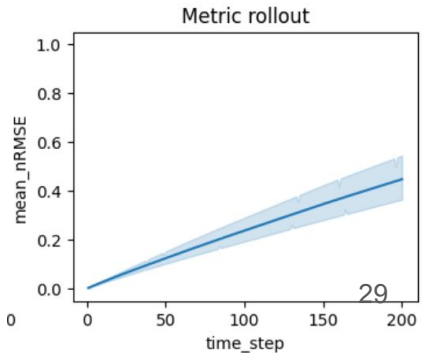
Test

Neural Arch

Train

Metric

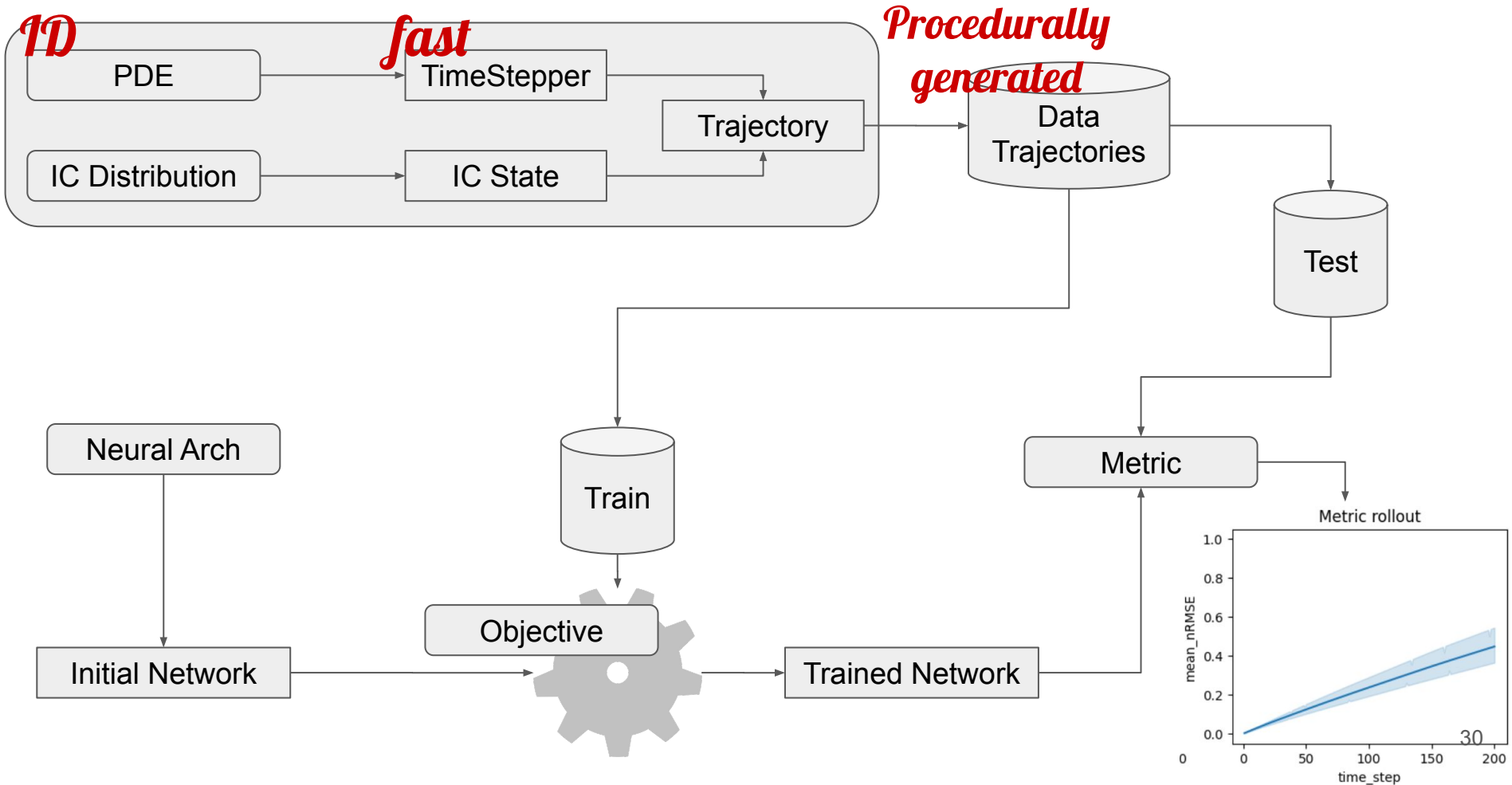
Metric rollout



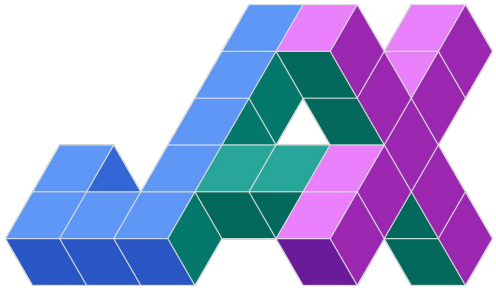
Objective

Initial Network

Trained Network


1D
fast
Procedurally generated
PDE
TimeStepper
Trajectory
IC Distribution
IC State
Data Trajectories
Test
Neural Arch
Train
Metric
Metric rollout
Objective
Initial Network
Trained Network
mean_nRMSE
time_step
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200

Key Contribution: Fourier-ETDRK in JAX



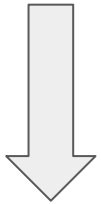
$$\partial_t u = \mathcal{L} u + \mathcal{N}(u)$$

$$\hat{u}_h^{[t+1]} = \exp(\hat{\mathcal{L}}_h \Delta t) \odot \hat{u}_h^{[t]} + \frac{\exp(\hat{\mathcal{L}}_h \Delta t) - 1}{\hat{\mathcal{L}}_h} \odot \hat{\mathcal{N}}_h(\hat{u}_h^{[t]})$$

PDE Identifiers (Difficulty-based interface)

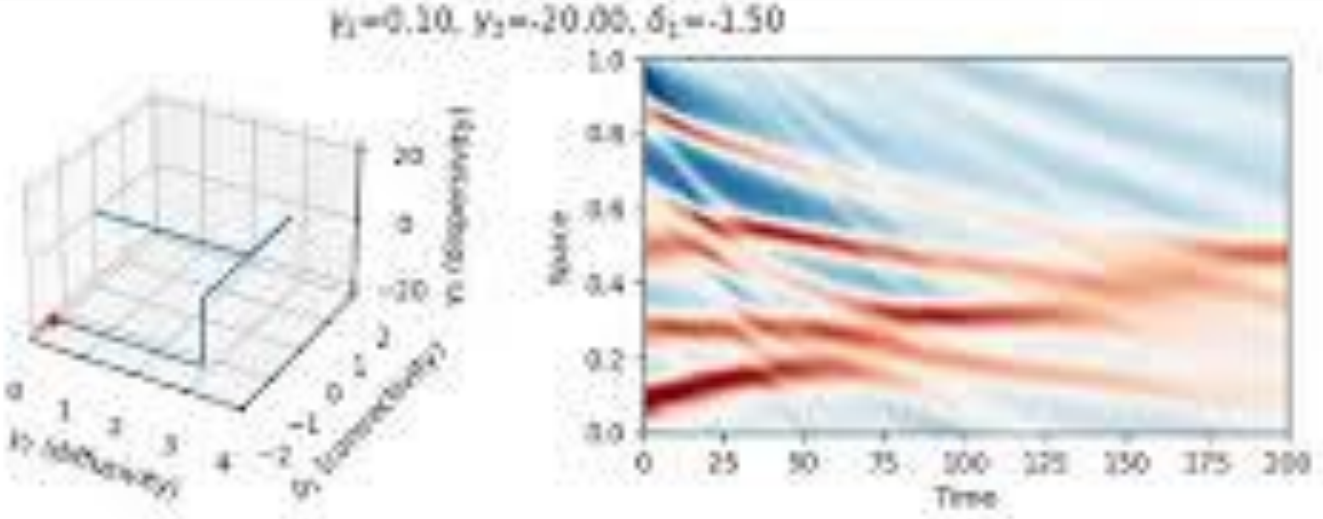
$$\frac{\partial u}{\partial t} = b_1 \frac{1}{2}\frac{\partial u^2}{\partial x} + a_0 u + a_1 \frac{\partial u}{\partial x} + a_2 \frac{\partial^2 u}{\partial x^2} + a_3 \frac{\partial^3 u}{\partial x^3} + a_4 \frac{\partial^4 u}{\partial x^4} + \dots$$

$$\{b_1, a_0, a_1, a_2, a_3, a_3, ...\} \times \{D, L, N, \Delta t\} \times \text{numerics}$$



\+ Stable 2nd order ETDRK

$$\{\delta_1, \gamma_0, \gamma_1, \gamma_2, \gamma_3, \gamma_4, \dots\} \times \{D, N\}$$

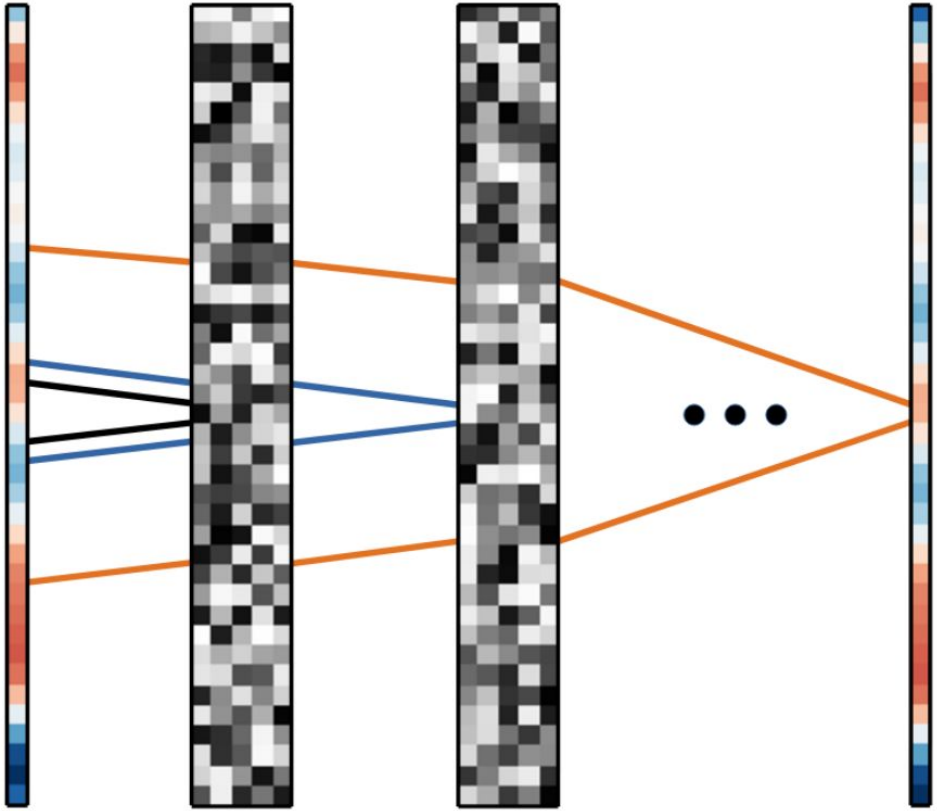

γ_1=0.10, γ_2=-20.00, δ_1=-1.50
20
0
-20
1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
200
Time

