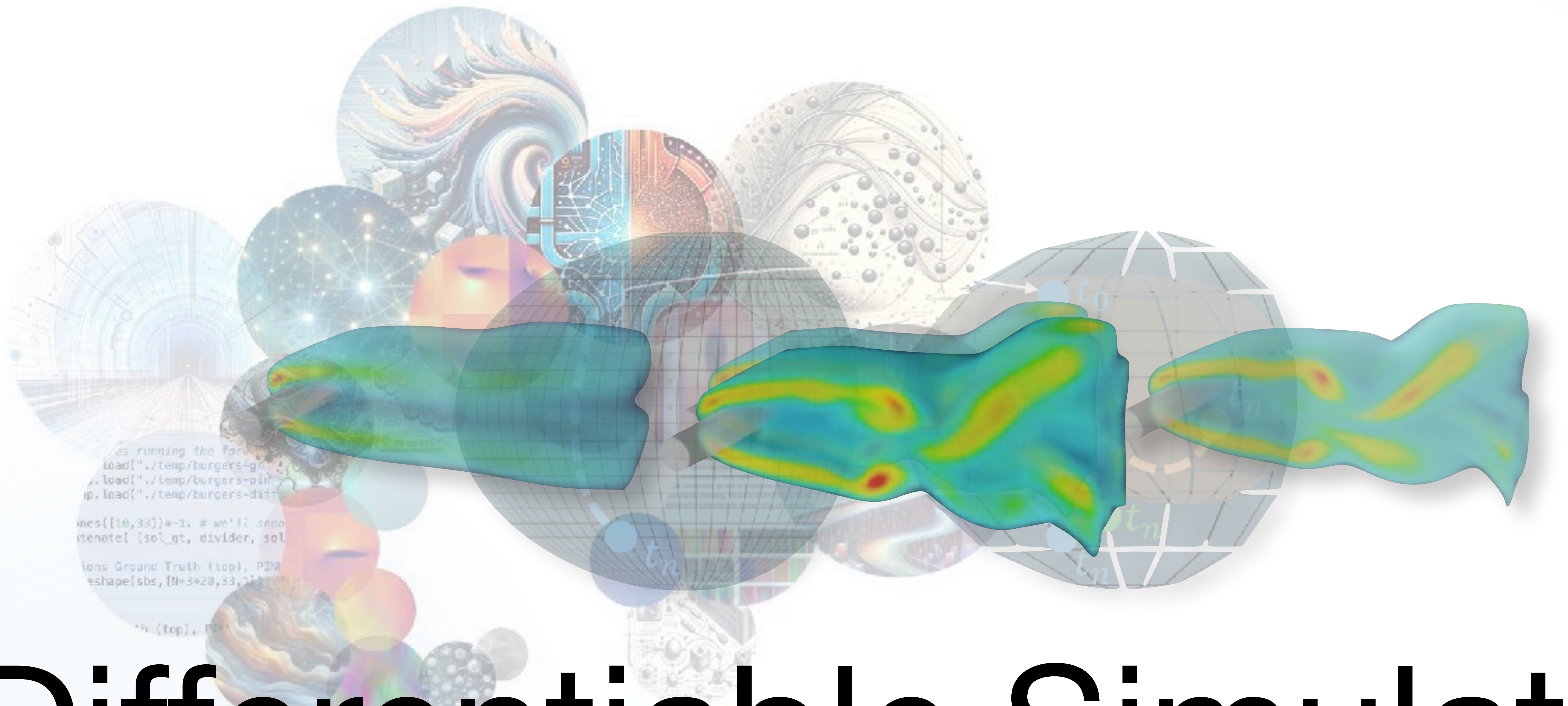


Differentiable Simulations

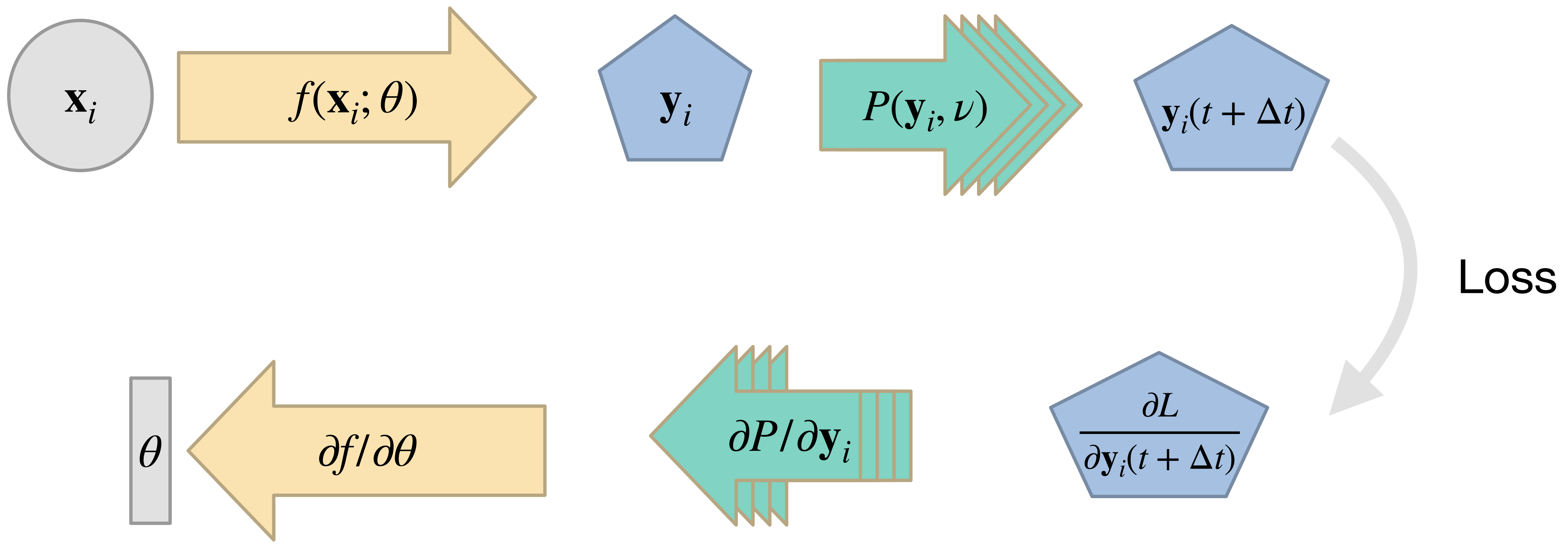
DEEP LEARNING FROM AND WITH NUMERICAL PDE SOLVERS (PART 2)

Contents

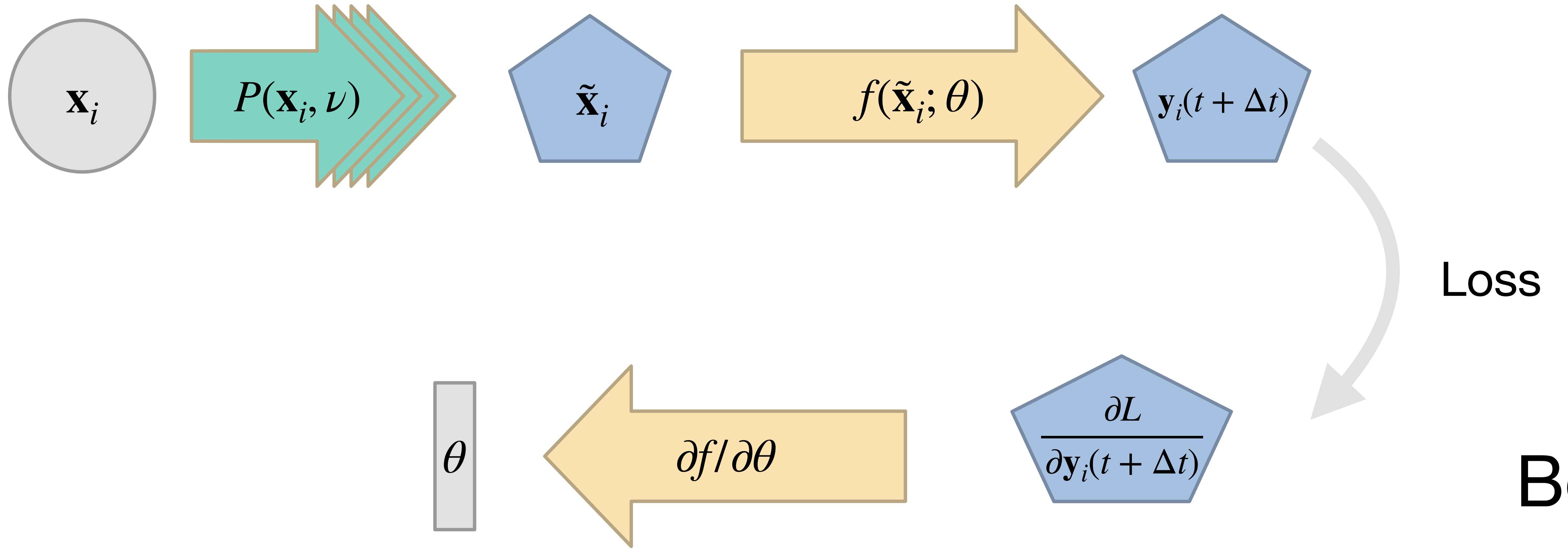
Differentiable Physics for Deep Learning

Starting with Combination Possibilities

Many different combinations beyond NN f → solver $\mathscr{P}$ → loss L possible



Mostly still a “residual”

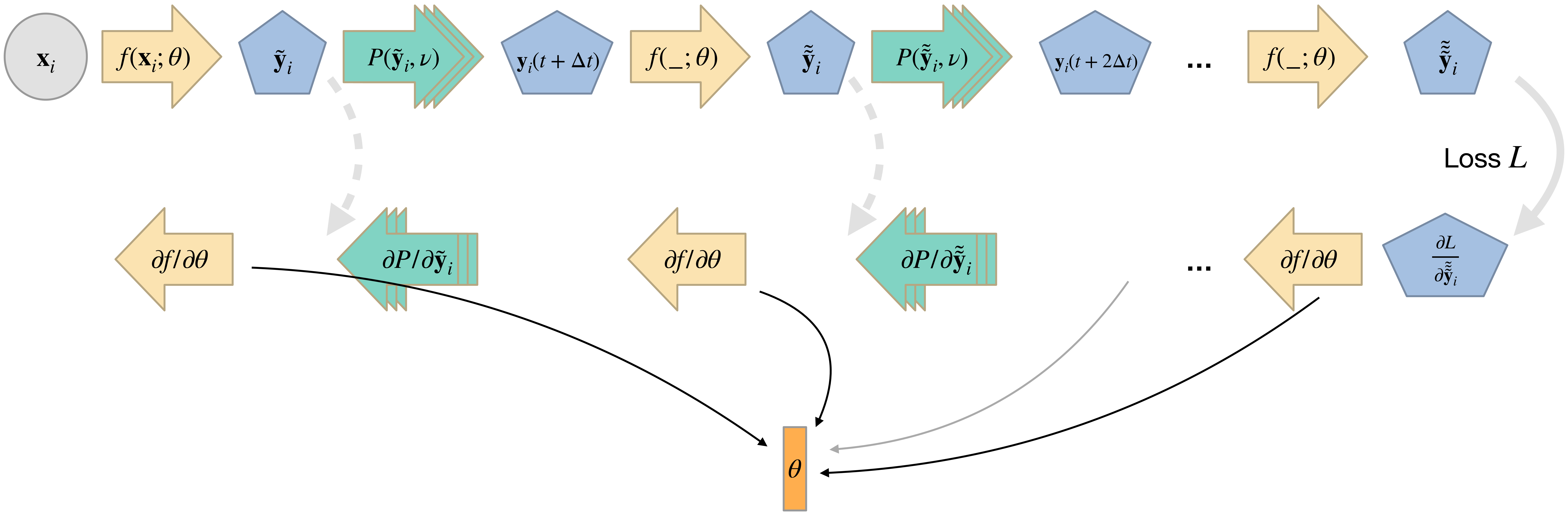


“On-the-fly” simulation

(No solver derivative needed)

Both not too interesting, better: freely combine…

Starting with Combination Possibilities



Re-cap Notation

- Ground truth denoted by $*$
- Learning goal: approximate $f^*(x) = y^*$
- Data set (x_i, y_i^*)
- Training: arg $\min_\theta |f(x;\theta) - y^*|_2^2$
- Physical quantity, such as flow field, denoted by $\mathbf{u}(t)$
- Bold to indicates vectors, e.g., $(\mathbf{x}_i, \mathbf{y}_i^*)$, the rest is equivalent…

An Attempt at Categorization

Target time series (*transient problems*). Distinguish main steps of the form:

- **Correction** task: $\mathbf{x}_{\text{new}} = \mathscr{P}(f(\mathbf{x};\theta),\nu)$
- **Prediction** task: $\mathbf{x}_{\text{new}} = f(\mathbf{x};\theta)$, i.e., $\mathscr{P} := Id$

Both could be applied autoregressively (== iteratively) over time

Denote with $g(\mathbf{x}) := \mathscr{P}(f(\mathbf{x};\theta),\nu)$ for correction, $g(\mathbf{x}) := Id(f(\mathbf{x};\theta))$ for prediction

Recursive application s times: $g^s(\mathbf{x})$

An Attempt at Categorization

Tasks as continuum:

- Perfect simulation → nothing left to do
- Pure **prediction** → no solver involved (only recurrent NN)
- **Correction** → hybrid simulator, numerics plus NN

Learning "Correctors"

Hybrid / Neural Solvers , Differentiable Physics (DP)

Neural network $f(\mathbf{x}_i;\theta)$ and simulator are evaluated multiple times, $\mathbf{u}(t_j) = g^s(\mathbf{u}(t_{j-1}))$ with $t_j = t + j\Delta t$

For s steps, with $\mathbf{x}_i = \mathbf{u}(t)$, $\mathbf{y}^*_{i,s} = \mathbf{u}^*(t + s\Delta t)$

Subtleties of correction alternatives, the interna of f in with NN component NN_θ :

Version 1: NN generates full state via $f(\mathbf{x};\theta) := \text{NN}_\theta(\mathbf{x})$

Version 2: Residual via operator "∘": $f(\mathbf{x};\theta) := \text{NN}_\theta(\mathbf{x}) \circ \mathbf{x}$, e.g. e.g. *additive* interaction: $\circ := +$

In comparison:

→ Version 1 can be more stable (no temporal "drift")

→ Version 2 typically much simpler task for NN

Learning “Correctors”

Hybrid / Neural Solvers , Differentiable Physics (DP)

Modified state $\mathbf{u}$ at later time influenced by previously modified steps

Solver alone does not immediately produce the correct answer: $\mathbf{u}^*(t + s\Delta t) \neq \mathscr{P}^s(\mathbf{u}(t))$

Outputs differ from input $\mathbf{u}(t + s\Delta t) \neq g^s(\mathbf{u}(t))$, and from reference $\mathbf{u}^*(t + s\Delta t) \neq g^s(\mathbf{u}(t))$

Minimization for all steps s of $\mathscr{L} := \sum_s \left| \mathbf{u}^*(t + s\Delta t) - g^s(\mathbf{u}(t)) \right|^2$

Requires back propagation through all s steps of physics solver and NN!

