APEBENCH

A Benchmark for Autoregressive Neural Emulators of PDEs

Felix Koehler, Simon Niedermayr,
Rüdiger Westermann, Nils Thuerey

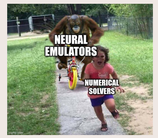
Autoregressive Emulation

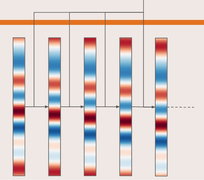
Approximate the time stepping behavior **of classical numerical methods** via a data-driven optimization.

Benchmark its long-term performance using **rollout metrics.**

$$\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x} \longrightarrow \mathcal{P}_h$$

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



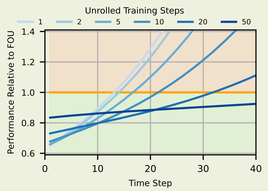


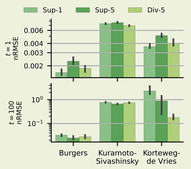
Unrolled Training Taxonomy

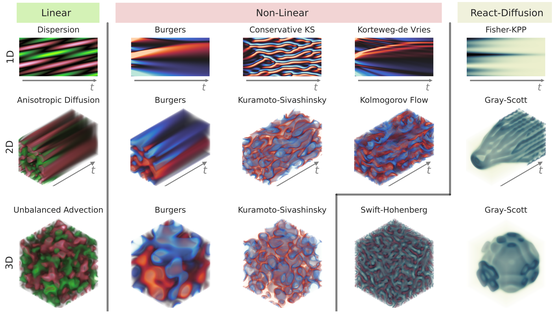
We unify many **unrolled training strategies.**

$$L(\theta) = \mathbb{E}_{u_{t_0}\propto\mathcal{D}_h}\left[\sum_{t=0}^{(T-B)}\sum_{b=1}^{B} w_t w_b\, l\left(f_\theta^{t+b}(u_h),\ \mathcal{P}_h^b(f_\theta^t(u_h))\right)\right]$$

One-Step Supervised: T=B=1
Unrolled Supervised: T=B
Simple Diverted Chain: T<B=1
General Diverted Chain: T<B<=2
… (gradient cuts for pushforward, exponential discounting...)







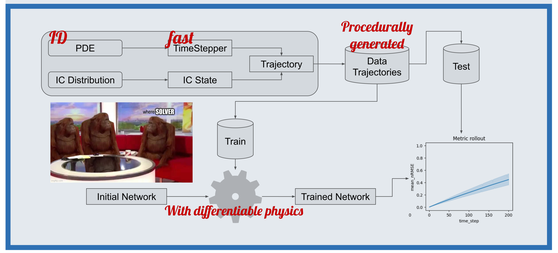
Workflow, IDs and Seed Statistics

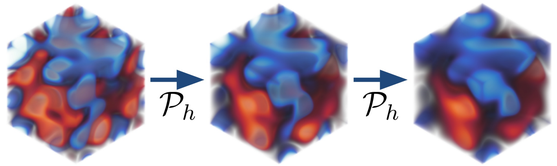
A reproducible emulator learning pipeline is encapsulated in an **APEBench scenario:**

- What is the reference simulator? (PDE and numerical scheme)
- What is the architecture of the neural emulator? (Predict or Correct?)
- How do emulator and simulator interact during training? (Unrolled steps, loss function, gradient cuts, optimizer, differentiable physics)
- What is the setting? (IC distribution, test rollout length, evaluation metrics)
- What is the statistical conclusion over multiple random seeds?

46 scenarios across 1D, 2D, and 3D:

- **Linear** (Advection, Diffusion, Dispersion, ...)
- **Nonlinear** (Burgers, KdV, KS,Navier-Stokes, ...)
- **Reaction-Diffusion** (Swift--Hohenberg, FisherKPP, Gray-Scot, ...)





Fourier-Spectral Solver

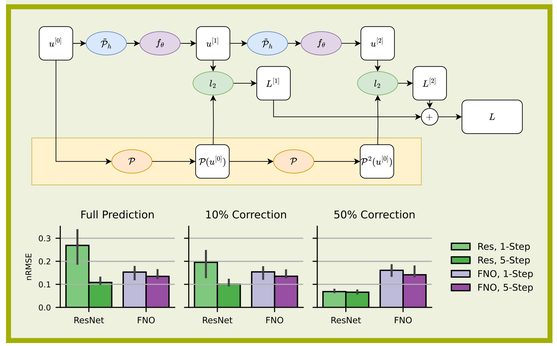
Procedural data generation enabled by an **efficient reference solver.**

$$\gamma_2 = 3 \wedge \delta_1 = -2 \wedge N = 160 \wedge D = 1$$
$$\implies \frac{\partial u}{\partial t} + 12.5\frac{1}{2}\frac{\partial u^2}{\partial x} = 0.059\frac{\partial^2 u}{\partial x^2}, x \in \Omega = (0,1), \Delta t = 0.001$$

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

Neural-Hybrid Emulation

Neural networks can correct outputs from coarse solvers. This requires **differentiable physics**, readily available within APEBench.

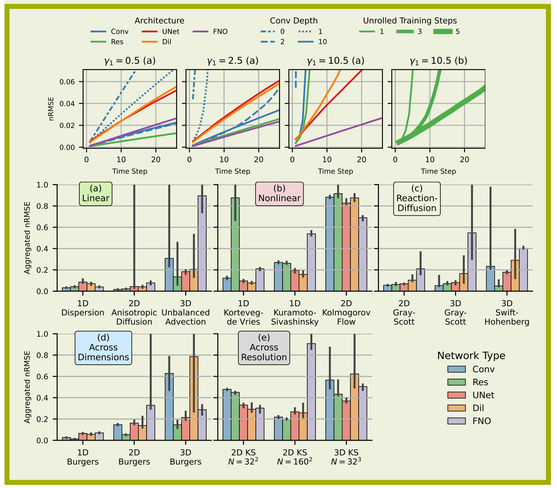


Emulators & Classical Simulators

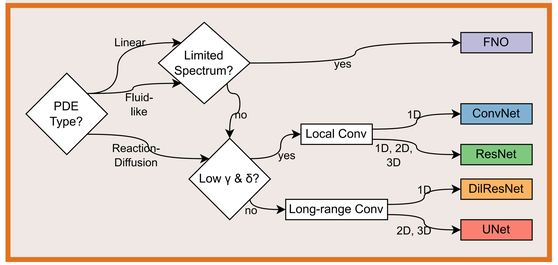
There is an inherent relation between **emulator architectures and classical numerical methods:**

- Convolution-based architectures inherit limits from finite differences
  -> They are bound by receptive field and CFL numbers
- FNO inherits limits from pseudo-spectral architectures
  -> Best for limited modes or clear spectral decay

The ideal architecture is often also the preferred numerical method



Architecture Decision Tree



Munich Center for Machine Learning