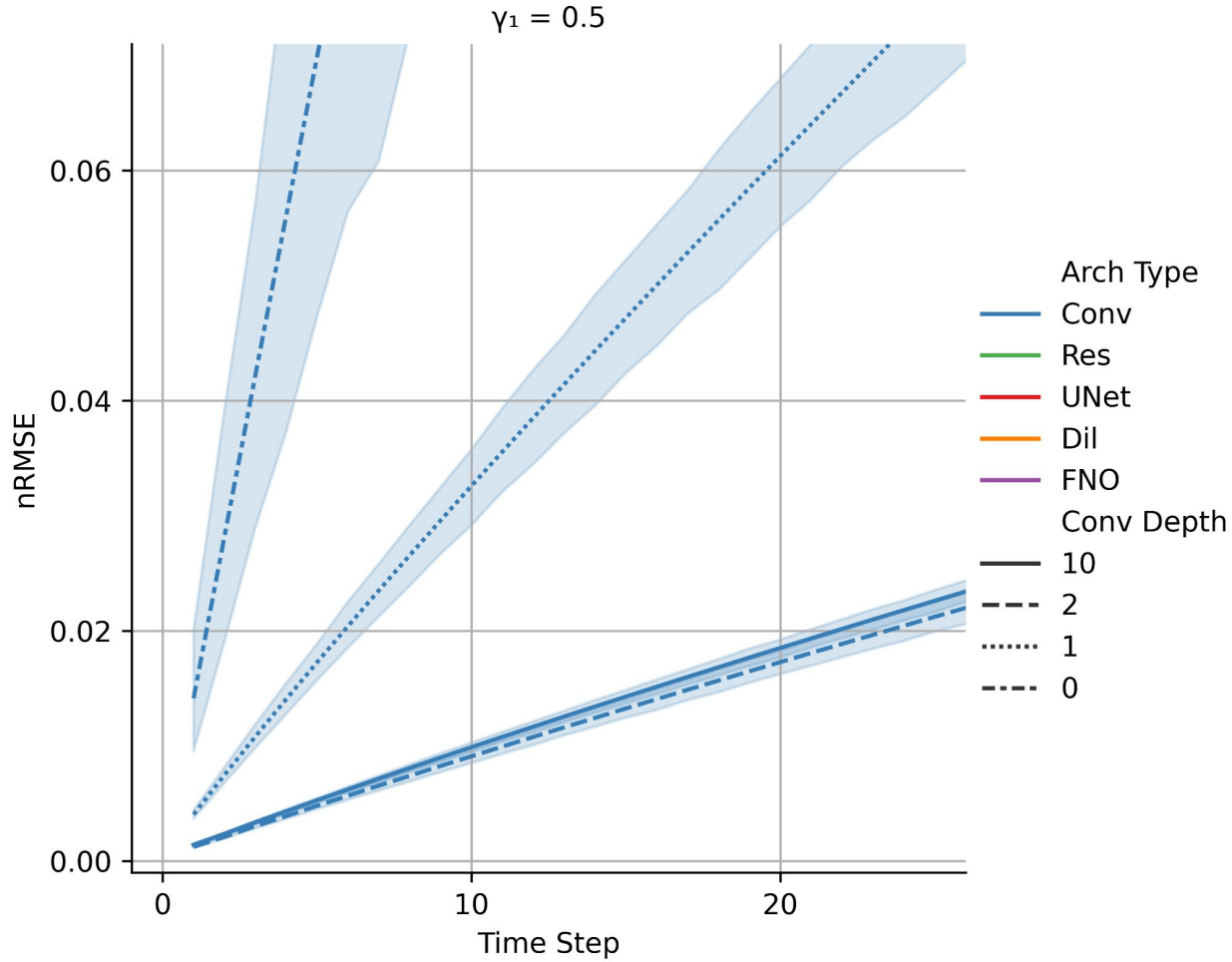
Axes of Investigation

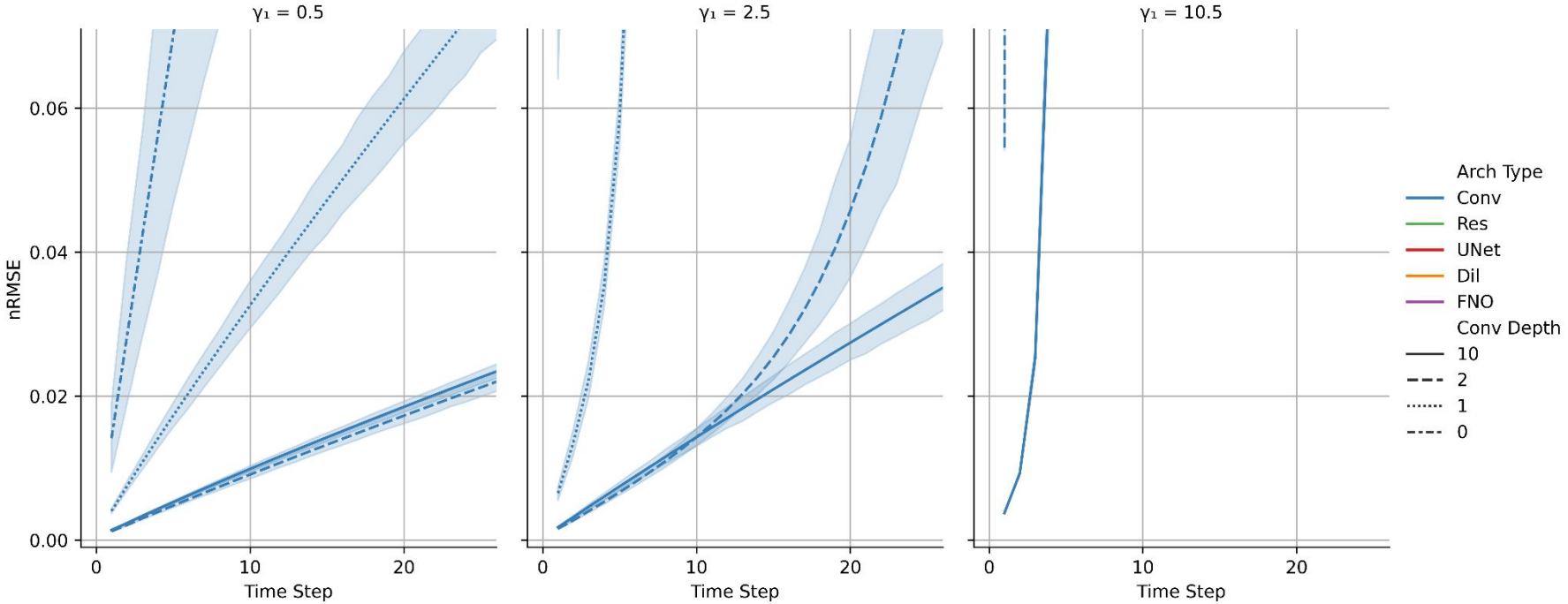
- **PDE (in terms of its difficulty, resolution and spatial dims)**
- **Neural Architecture**
- Learning Methodology (unrolled training, differentiable physics, …)
- Learning Goal (prediction or neural-hybrid correction)