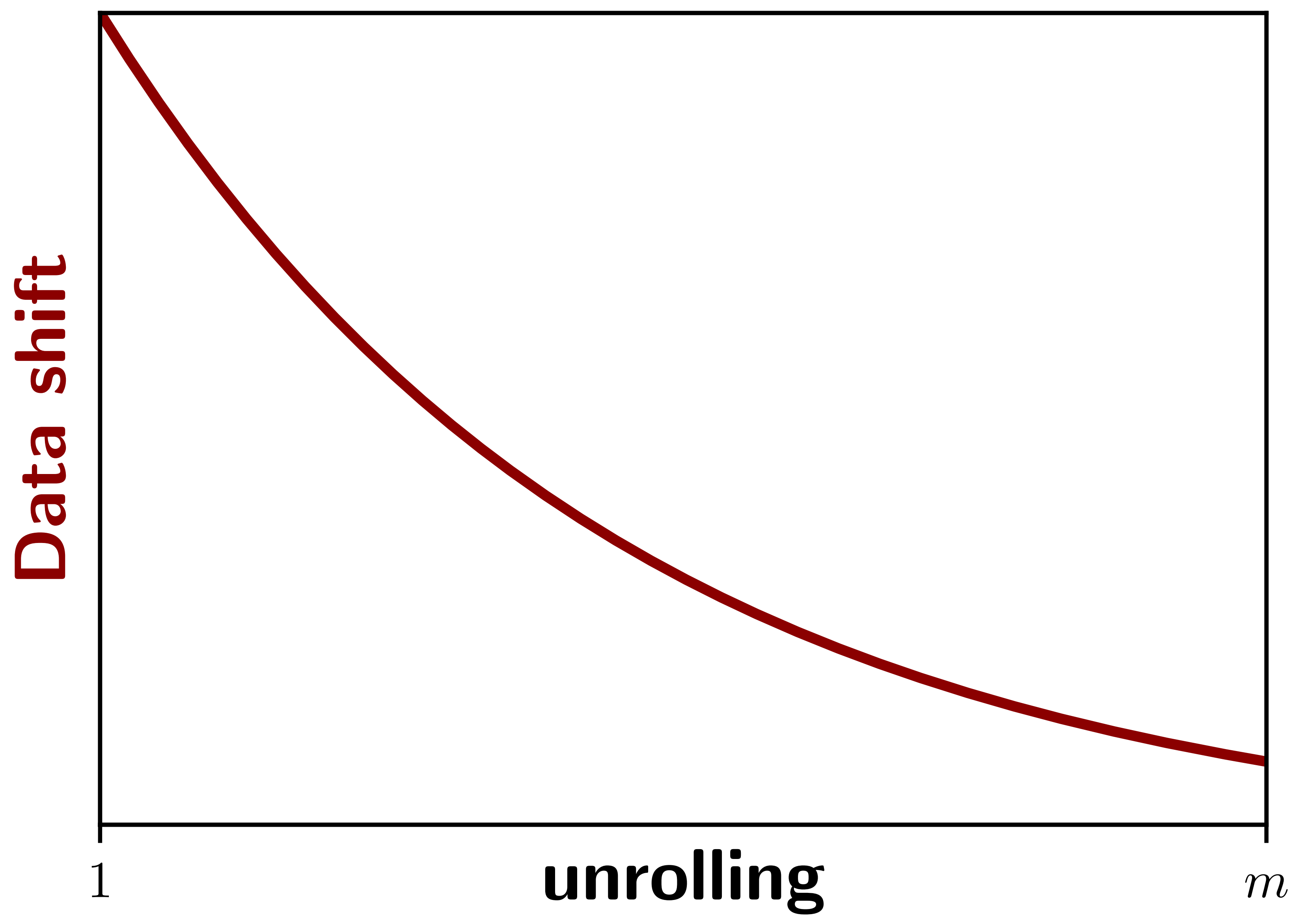
[Note: Similar to “regular” recurrent neural network training, but additionally involves PDE solvers]

Data Shift

Changing states: classic *data shift* problem

Distribution of inputs changes, esp. while training!

→ The more *unrolling* the better? “Case closed”?



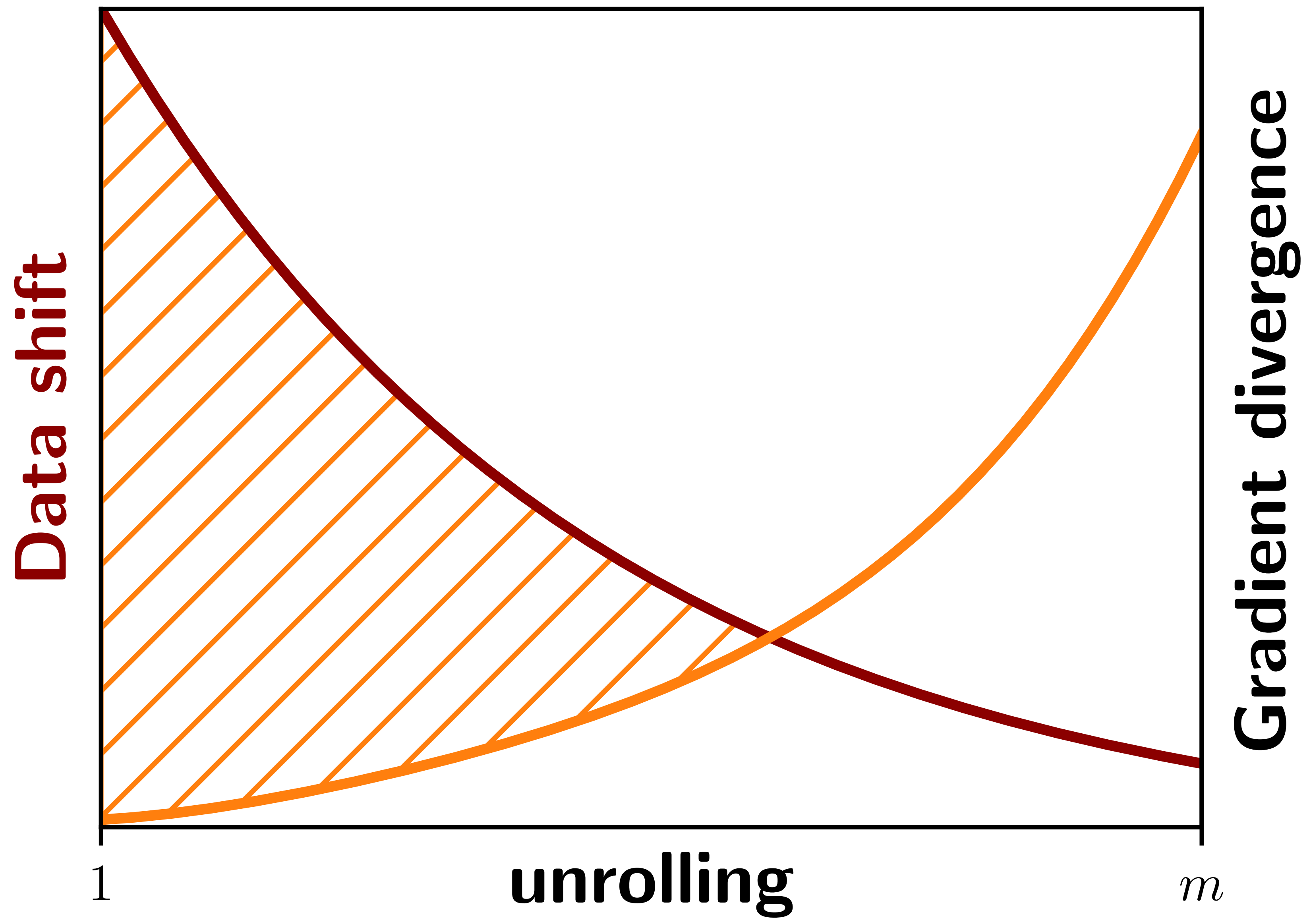
Recurrent Training

Not quite 😥 , unrolling introduces problems:

Recurrent NN gradients can diverge

One step training:

$$\frac{\partial \mathscr{L}^1}{\partial f_\theta^1}\frac{\partial f_\theta^1}{\partial \theta}$$



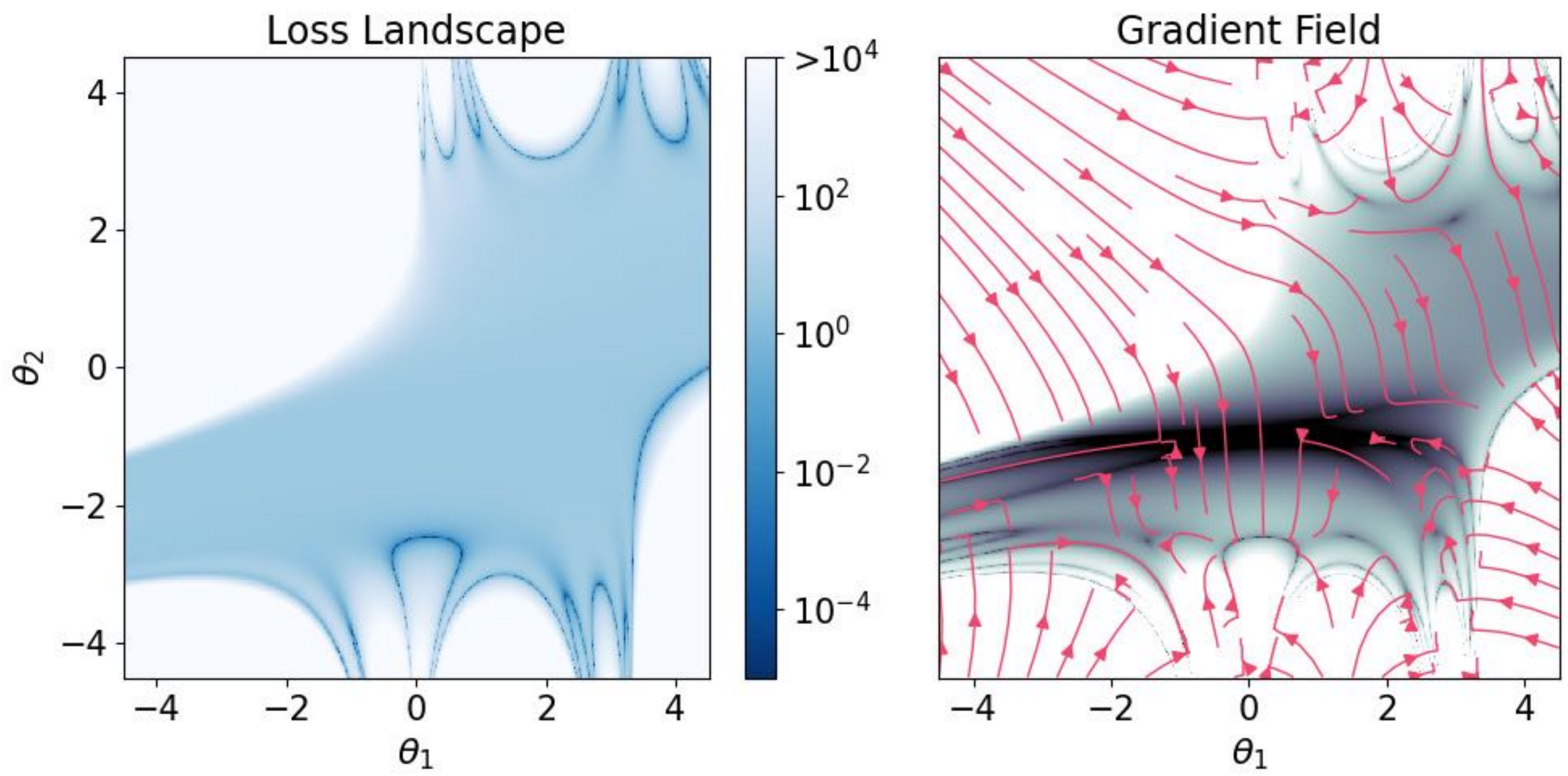
Unrolled training:

$$\sum_s \sum_{B=1}^{s} \frac{\partial \mathscr{L}^s}{\partial g^s}\frac{\partial g^s}{\partial g^B}\frac{\partial g^B}{\partial \theta}$$