+ seed statistics for hypothesis testing

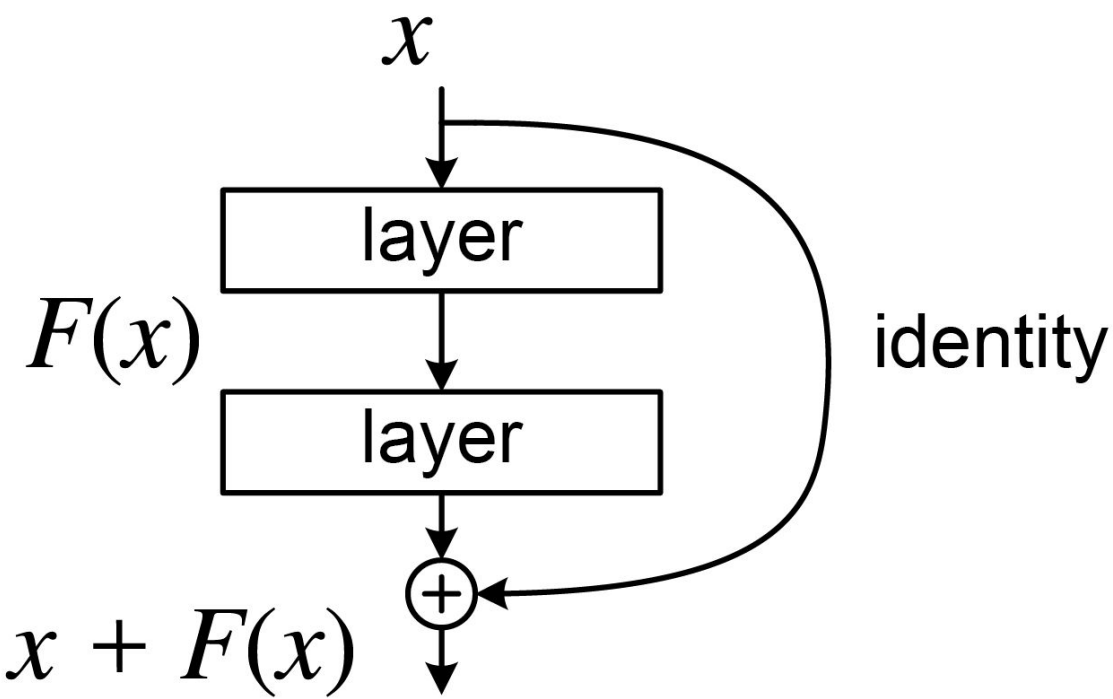
ConvNets as Autoregressive Emulators



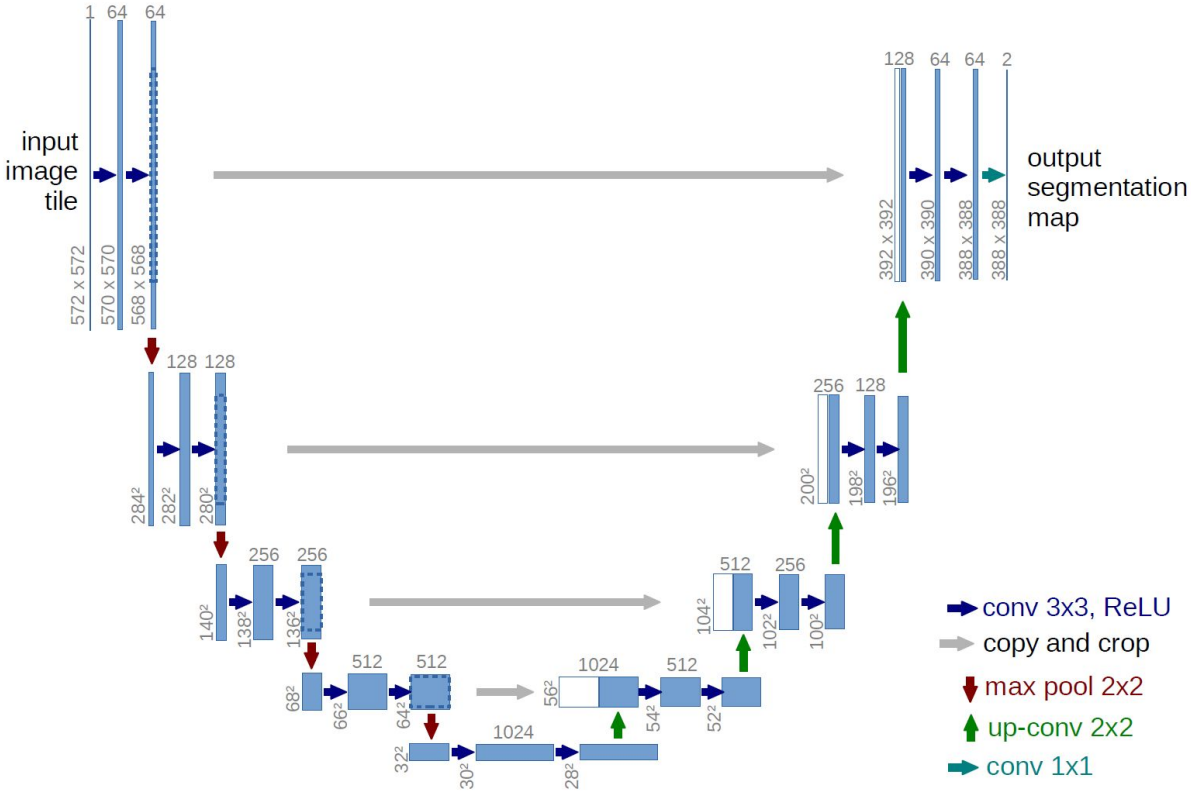

$\gamma_1 = 0.5$
nRMSE
0.00
0.02
0.04
0.06
0
10
20
Time Step
Arch Type
Conv
Res
UNet
Dil
FNO
Conv Depth
10
2
1
0


γ₁ = 0.5
γ₁ = 2.5
γ₁ = 10.5
nRMSE
0.00
0.02
0.04
0.06
0
10
20
Time Step
Arch Type
Conv
Res
UNet
Dil
FNO
Conv Depth
10
2
1
0

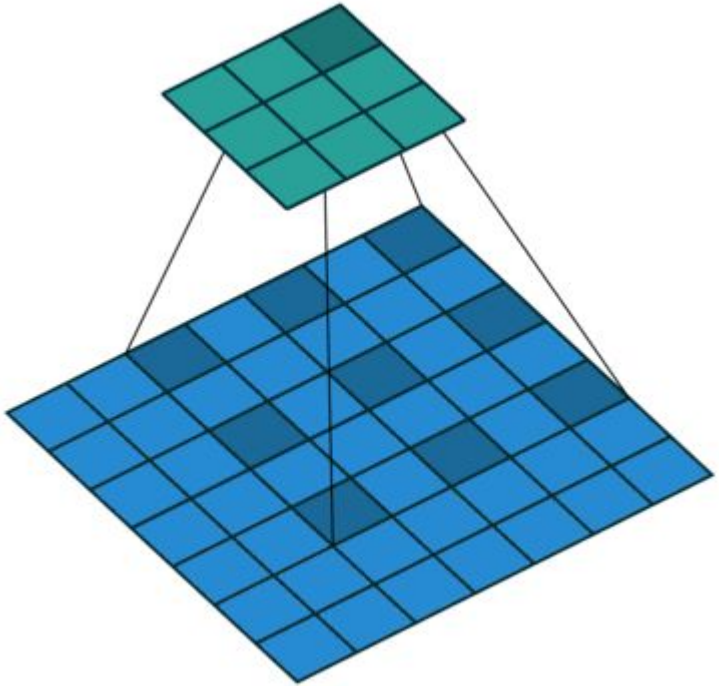
More Architectures - ResNets



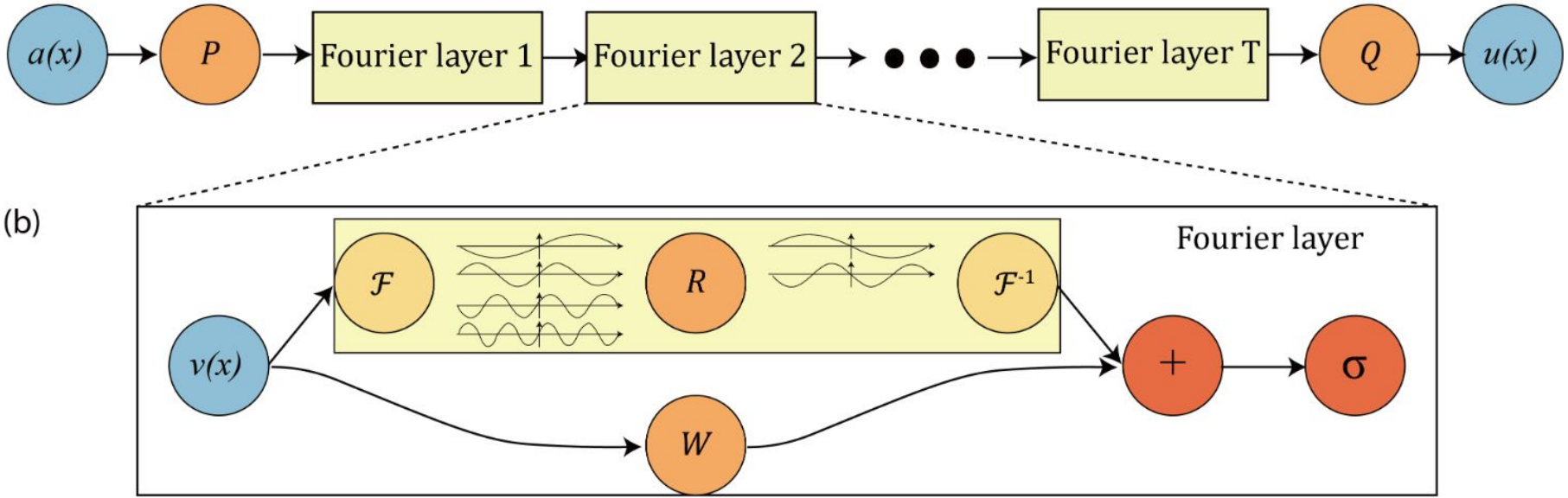
More Architectures - UNets

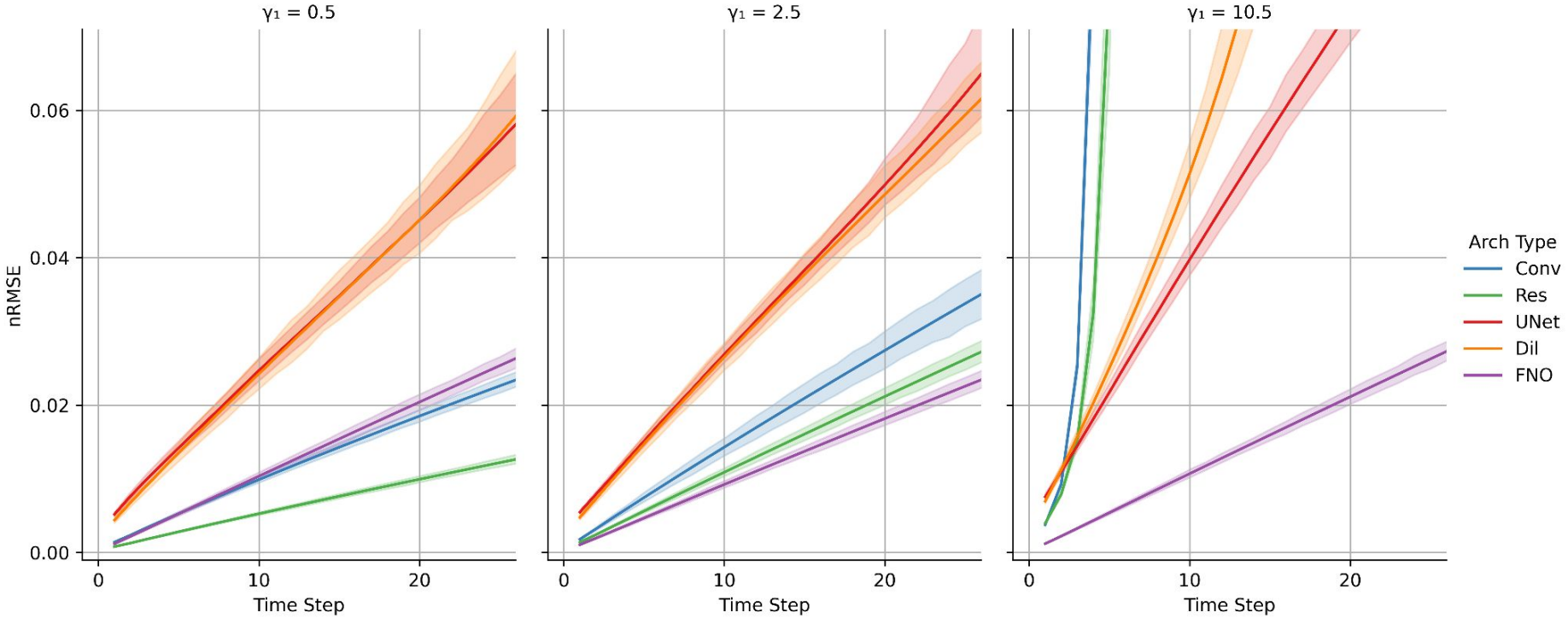


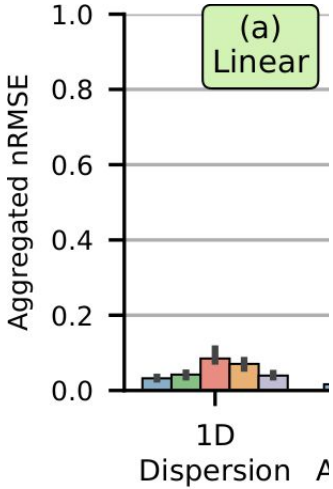
More Architectures - Dilated ConvNets



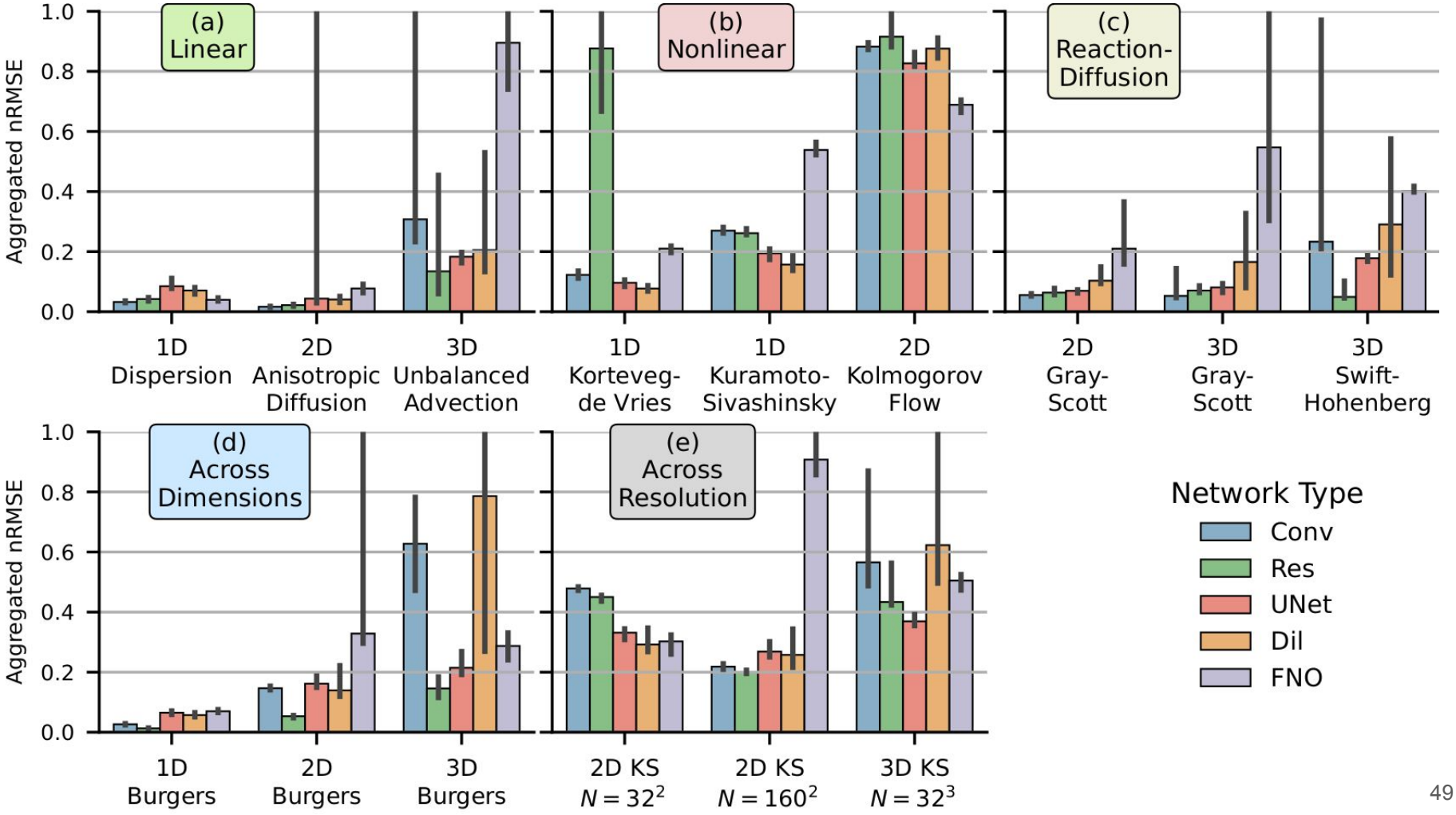
More Architectures - Fourier Neural Operators

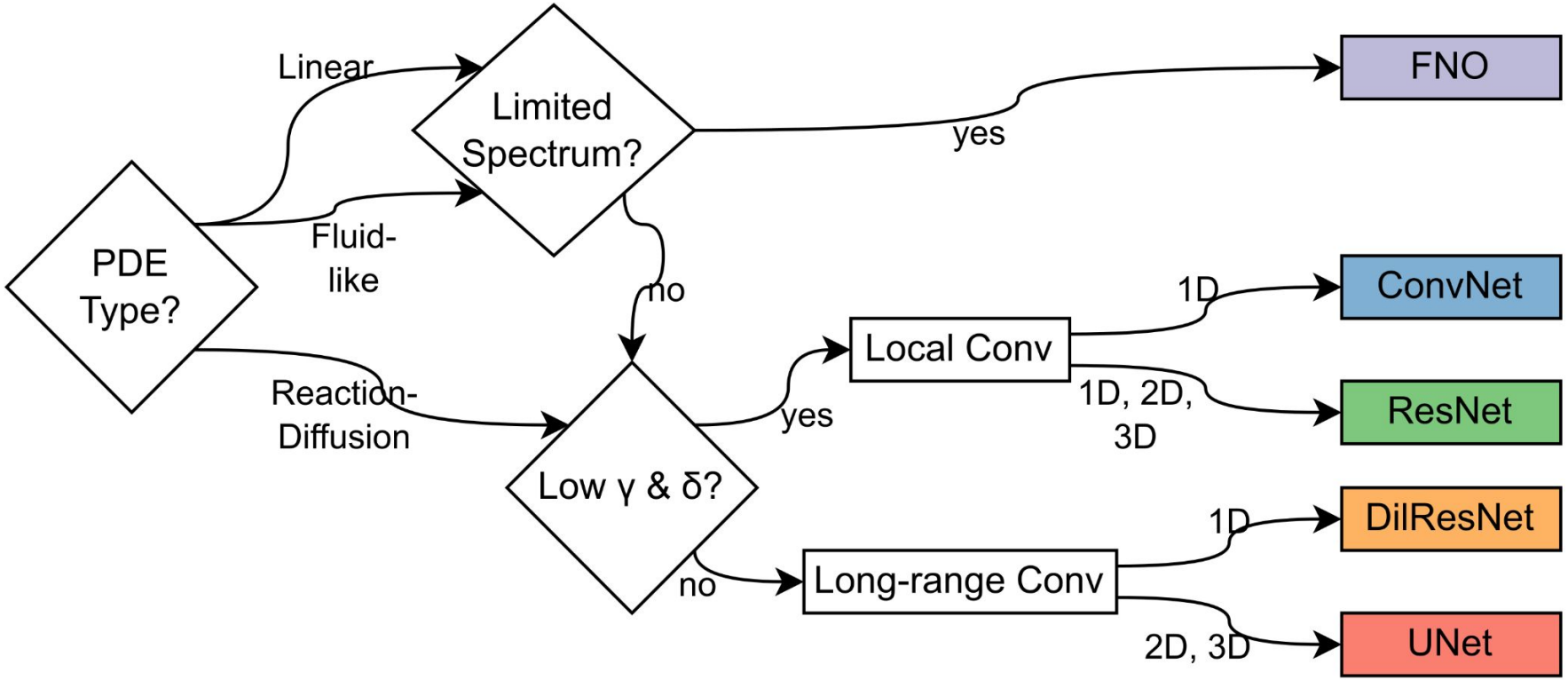



γ₁ = 0.5
γ₁ = 2.5
γ₁ = 10.5
nRMSE
0.00
0.02
0.04
0.06
0
10
20
Time Step
Arch Type
Conv
Res
UNet
Dil
FNO