Loss Landscapes

Unrolling increases *complexity* of loss landscape and gradients

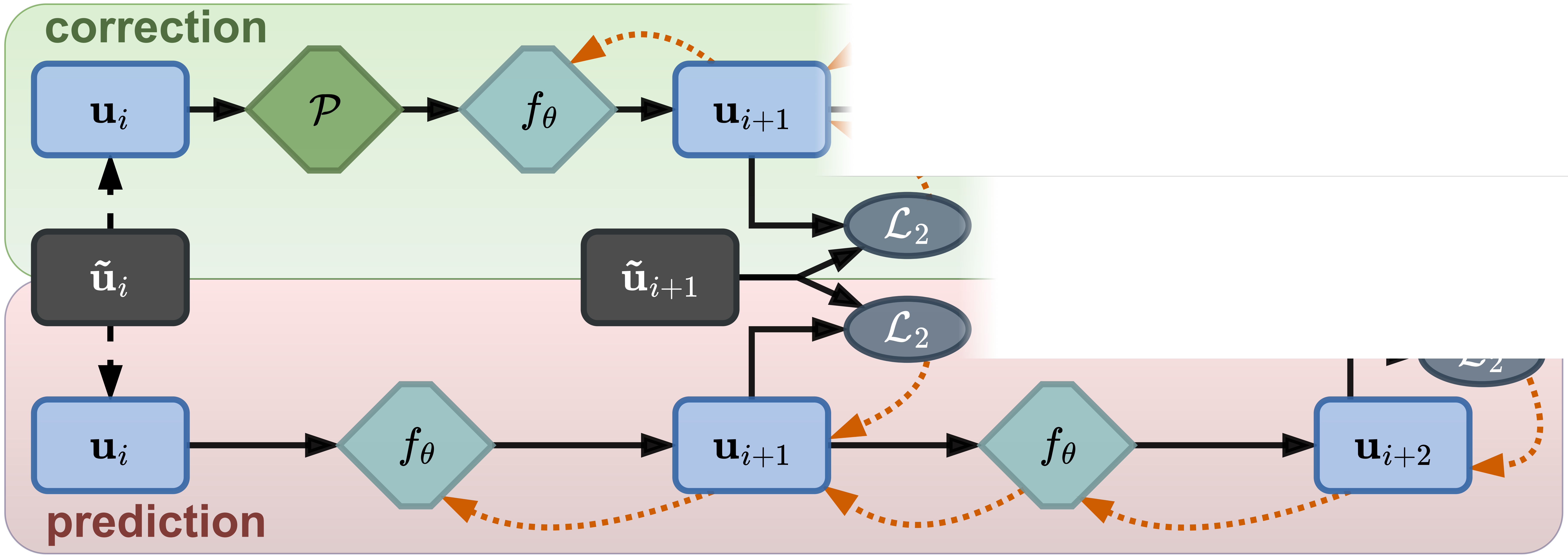
Toy example with polynomial $\mathscr{P}(x, \theta) = -\theta_1 x^2 + \theta_2 x$:



Schnell et. al: Stabilizing Backpropagation through Time for Learning Complex Physics

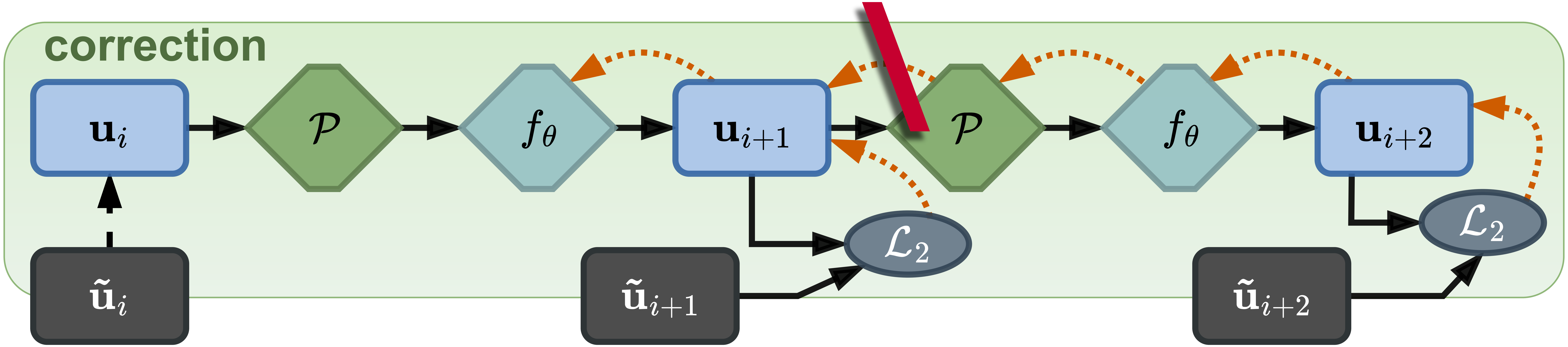
Prediction and Correction Tasks

Gradient flow visualized for states u_i:



No-gradient (NOG) Training

Alternative: train without gradient from simulator



Disentangling Contributions

How much does each part matter?

Open question so far, how much does each component contribute:

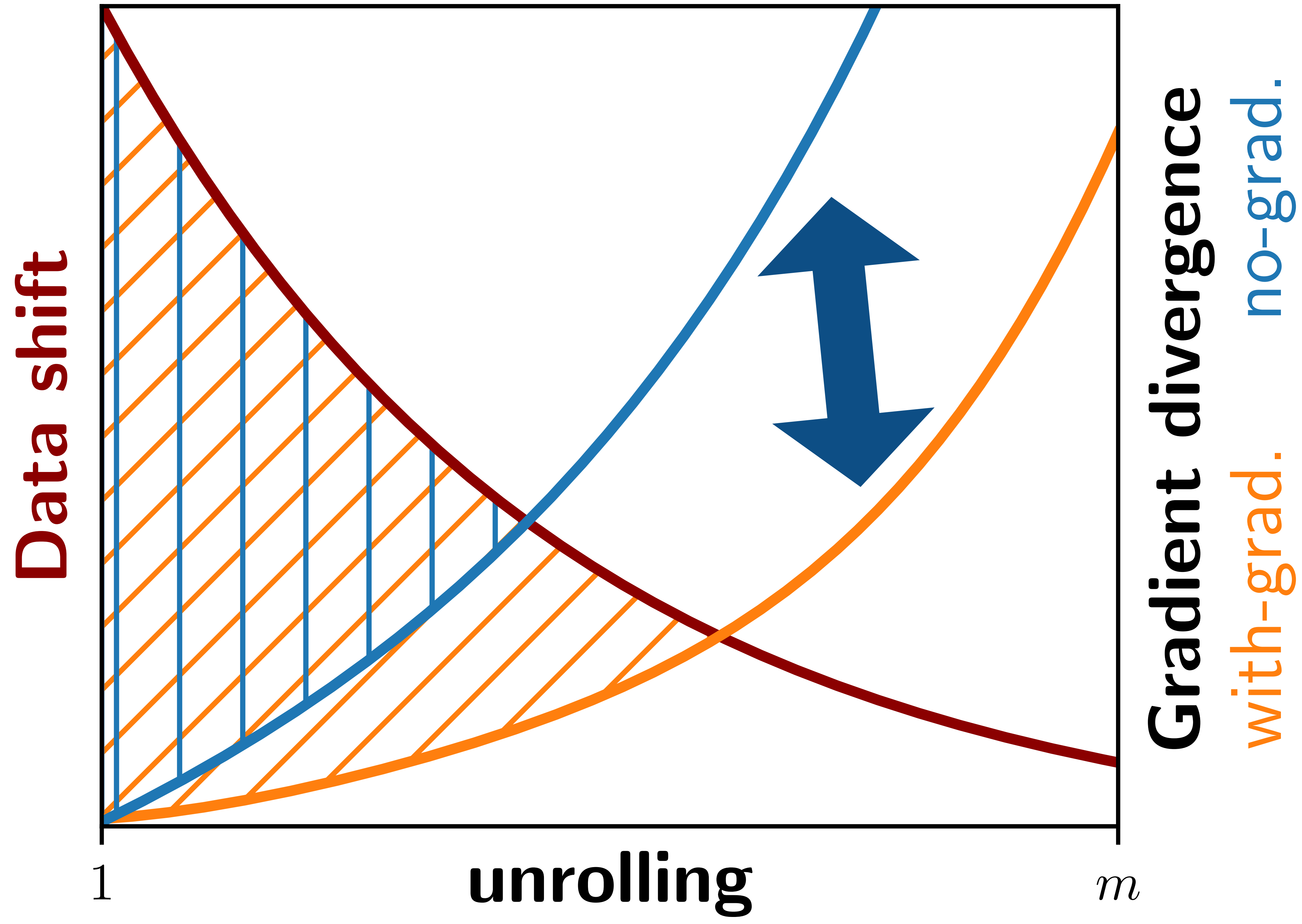
- (0) Basis: pure neural network prediction
- (1) Add *non-differentiable solver* (correction)
- (2) Apply *unrolling* (data-shift)
- (3) *Backpropagate* gradients (“correct” gradients)

No-gradient (NOG) Training

Resulting training gradient:

$$\sum_s \frac{\partial \mathscr{L}_2^s}{\partial f_\theta^s} \frac{\partial f_\theta^s}{\partial \theta}$$

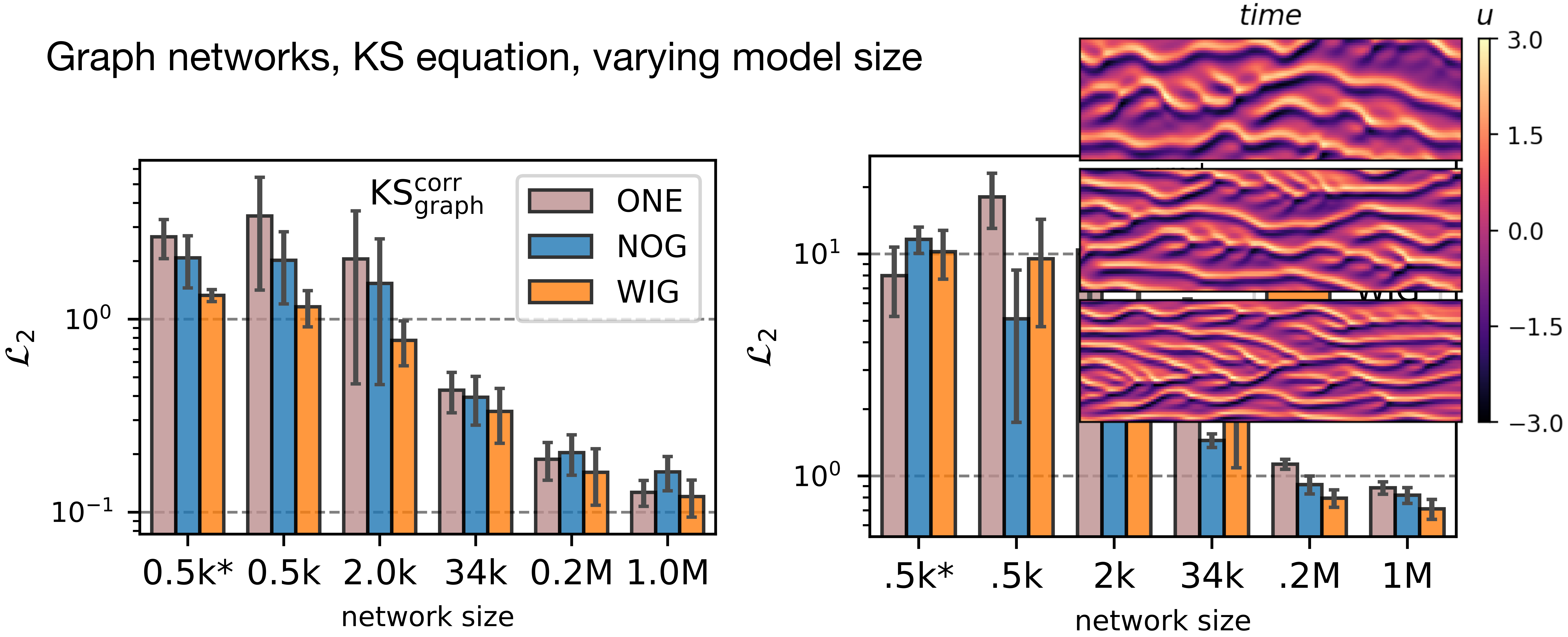
"Worse, but can it provide benefits?"