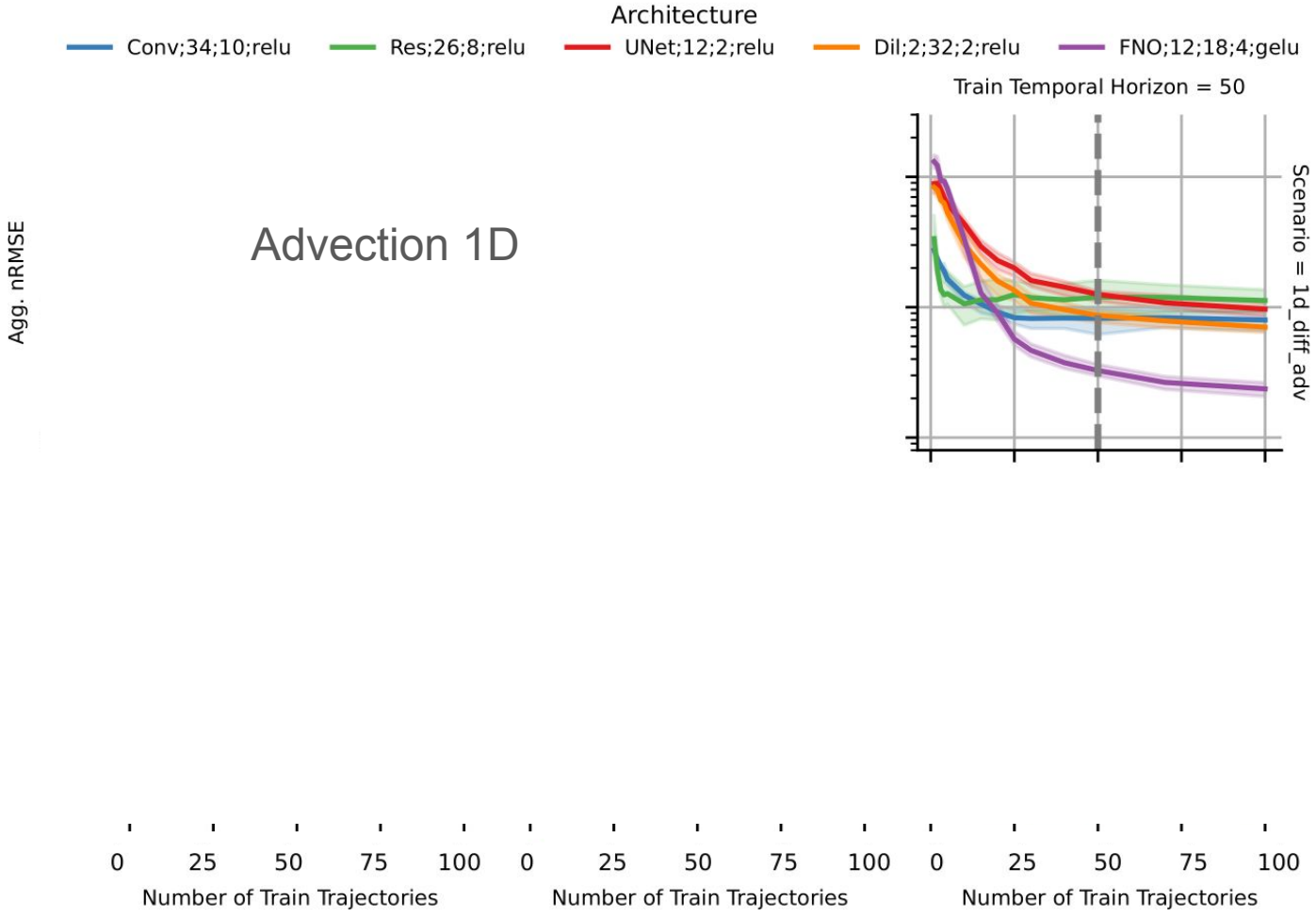


Network Type
Conv
Res
UNet
Dil
FNO

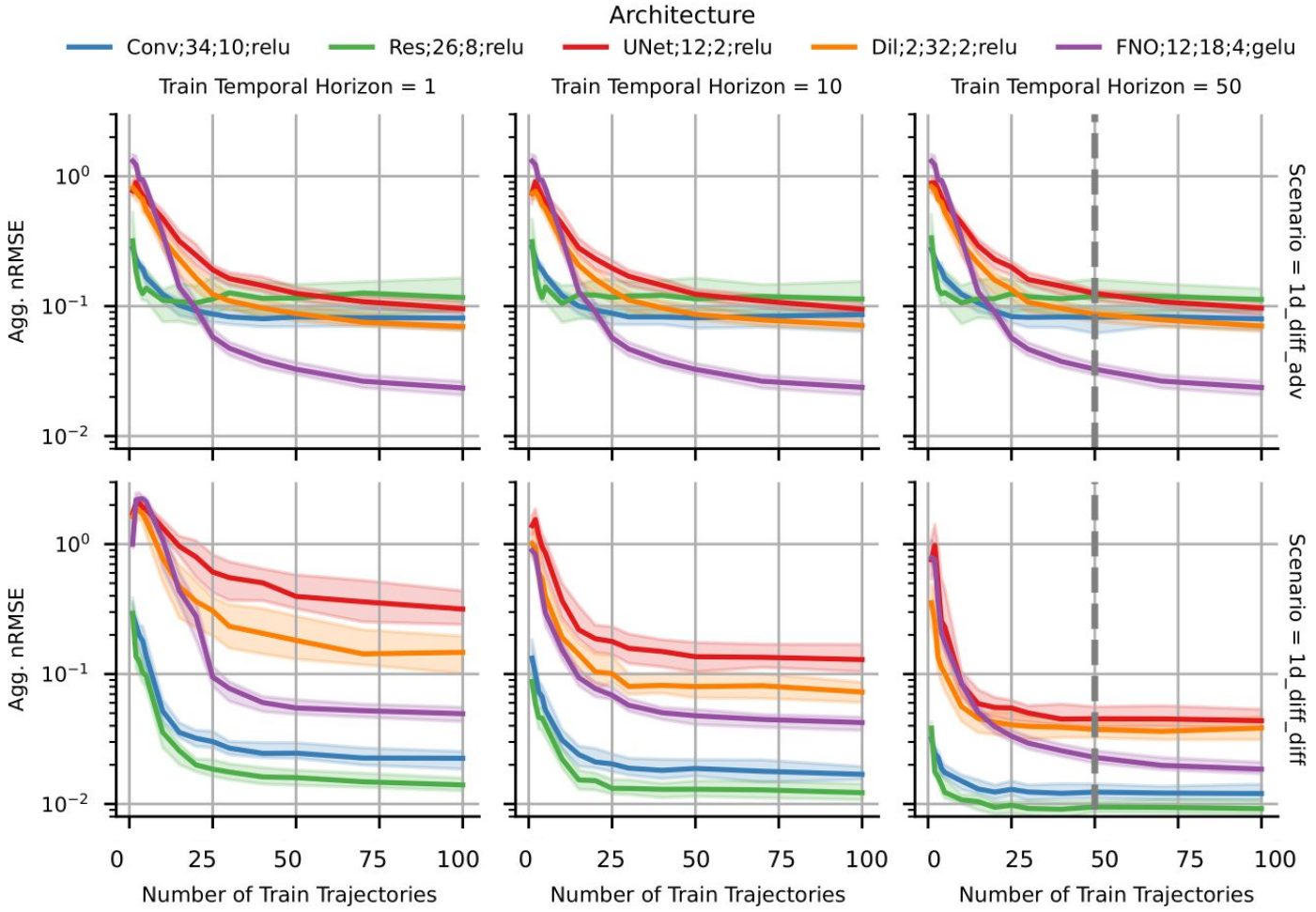

(a) Linear
(b) Nonlinear
(c) Reaction-Diffusion
(d) Across Dimensions
(e) Across Resolution
Aggregated nRMSE
1.0
0.8
0.6
0.4
0.2
0.0
1D Dispersion
2D Anisotropic Diffusion
3D Unbalanced Advection
1D Korteveg-de Vries
1D Kuramoto-Sivashinsky
2D Kolmogorov Flow
2D Gray-Scott
3D Gray-Scott
3D Swift-Hohenberg
1D Burgers
2D Burgers
3D Burgers
2D KS $N = 32^2$
2D KS $N = 160^2$
3D KS $N = 32^3$
Network Type
Conv
Res
UNet
Dil
FNO

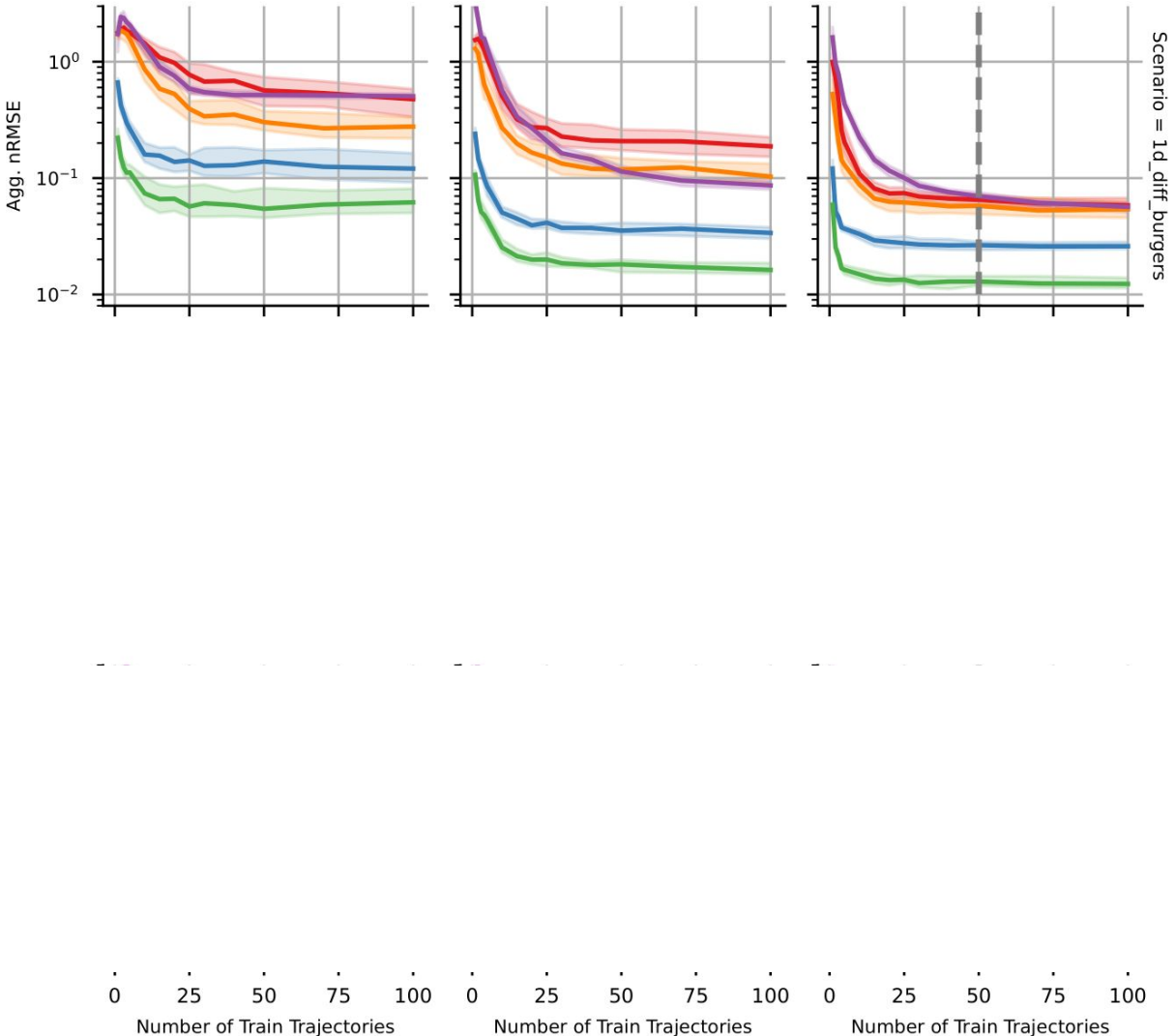

PDE Type?
Linear
Fluid-like
Reaction-Diffusion
Limited Spectrum?
yes
no
FNO
Low γ & δ?
yes
no
Local Conv
1D
1D, 2D, 3D
ConvNet
ResNet
Long-range Conv
1D
2D, 3D
DilResNet
UNet

Dataset Ablation

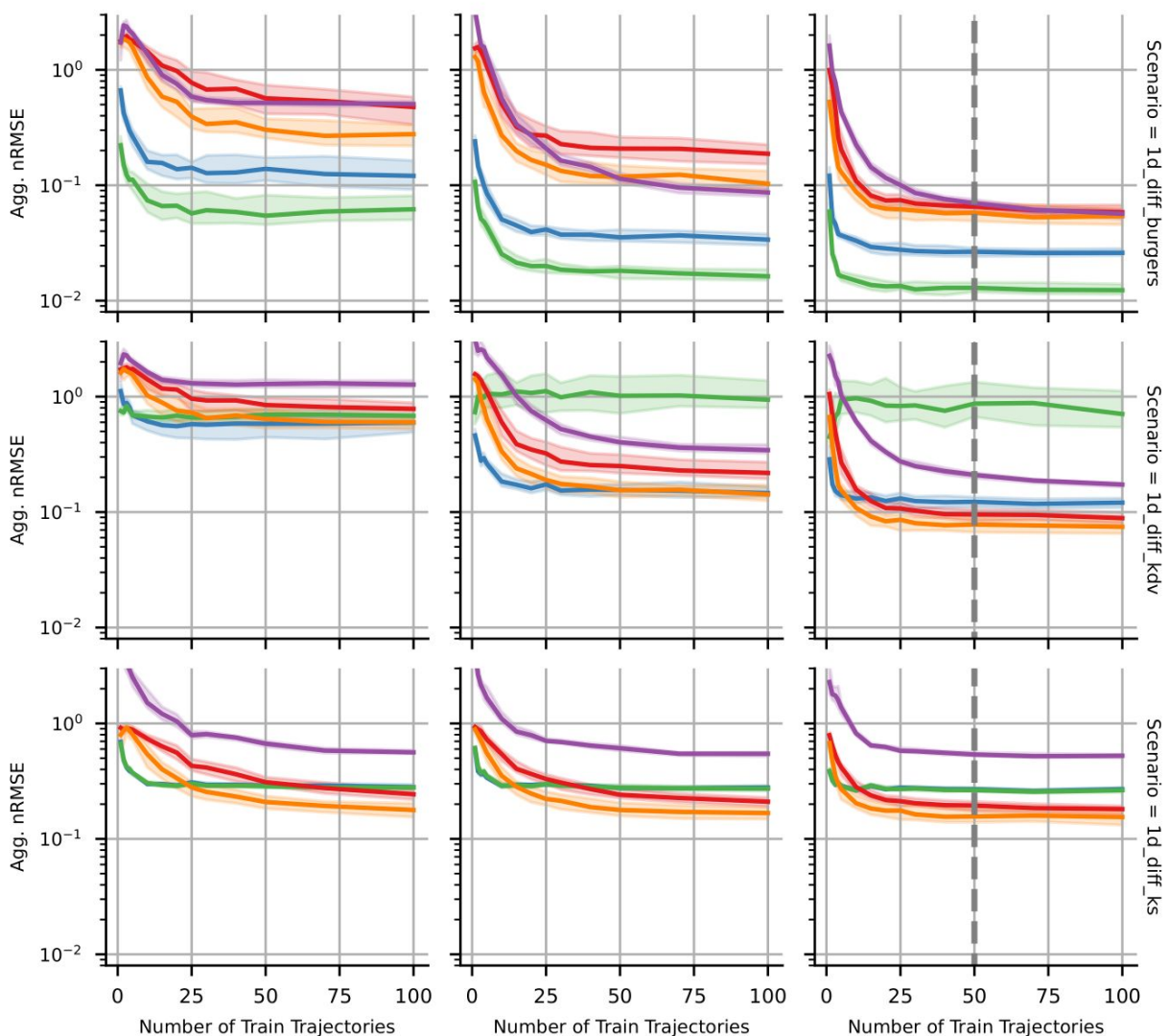


Dataset Ablation




Agg. nRMSE
10^0
10^{-1}
10^{-2}
Scenario = 1d_diff_burgers
0
25
50
75
100
Number of Train Trajectories
Number of Train Trajectories
Number of Train Trajectories

Burgers


Agg. nRMSE
10^0
10^{-1}
10^{-2}
Scenario = 1d_diff_burgers
Scenario = 1d_diff_kdv
Scenario = 1d_diff_ks
0
25
50
75
100
Number of Train Trajectories

Burgers

APEBENCH in a nutshell

- JAX-based
- Fast reference simulator based on spectral methods:
  - Procedural data generation in seconds!
  - Differentiable Physics
- 46 PDEs in 1D, 2D, and 3D with unique identifiers
- Big selection of modern Emulator Architectures in JAX
- Built-in unrolled training with differentiable physics
- Built-in neural hybrid emulation
- An integrated volume renderer for 2D & 3D
- Understand neural emulators and draw analogies with classical numerical methods

APEBench: A Benchmark for Autoregressive Neural Emulators of PDEs

Felix Koehler
Technical University of Munich
Munich Center for Machine Learning
f.koehler@tum.de

Simon Niedermayr
Technical University of Munich
simon.niedermayr@tum.de

Rüdiger Westermann
Technical University of Munich
westermann@tum.de

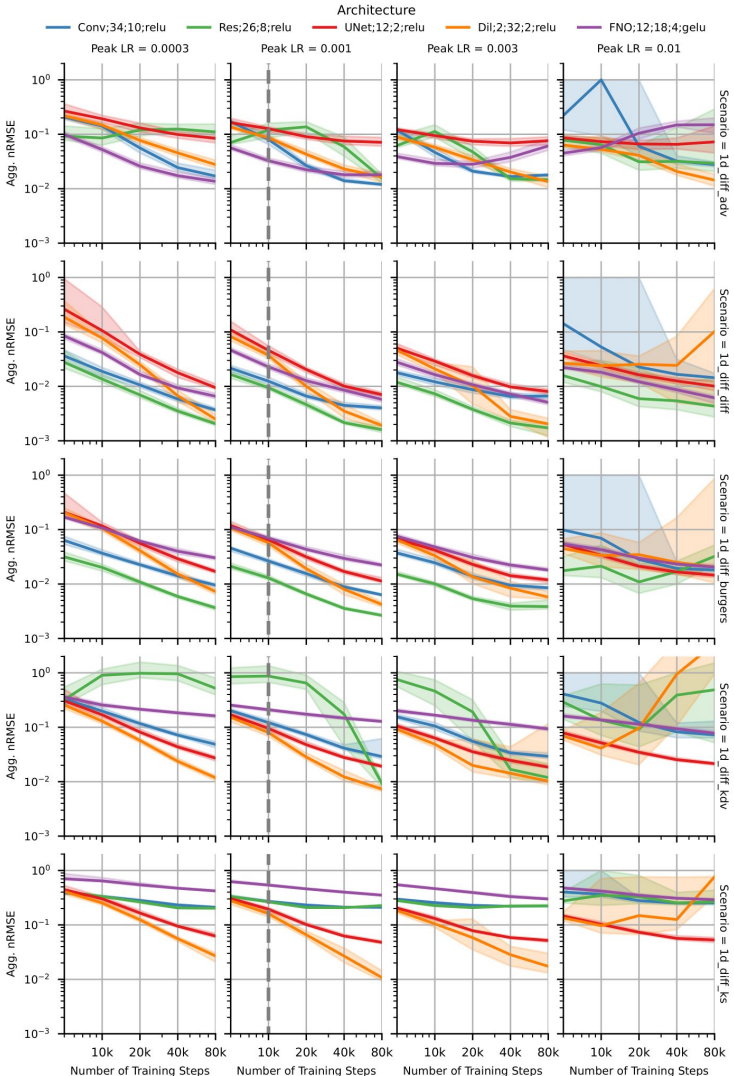
Nils Thuerey
Technical University of Munich
nils.thuerey@tum.de

```
pip install apebench
```