List et. al: How Temporal Unrolling Supports Neural Physics Simulators

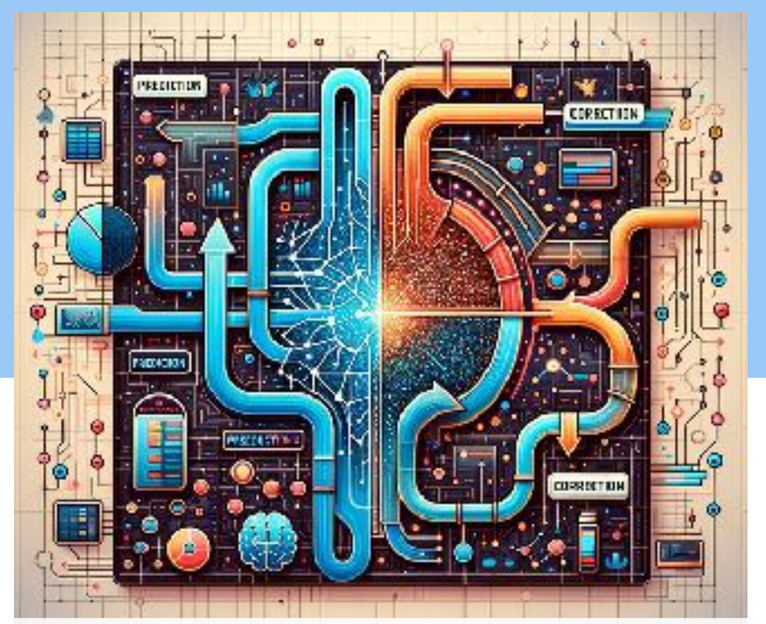
Results

Graph networks, KS equation, varying model size



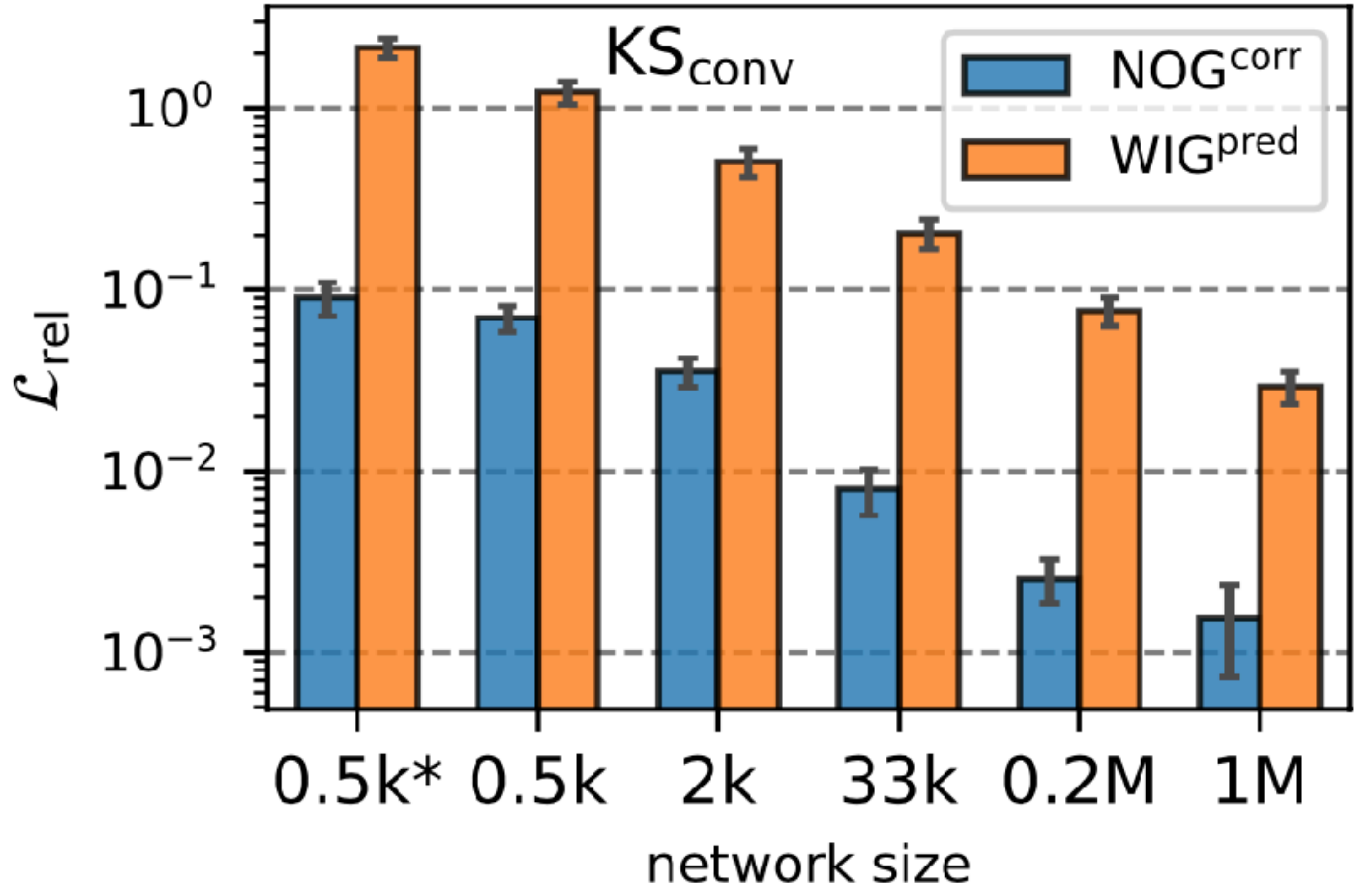
List et. al: How Temporal Unrolling Supports Neural Physics Simulators

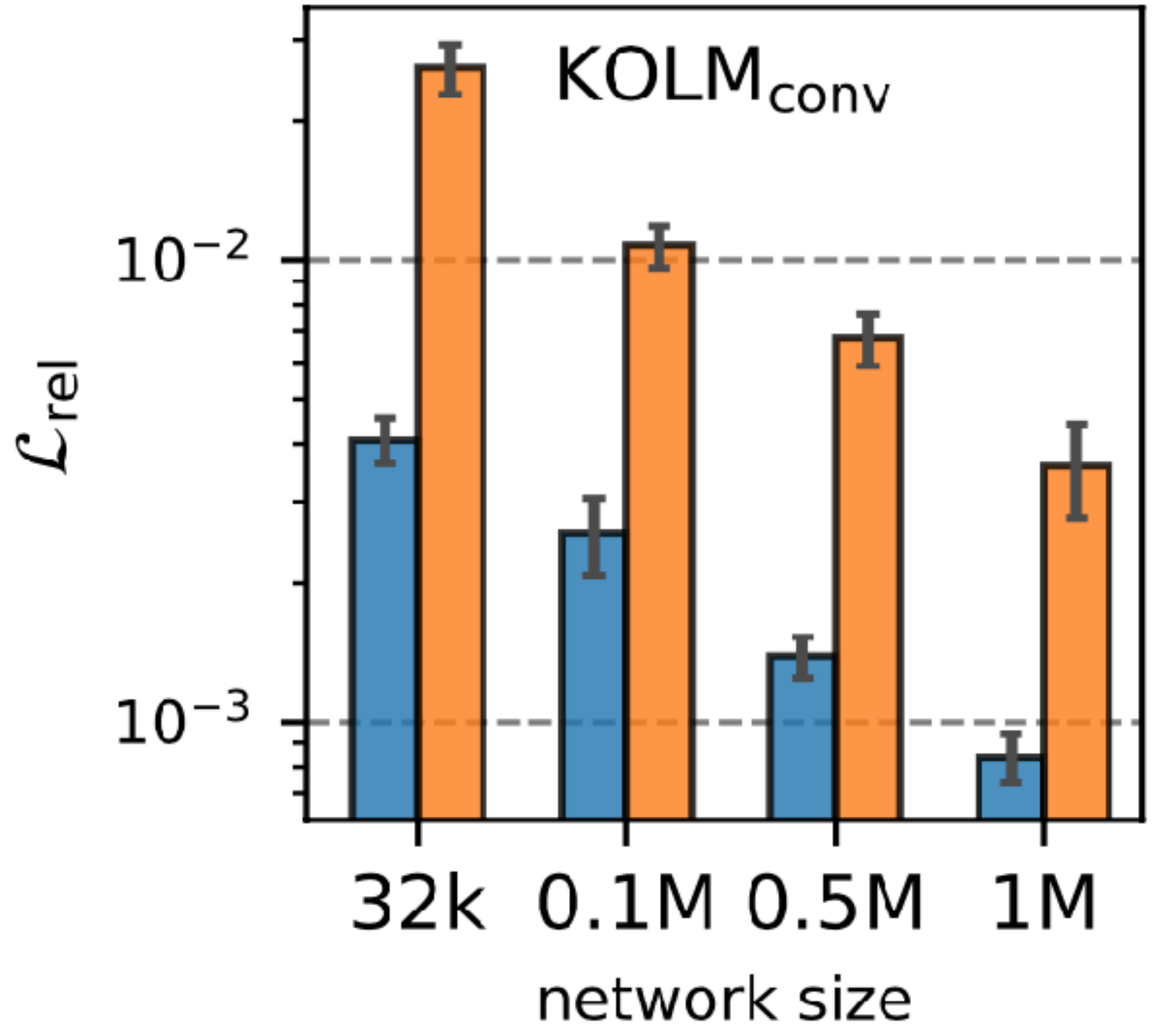
How much to gain in practice?



Central advantage: replace *prediction* task by *correction*

→ Improve accuracy by more than 10x with an *identical* network, but slightly higher compute cost





List et. al: How Temporal Unrolling Supports Neural Physics Simulators

Disentangling Contributions

How much does each part matter?

Open question so far, how much does each component contribute:

- (0) Basis: pure neural network prediction
- (1) Add *non-differentiable solver* (correction) → **10x**
- (2) Apply *unrolling* (data-shift) → **33%**
- (3) *Backpropagate* gradients (“correct” gradients) → **15%**

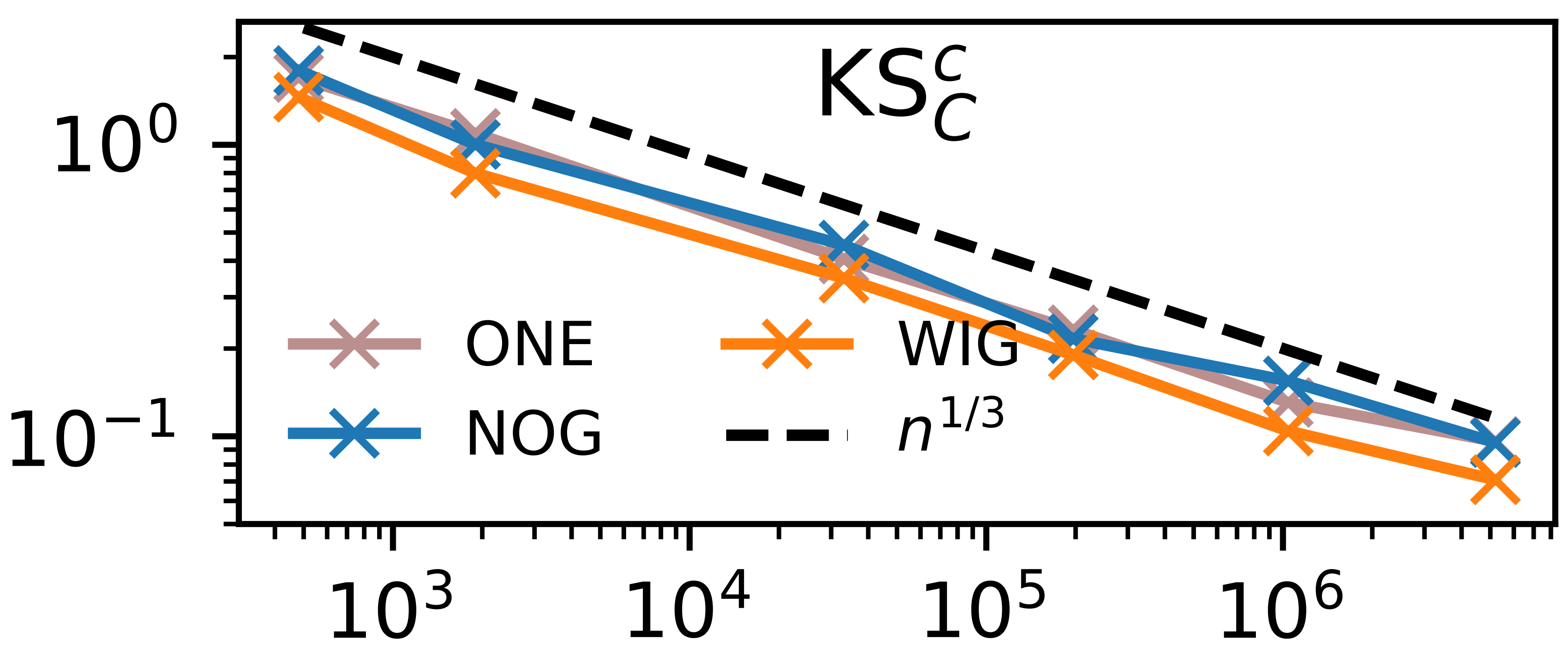
In total:
More than **15x** on average

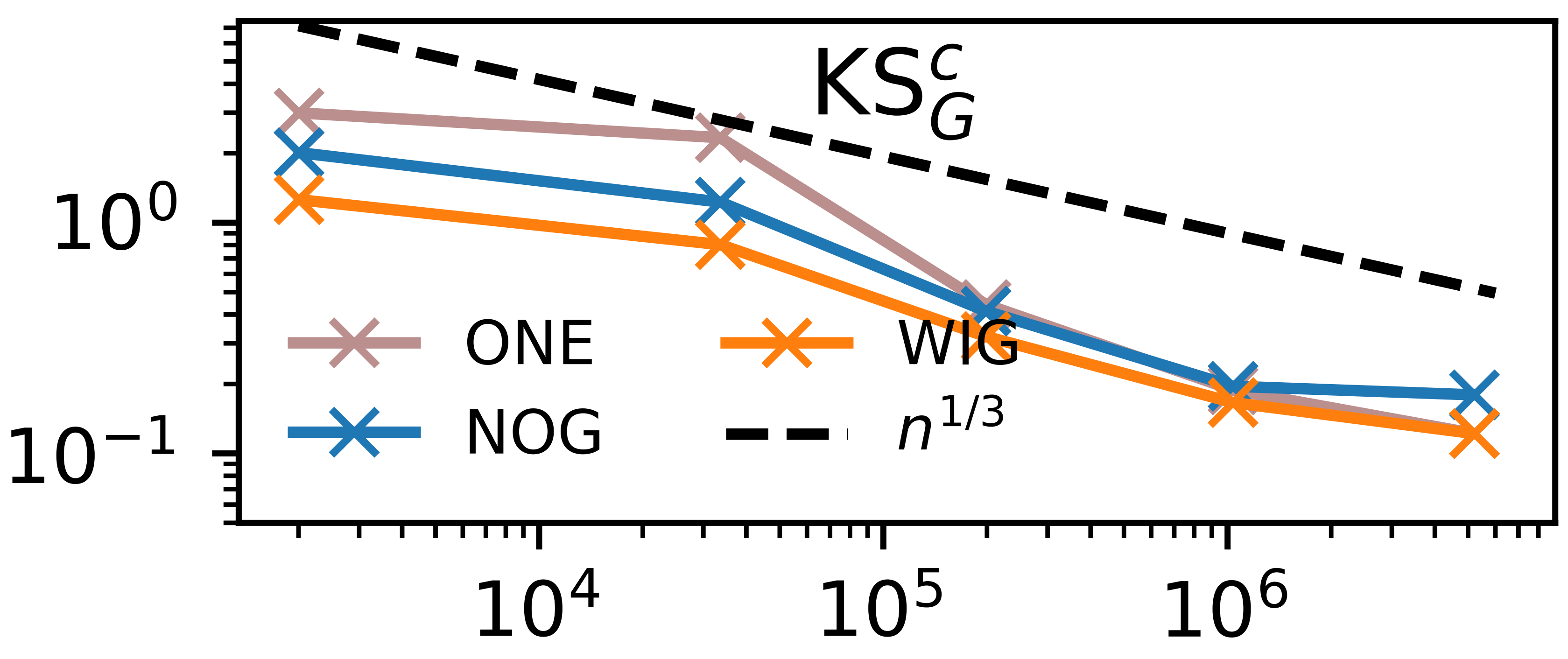
Outlook - Scaling

In general: suboptimal for NNs , scales with ca. 1/3

(→ *"It's good to keep NN small!"*)