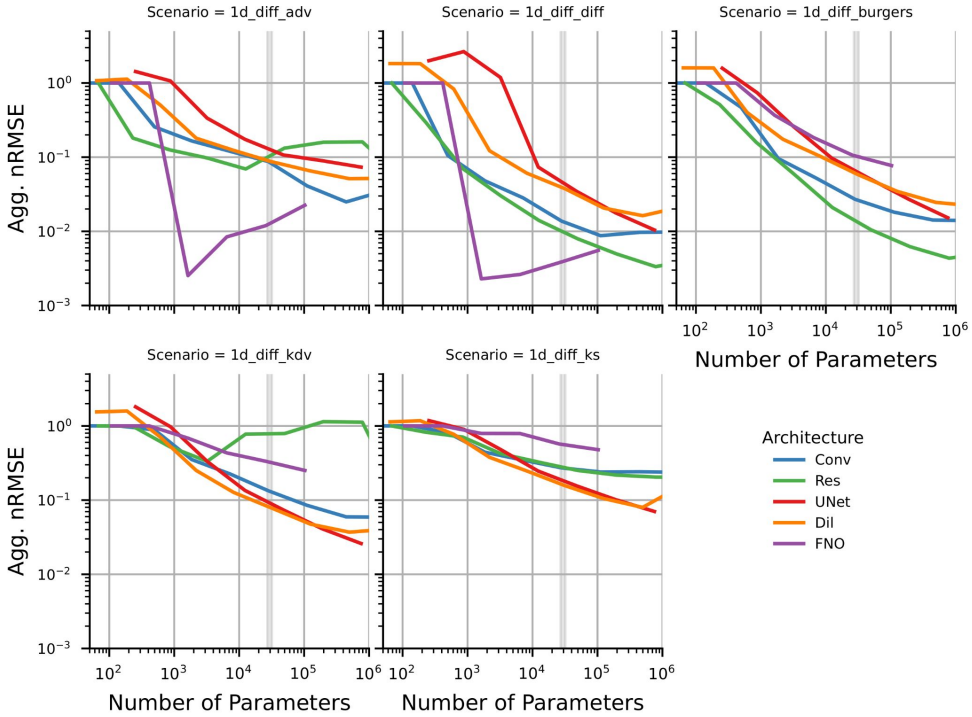
tum-pbs.github.io/apebench

Backup Slides

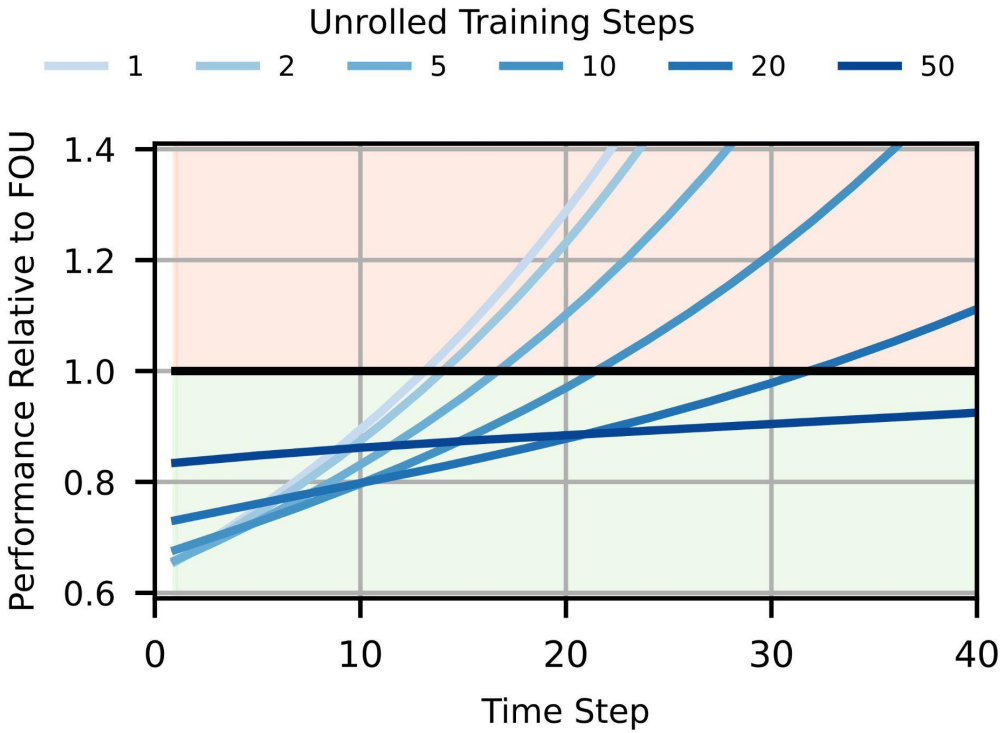
Optimization Ablation

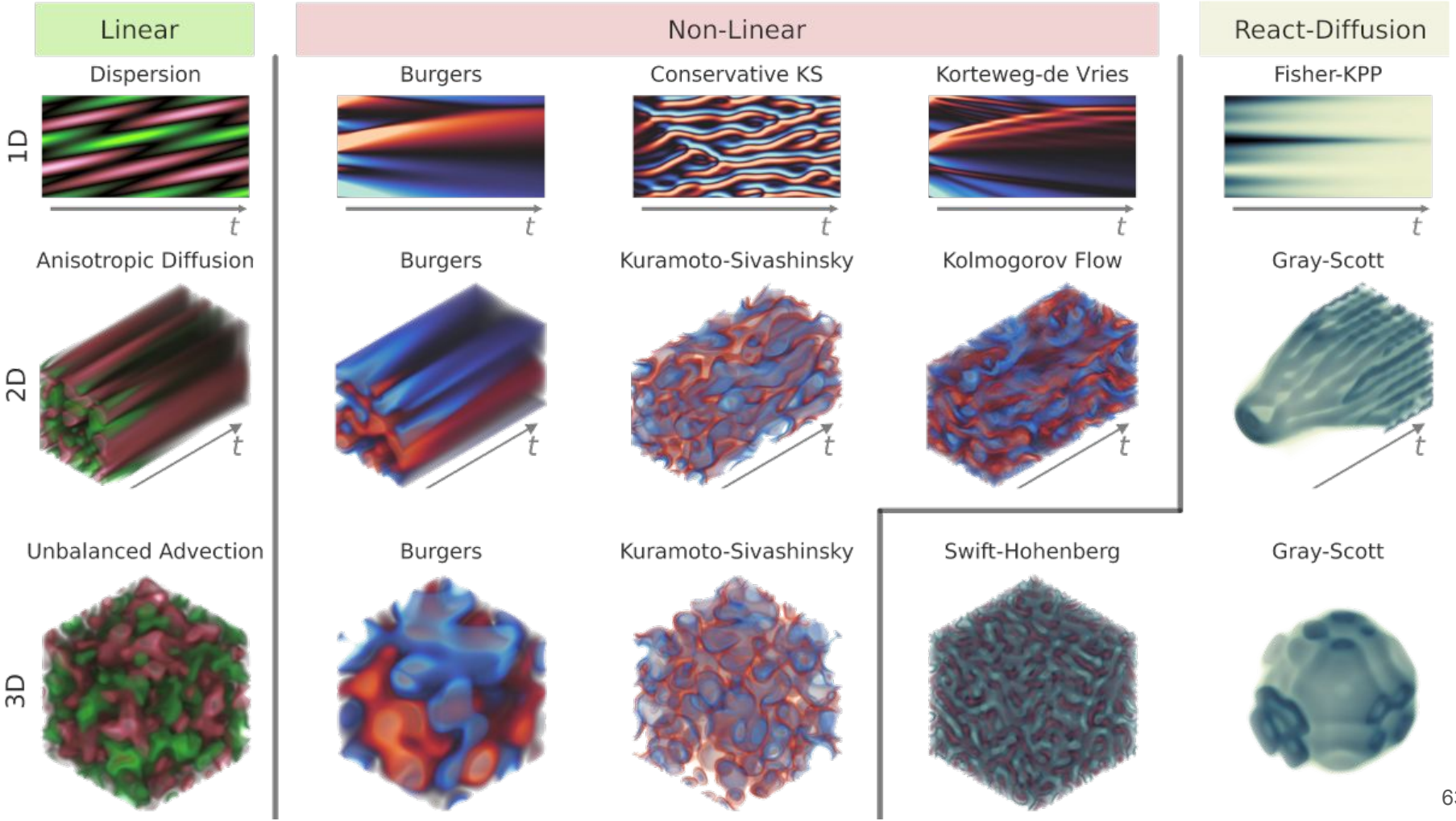


Network Size Ablation



A simple linear convolution

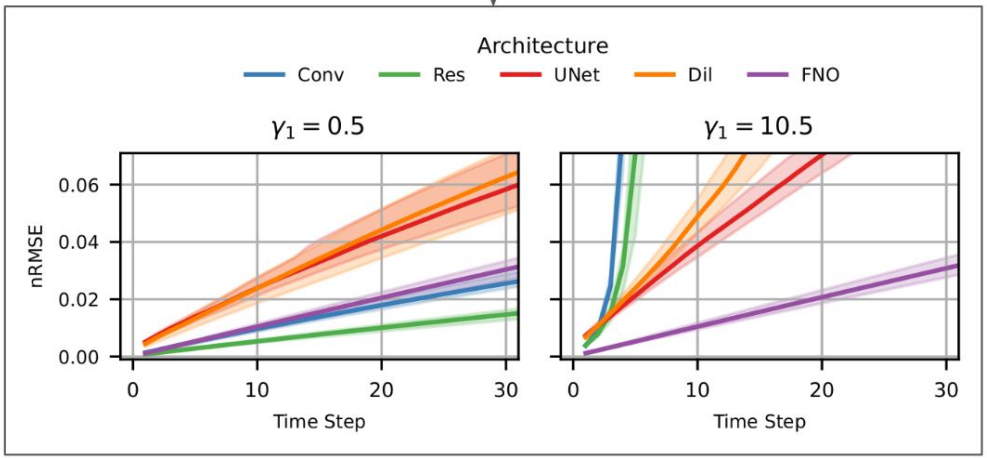



Linear
Non-Linear
React-Diffusion
1D
2D
3D
Dispersion
Burgers
Conservative KS
Korteweg-de Vries
Fisher-KPP
Anisotropic Diffusion
Burgers
Kuramoto-Sivashinsky
Kolmogorov Flow
Gray-Scott
Unbalanced Advection
Burgers
Kuramoto-Sivashinsky
Swift-Hohenberg
Gray-Scott
t

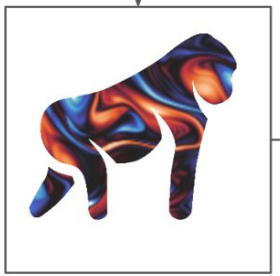
```python
CONFIGS = [
    {
        "scenario": "diff_adv",
        "task": "predict",
        "net": net,
        "train": "one",
        "advection_gamma": advection_gamma,
    }
    for net in [
        "Conv;34;10;relu",  # 31'757 params, 11 receptive field per direction
        "UNet;12;2;relu",  # 27'193 params, 29 receptive field per direction
        "Res;26;8;relu",  # 32'943 params, 16 receptive field per direction
        "FNO;12;18;4;gelu",  # 32'527 params, inf receptive field per direction
        "Dil;2;32;2;relu",  # 31'777 params, 20 receptive field per direction
    ]
    for advection_gamma in [0.5, 2.5]
]
```

pandas

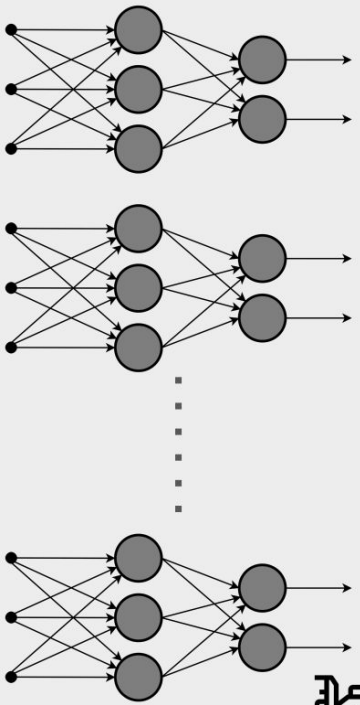
seaborn

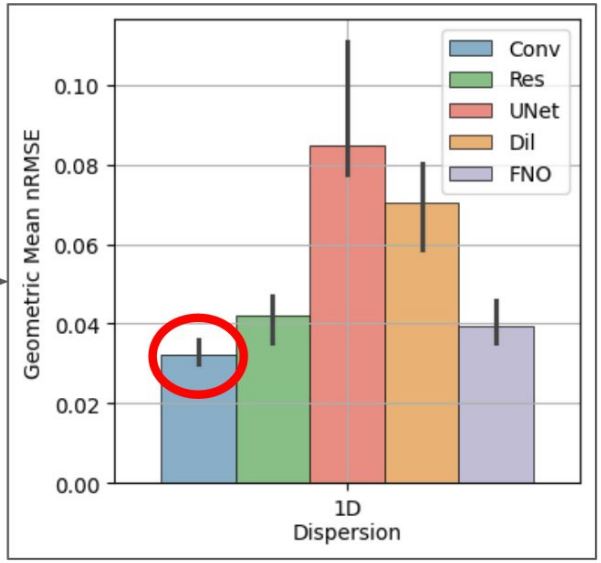


Study Definition



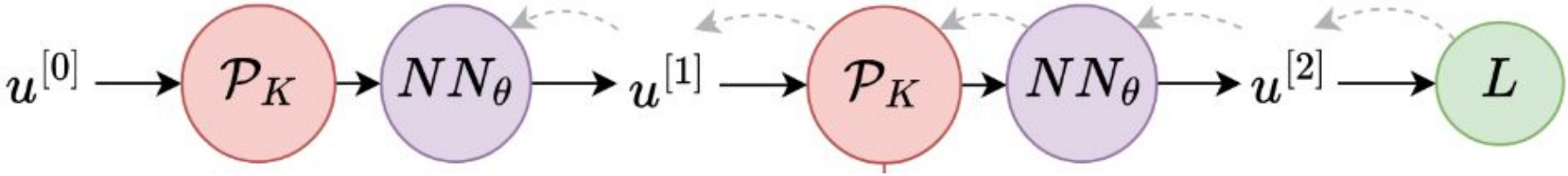
Multiple Random Seeds for init and minibatching