Correction tasks: let P handle large scale data shift

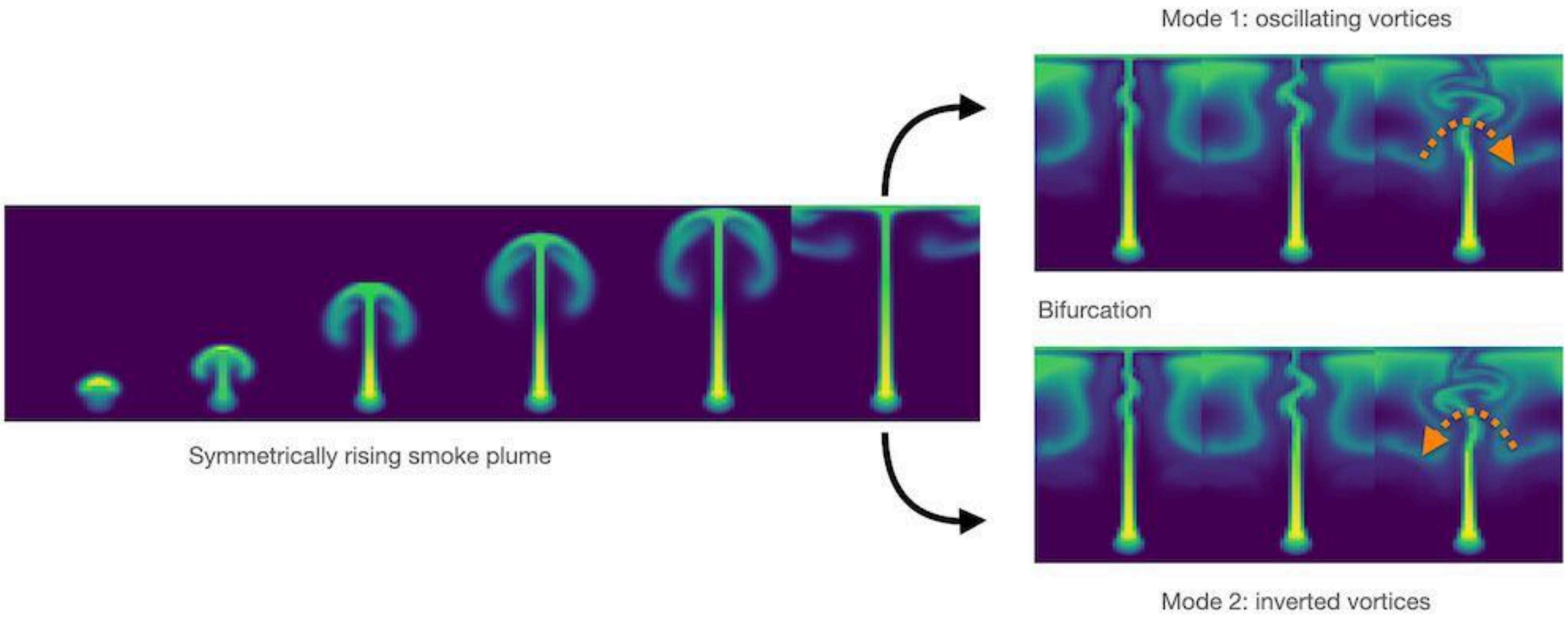




List et. al: How Temporal Unrolling Supports Neural Physics Simulators

Intuition - Multi-modal Problems

Example - Flow Bifurcation



Intuition - Multi-modal Problems

(Advantages of full gradient over Supervised Training)

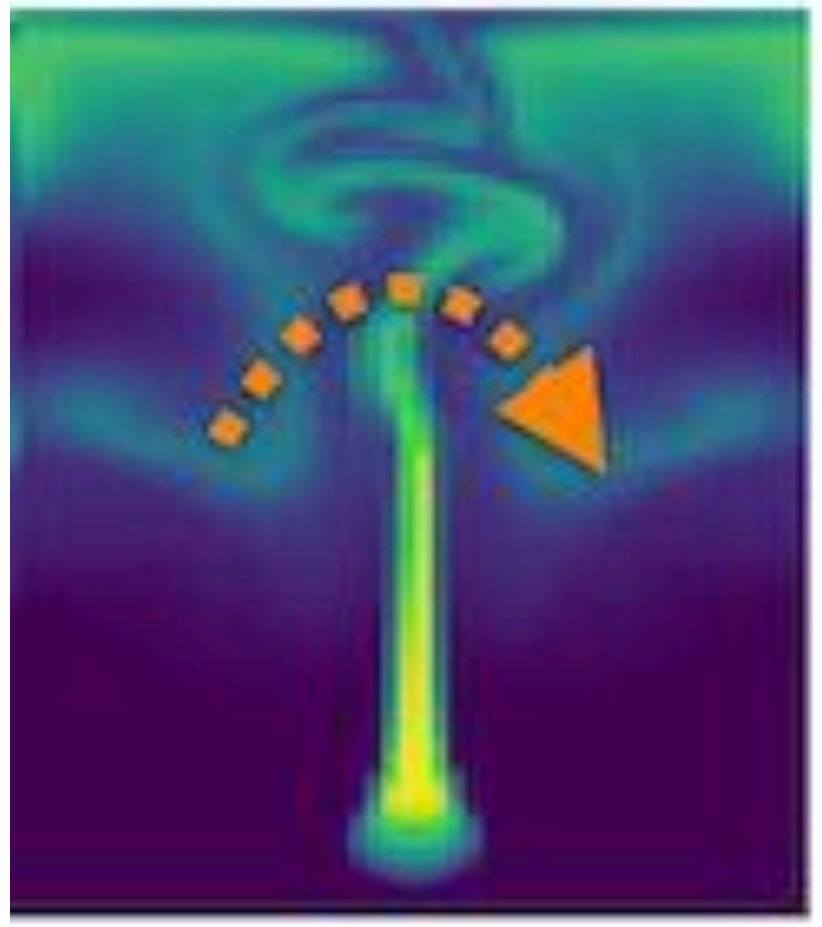
Strongly varying solutions for small changes of the input

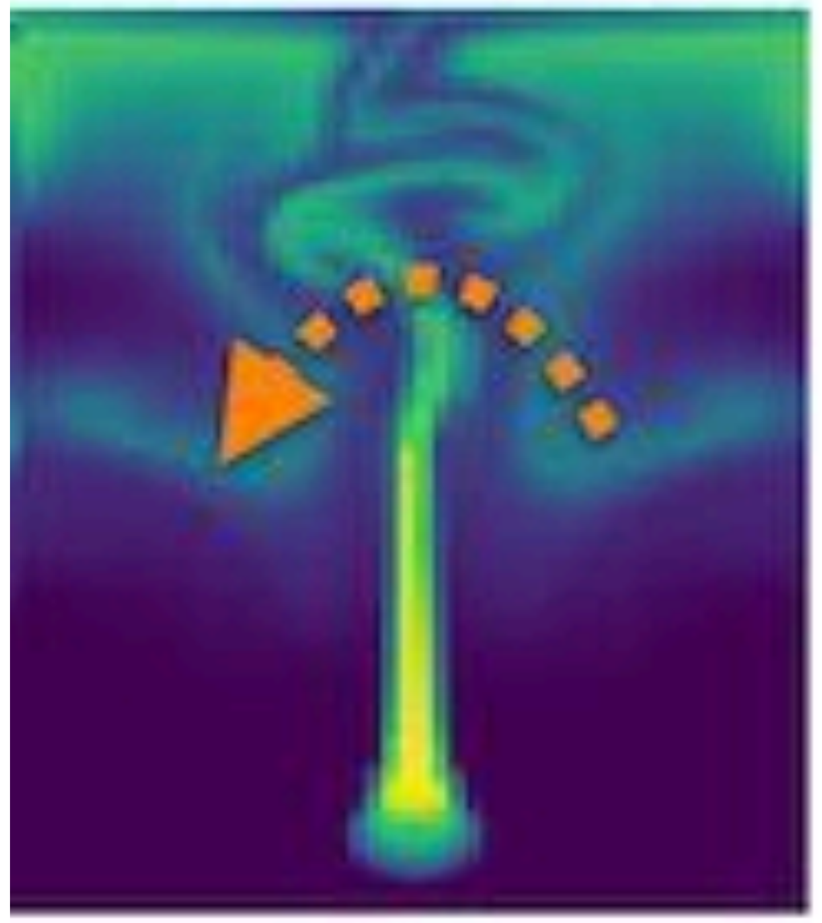
Extreme example: *super-resolution problems*

Cause undesirable averaging for supervised training

Differentiability can yield gradients for current state

⇨ No averaging, single gradient provided





Summary so far

Neural PDE Solvers

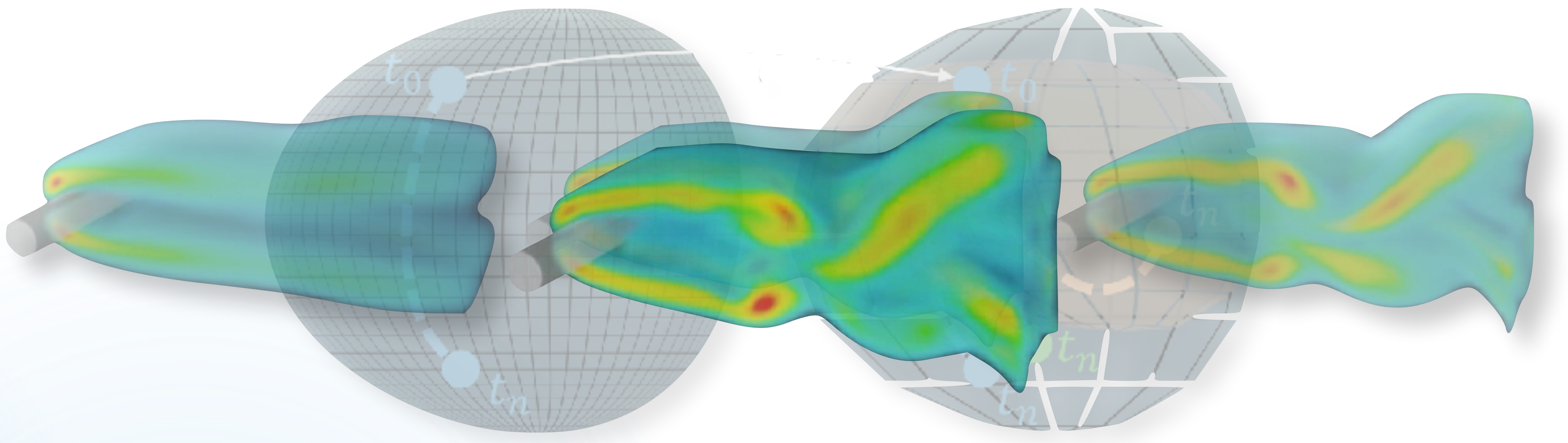
Most generic form of coupling: *arbitrary* combination and repetition

⇨ Neural network now works *alongside* $\mathscr{P}$ to produce the right answer

Reference states $\mathbf{y}^*$ can be from external & non-differentiable solver

Could also be obtained from experiments

[ScalarFlow: A Large-scale Data-set of Real-world Flows, 2019]



Differentiable Simulations

DEEP LEARNING FROM AND WITH NUMERICAL PDE SOLVERS (PART 3)