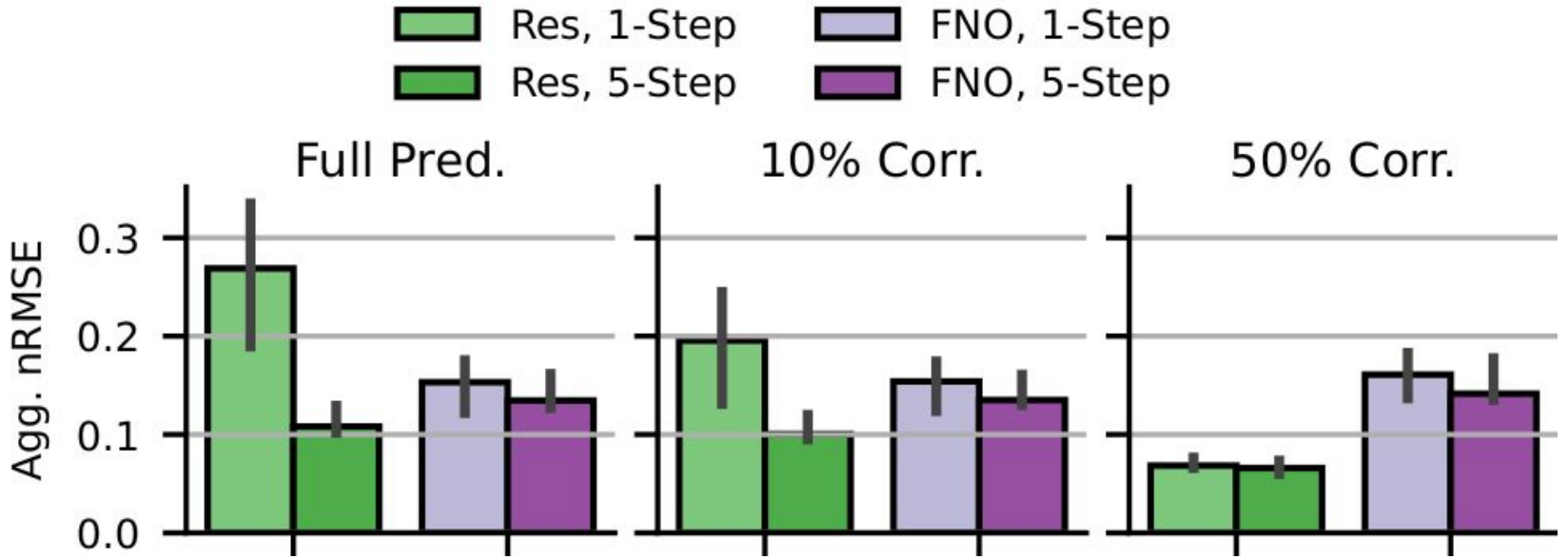




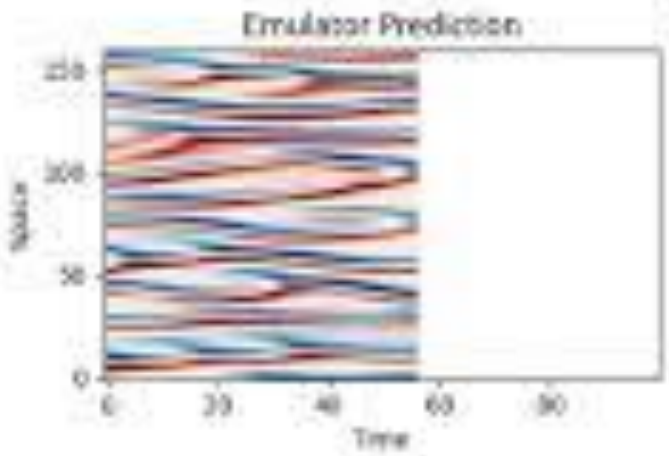
T-Test

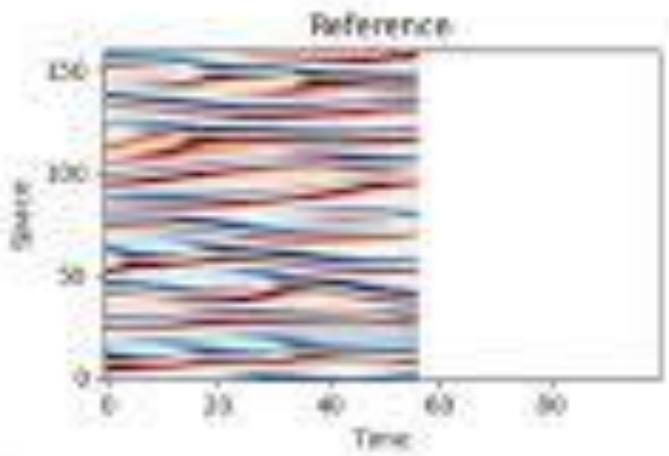
	Conv	Res	UNet	Dil	FNO
pvalue	1.0	1.1e-6	2.9e-9	3.7e-3	7.9e-8

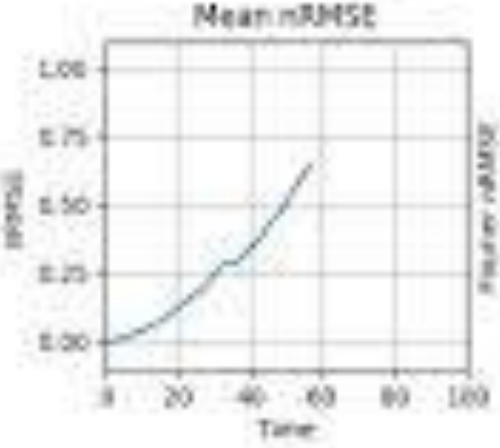

$u^{[0]}$
$\mathcal{P}_K$
NN_θ
$u^{[1]}$
$\mathcal{P}_K$
NN_θ
$u^{[2]}$
L

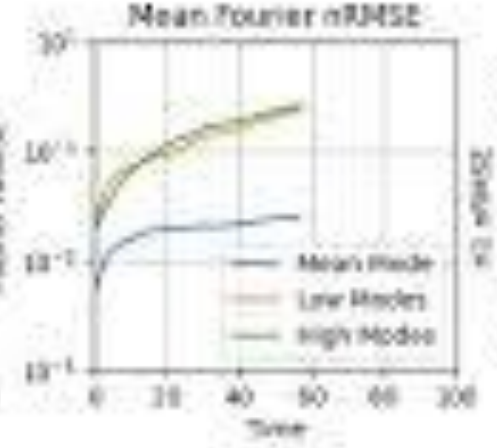

Res, 1-Step
Res, 5-Step
FNO, 1-Step
FNO, 5-Step
Full Pred.
10% Corr.
50% Corr.
Agg. nRMSE
0.3
0.2
0.1
0.0

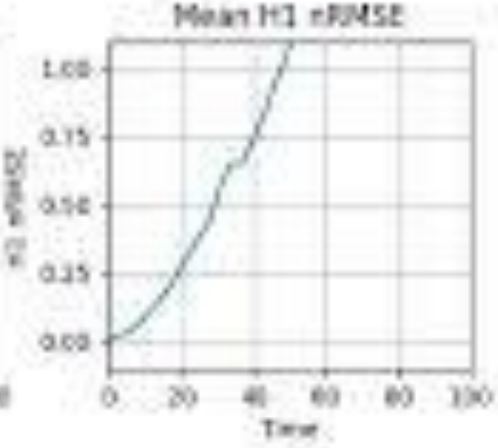
In [57]:

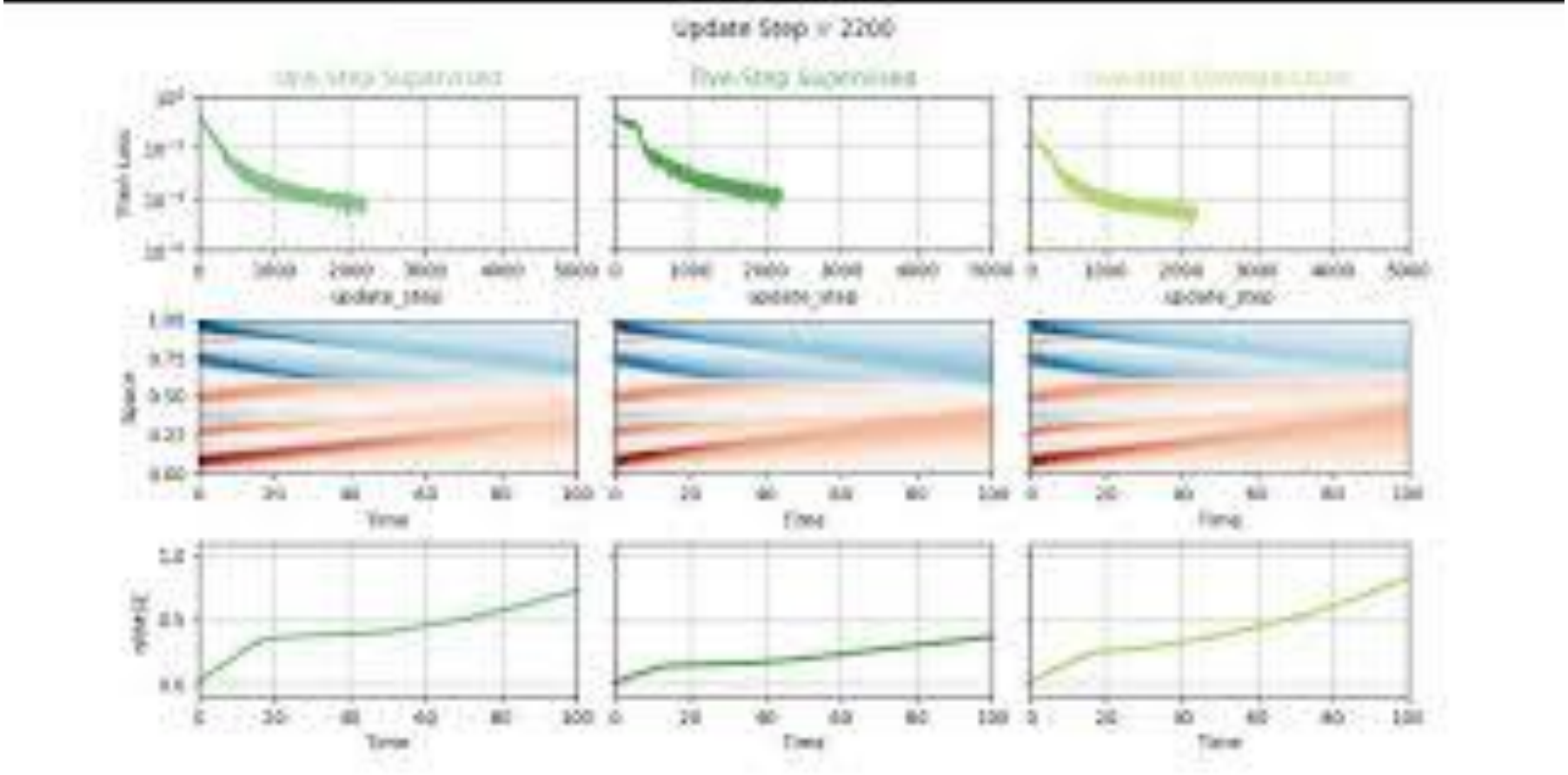

Emulator Prediction
Space
Time


Reference
Space
Time


Mean nRMSE
Time


Mean Fourier nRMSE
Mean Mode
Low Modes
High Modes
Time


Mean H1 nRMSE
Time


Update Step = 2200
One-Step Supervised
Five-Step Supervised
Train Loss
update_1000
update_3700
update_2000
Space
Time
Time
Time
alpha[1]
Time
Time
Time