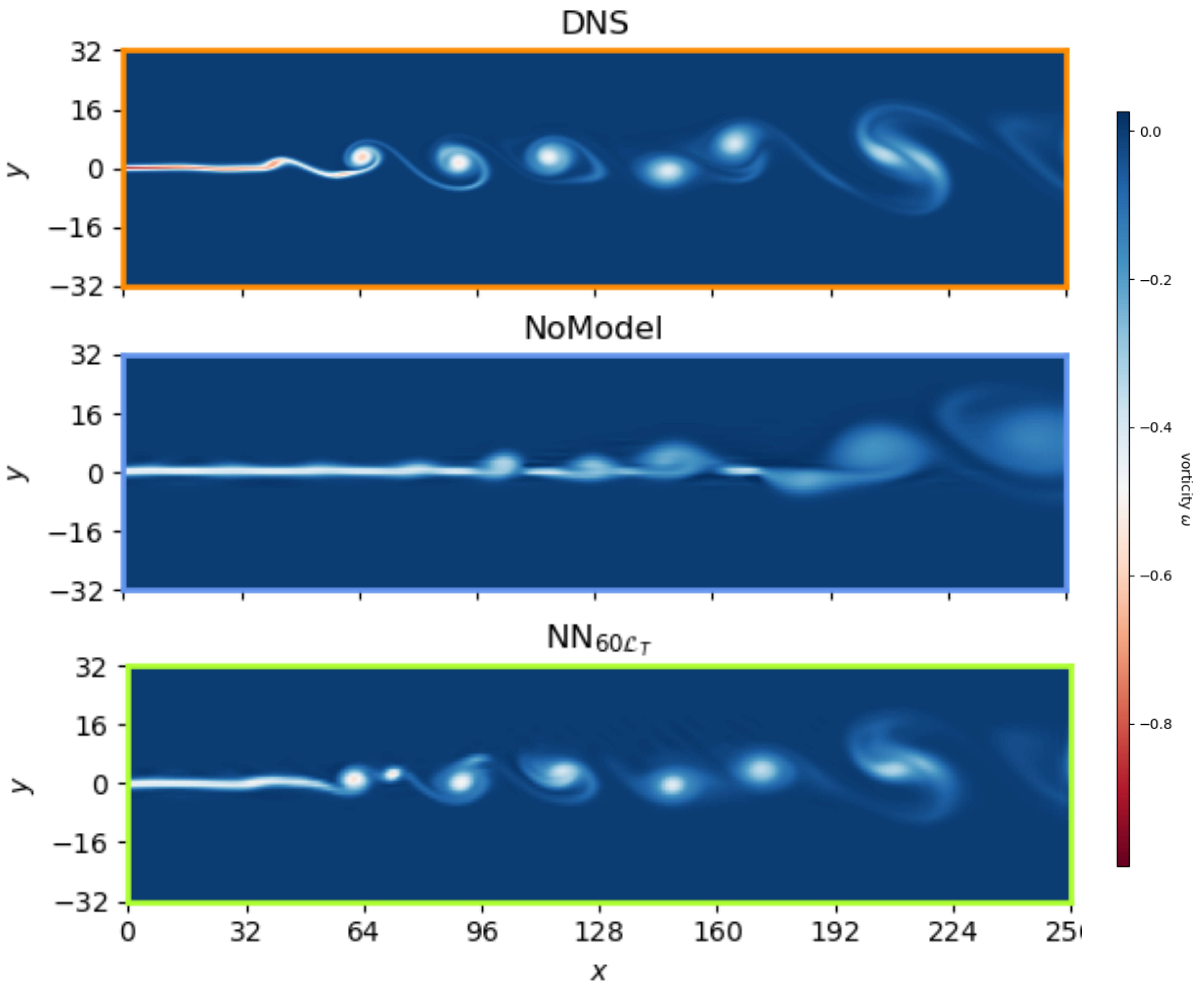
Contents

Differentiable Physics Example 1

List et. al: Learned Turbulence Modelling with Differentiable Fluid Solvers

Turbulence: Spatial Mixing Layer

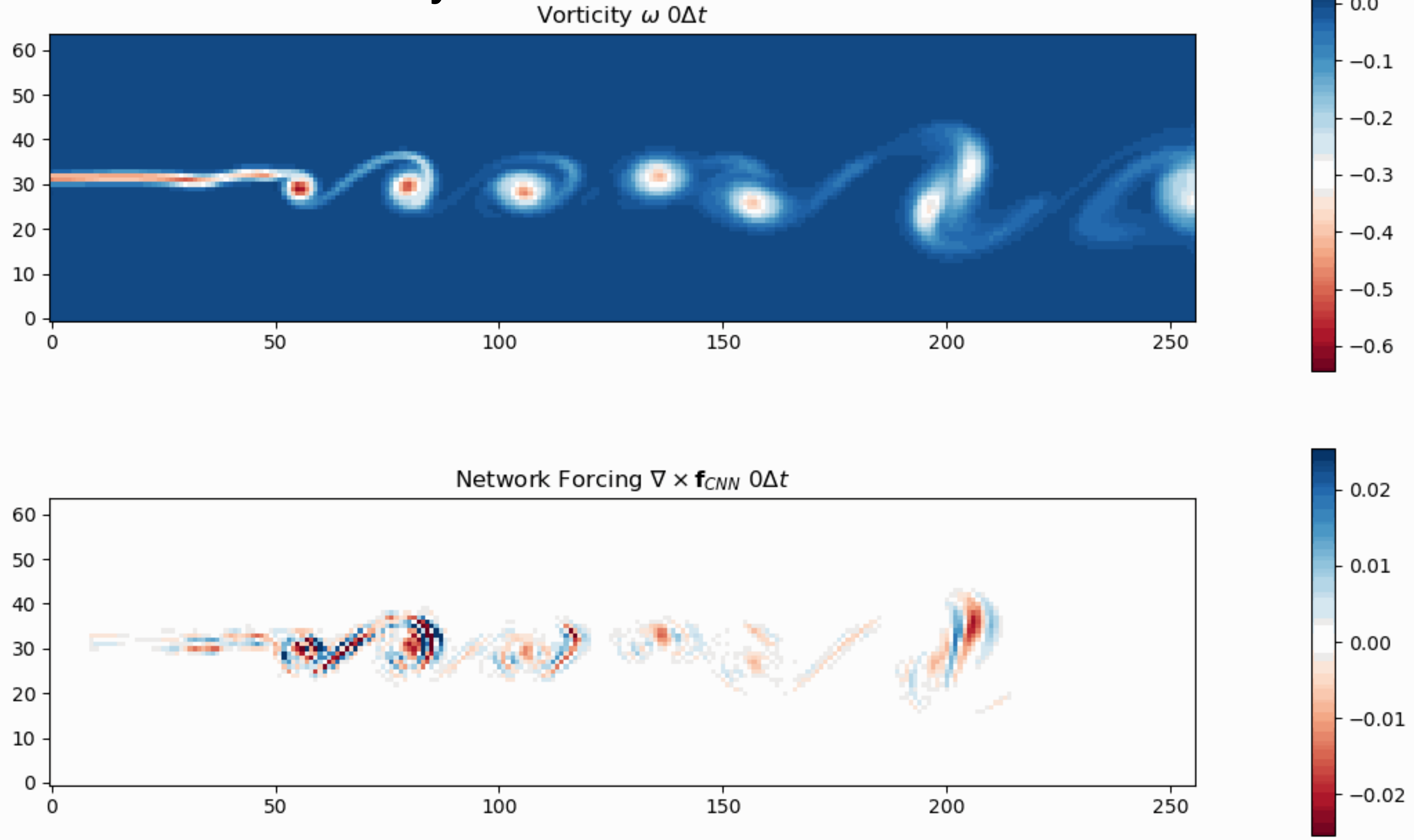
- Semi-implicit PISO solver (2nd order in time)
- Shear layer with vorticity thickness Re = 500
- Evaluate on test set of unseen perturbation modes



List et. al: Learned Turbulence Modelling with Differentiable Fluid Solvers

Turbulence: Spatial Mixing Layer

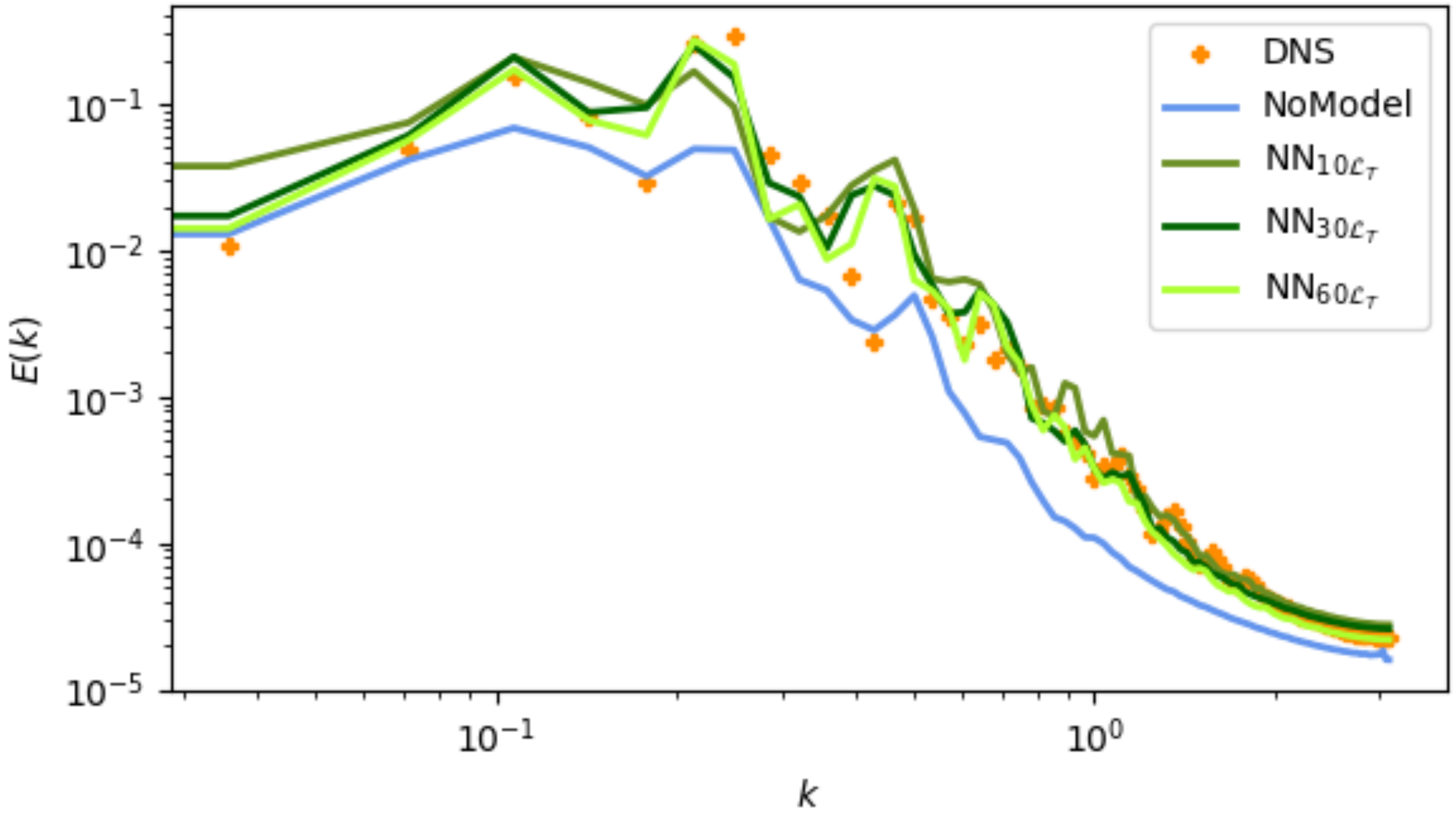
Learned Simulator only:



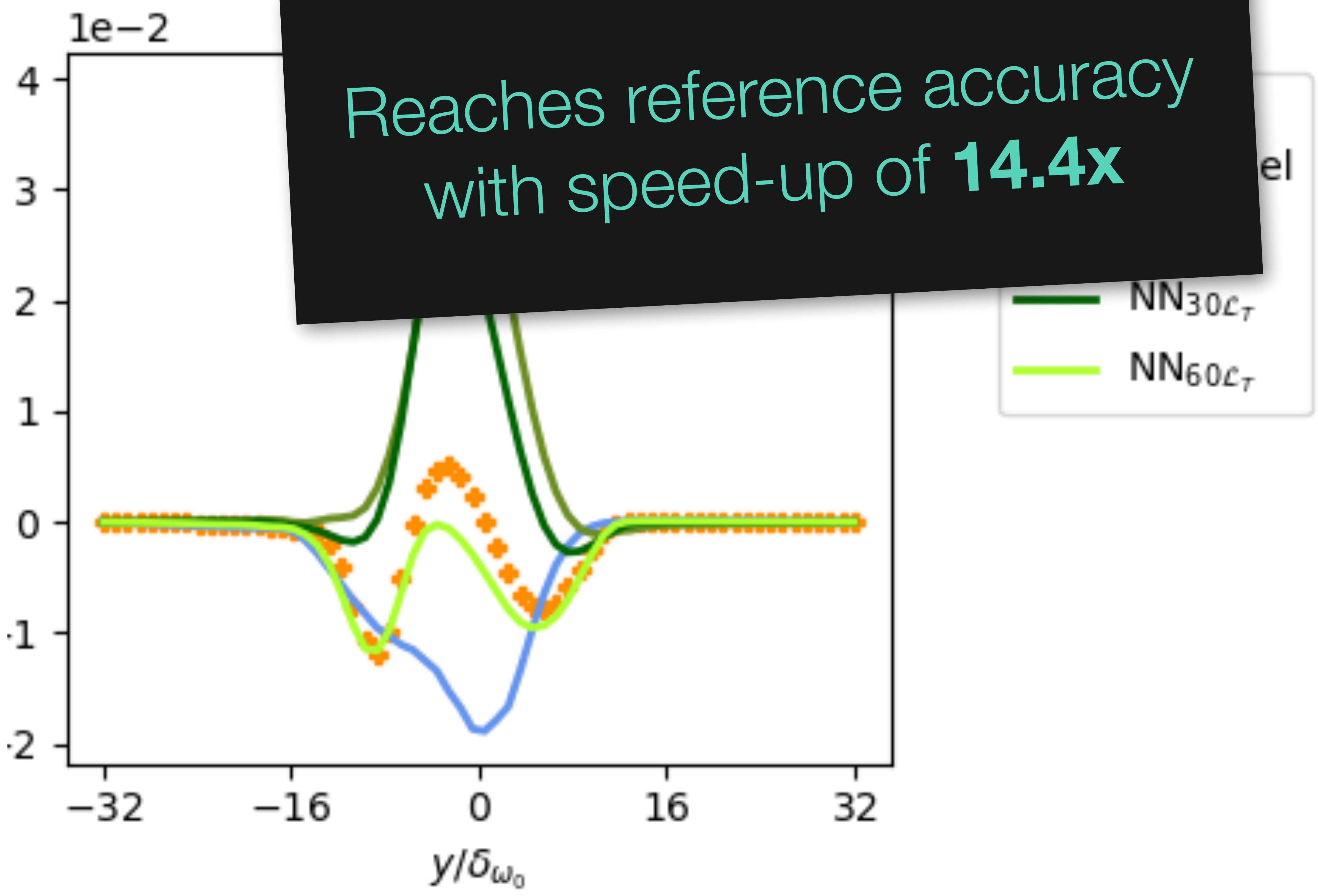
List et. al: Learned Turbulence Modelling with Differentiable Fluid Solvers

Turbulence: Spatial Mixing Layer

Closely matches DNS turbulence statistics (steady state over 2500 steps)



Energy spectrum



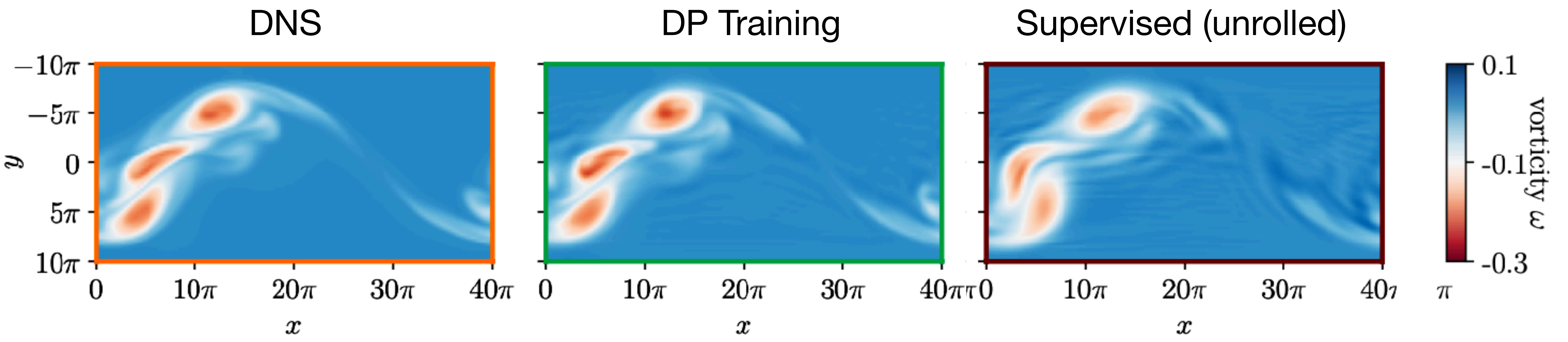
Reynolds stresses

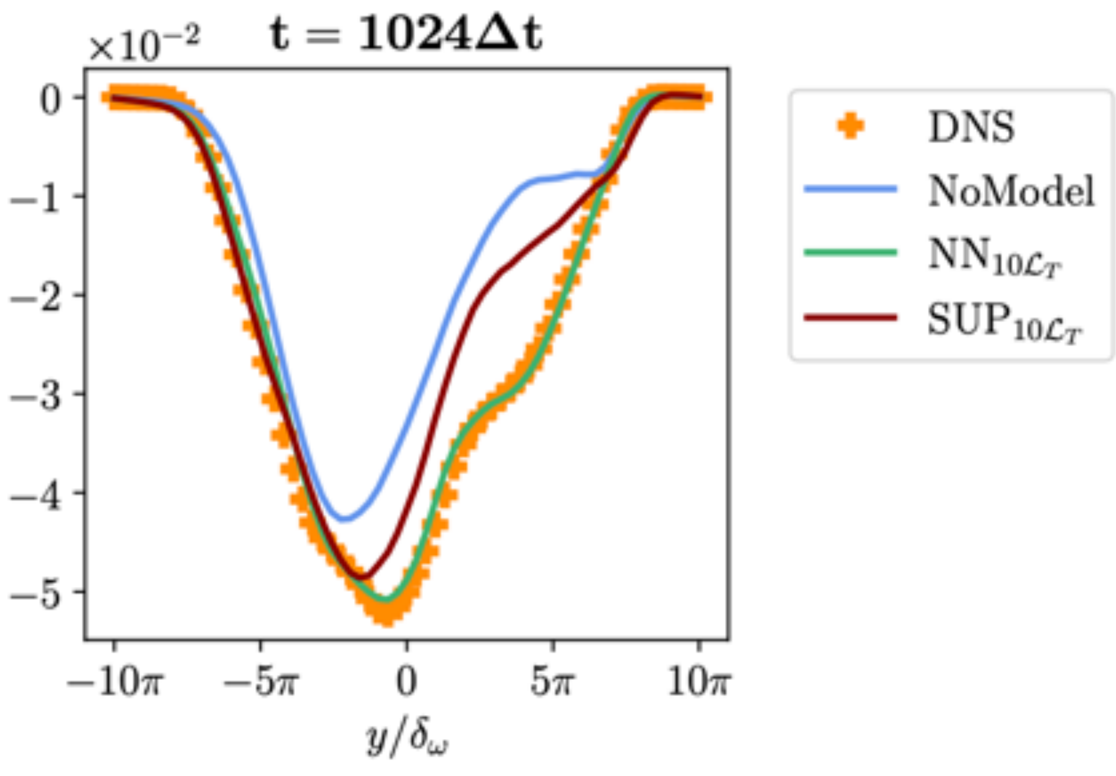
Reaches reference accuracy with speed-up of **14.4x**

List et. al: Learned Turbulence Modelling with Differentiable Fluid Solvers

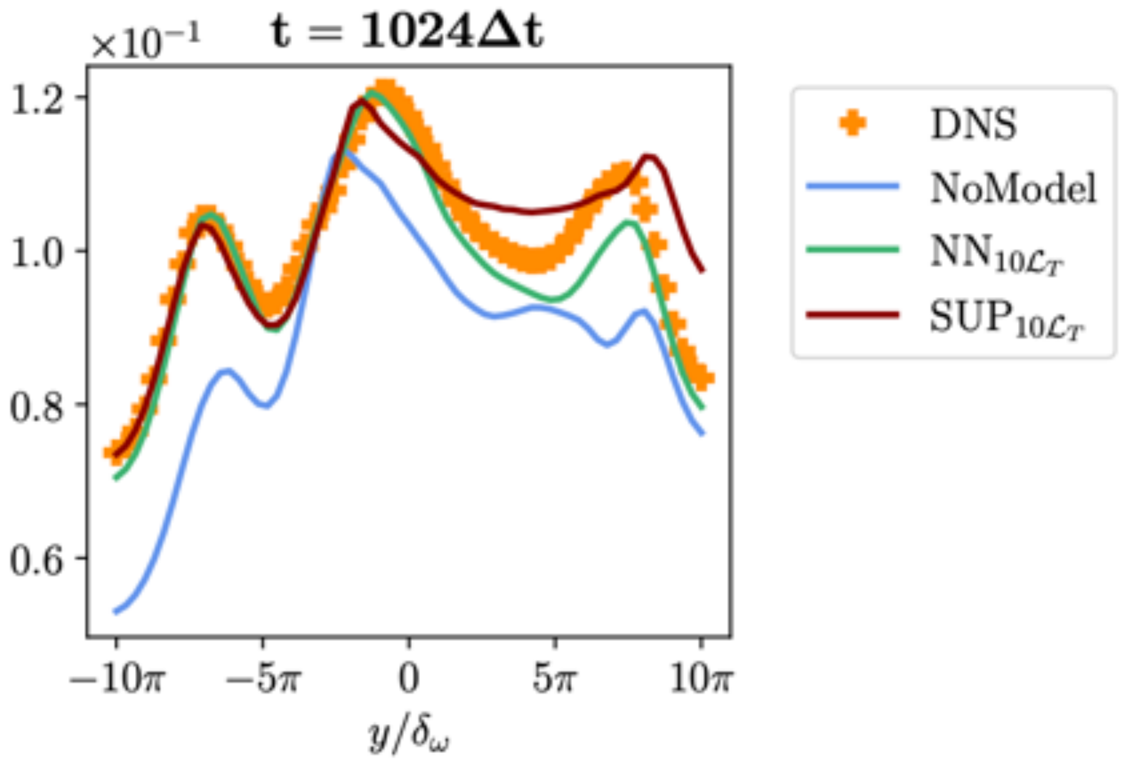
Turbulence: Temporal Mixing Layer

10 step unrolling, without and with DP training





Reynolds stresses



Turb. Kinetic Energy

List et. al: Learned Turbulence Modelling with Differentiable Fluid Solvers

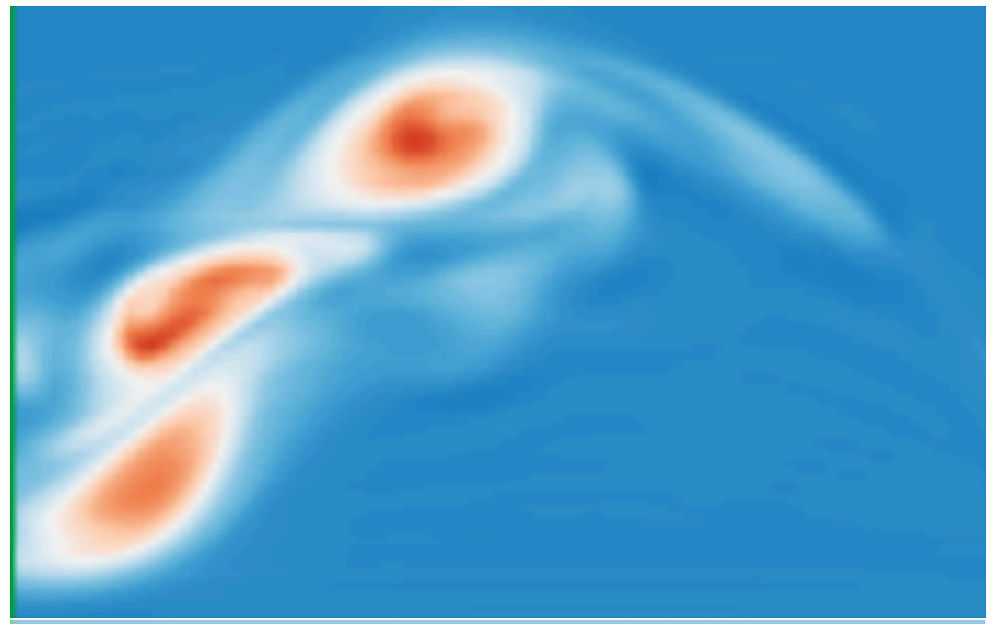
Turbulence Cases - Discussion

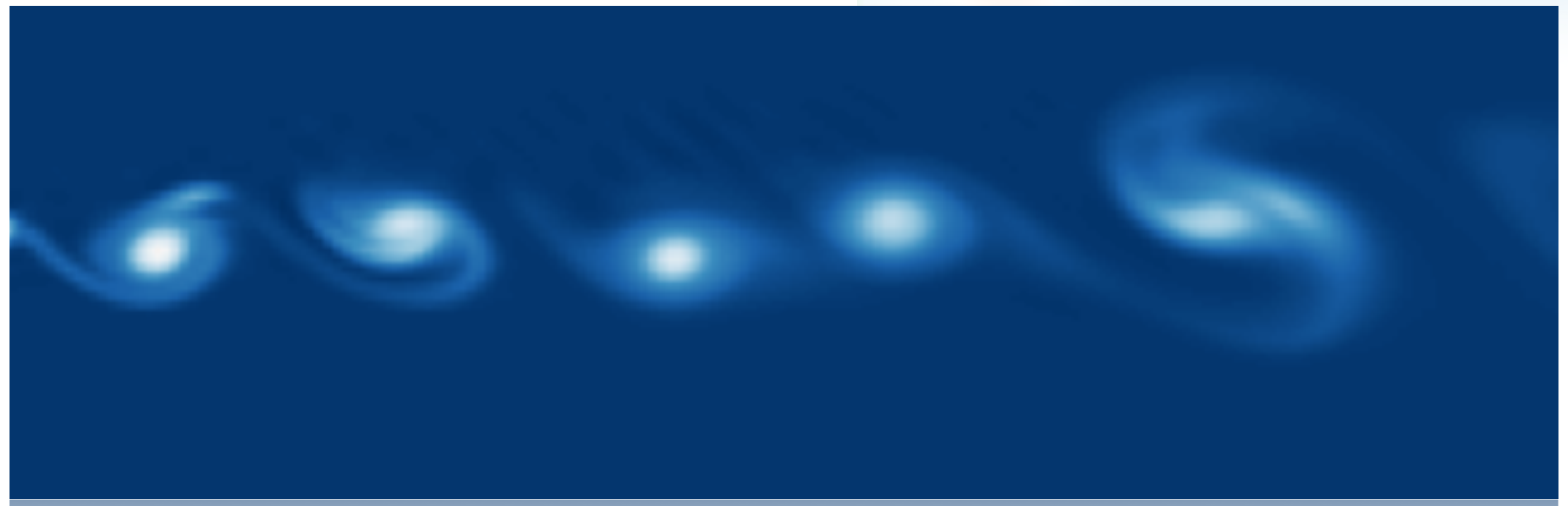
Illustrates importance of (long-term) DP training

Significant gains in accuracy per resource possible

Unrolled supervised training performs worse

⇨ Long term feedback (*gradient flow*) crucial





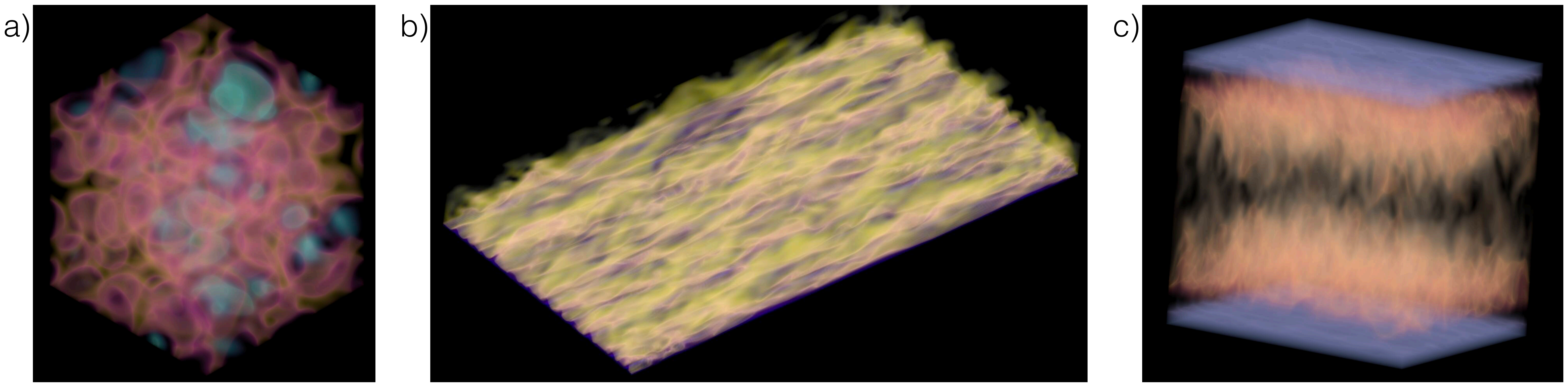
List et. al: Learned Turbulence Modelling with Differentiable Fluid Solvers

Differentiable Physics Example 2

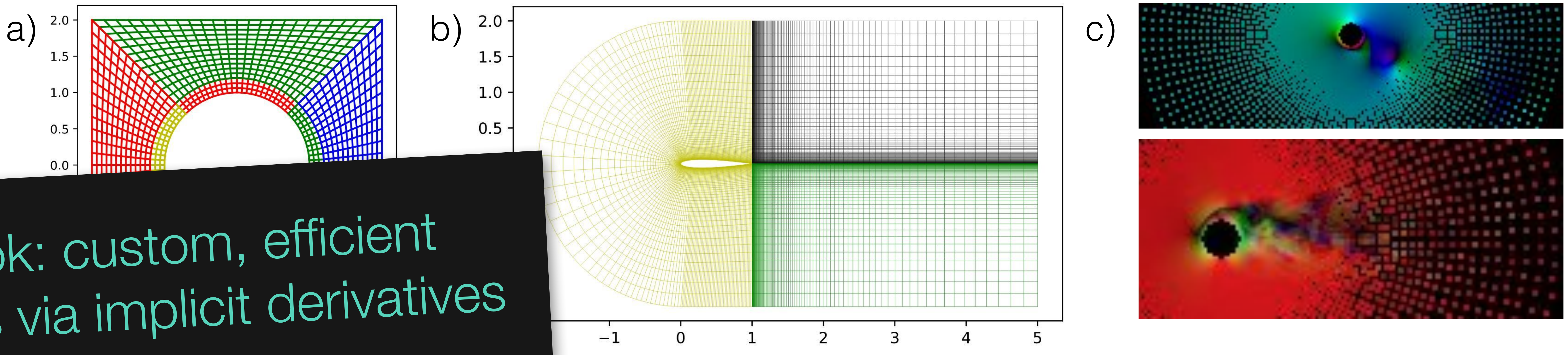
Franz et. al: PICT – A Differentiable, GPU-Accelerated Multi-Block PISO Solver for Simulation-Coupled Learning Tasks in Fluid Dynamics

More Advanced Solver

- Higher order, supports 3D simulations



- Adaptive meshes , refine near regions of interest



Outlook: custom, efficient gradients via implicit derivatives

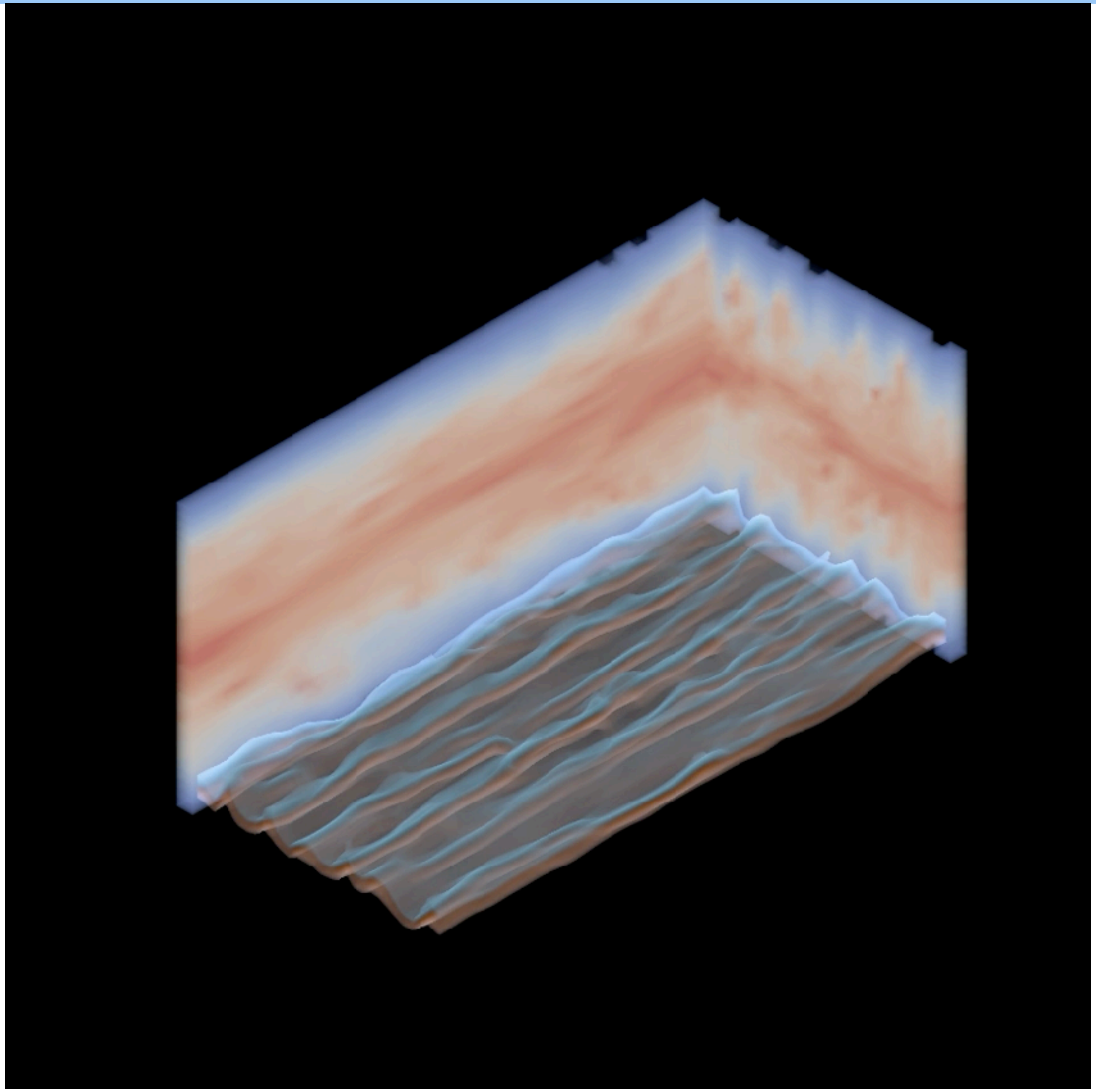
Franz et. al: PICT – A Differentiable, GPU-Accelerated Multi-Block PISO Solver for Simulation-Coupled Learning Tasks in Fluid Dynamics

Training Turbulence Closure

Chaotic systems: state supervision problematic in the long term

⇨ Supervise turbulence statistics

In this case from high-fidelity spectral solver



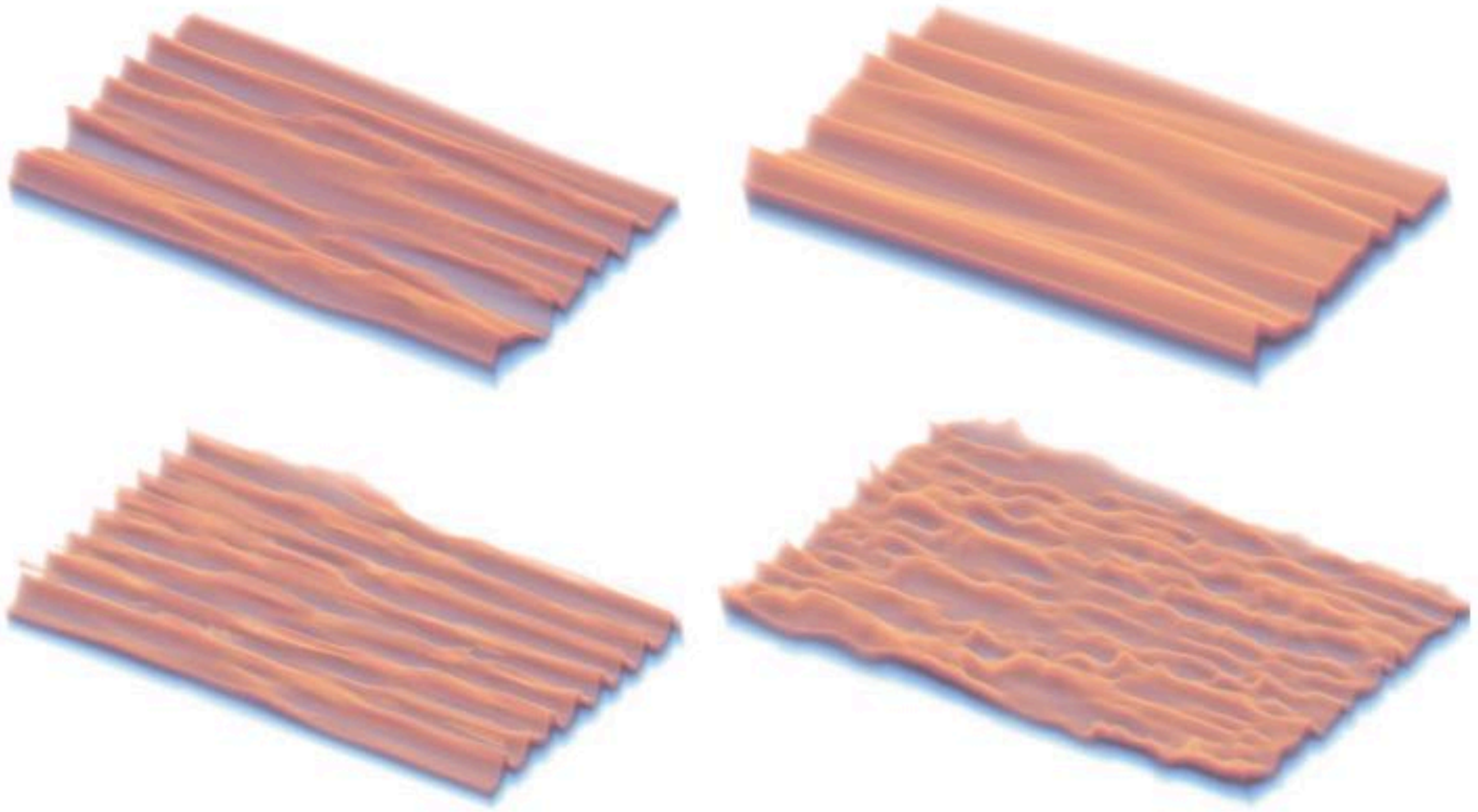
Franz et. al: PICT – A Differentiable, GPU-Accelerated Multi-Block PISO Solver for Simulation-Coupled Learning Tasks in Fluid Dynamics

Turbulent Channel Flow

Example States

Low-fidelity

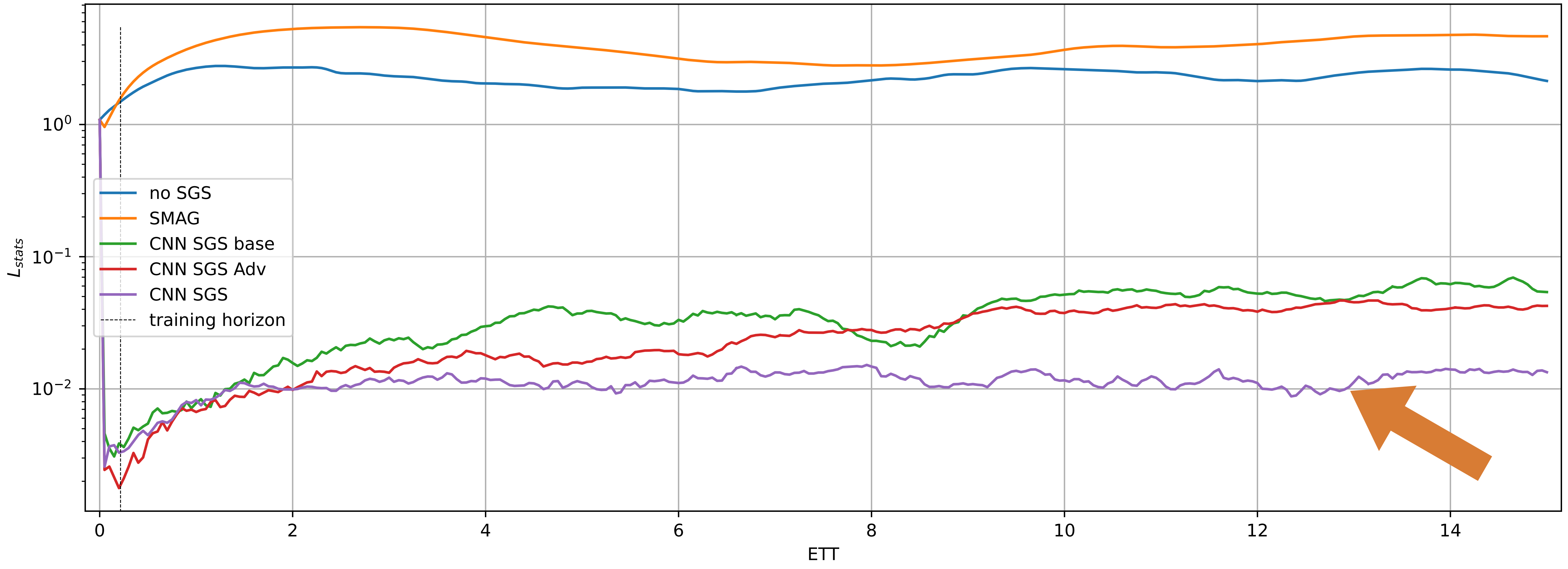
Simple Turb. Model



Learned

Reference

Turbulence Statistics over Time



Franz et. al: PICT – A Differentiable, GPU-Accelerated Multi-Block PISO Solver for Simulation-Coupled Learning Tasks in Fluid Dynamics

Differentiable Physics Examples Done

Differentiable Physics Training

Summary

✅ Fully uses solvers, existing methods and guarantees

✅ Efficiency and accuracy carries over

✅ Improved accuracy and generalization

❌ Needs solver support

❌ Higher bar for entry…

End