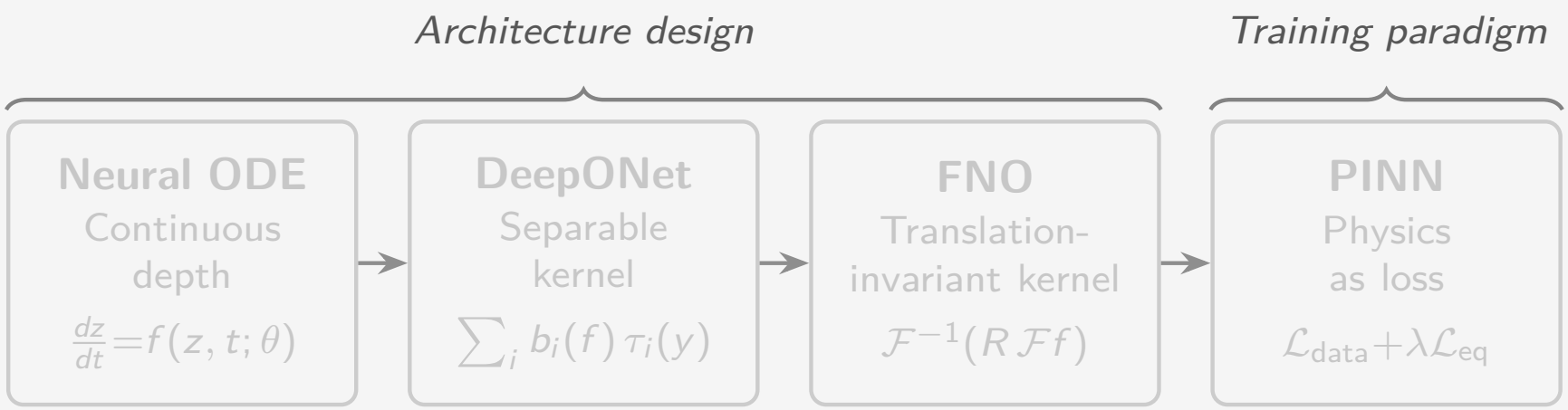
ECE 228 Spring 2026 Lecture 16: Foundation Models for Scientific Machine Learning

Luke Bhan

May 21, 2026

Where we've been

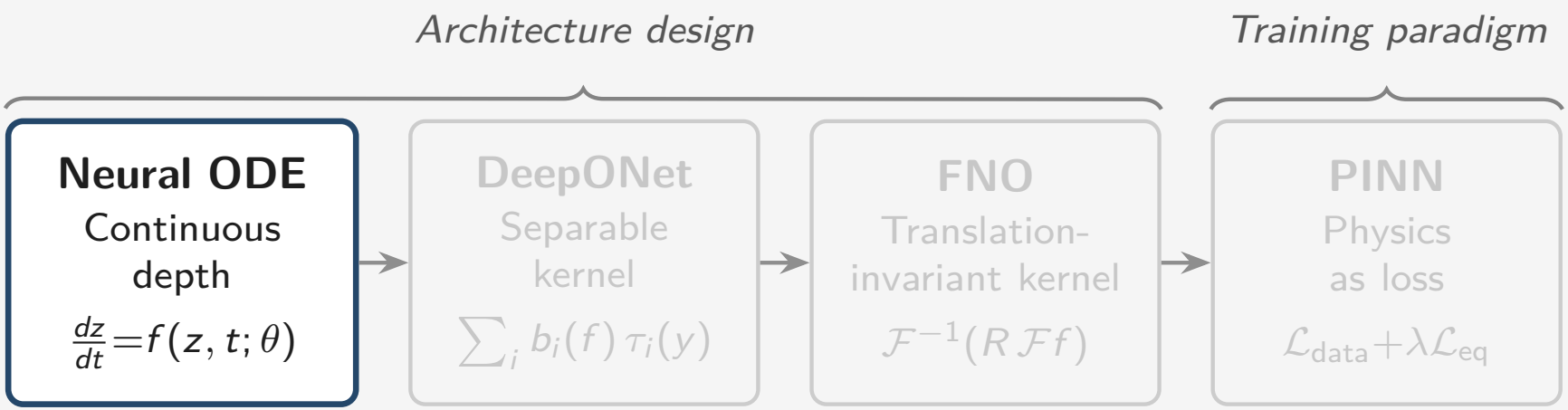
Four lectures, four ways to bake structure into a neural network:



The first four lectures asked "what should the network and loss look like?" Today: "what should we train it on, and how broadly?"

Where we've been

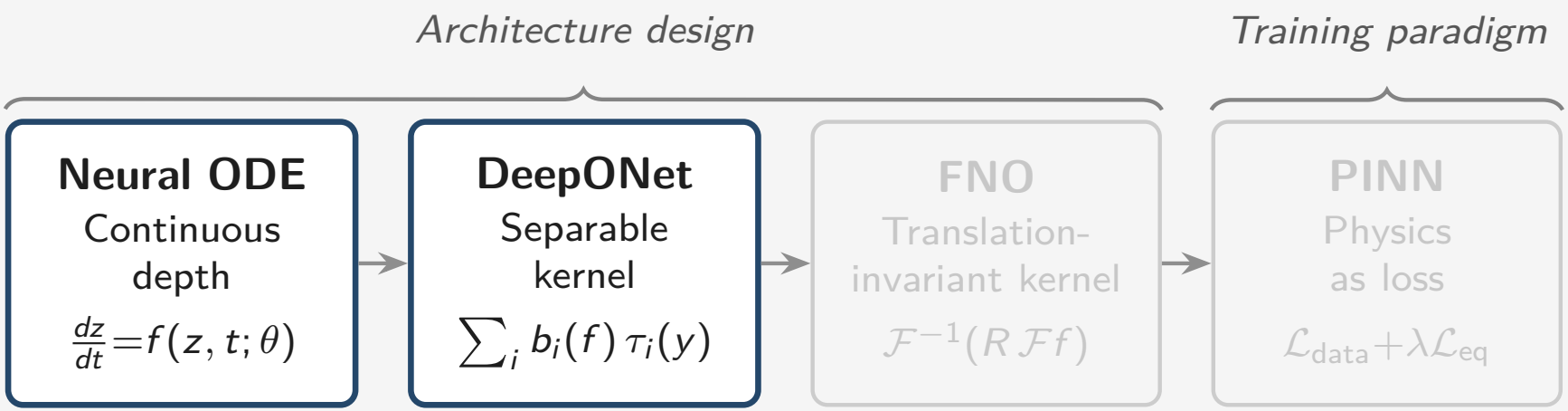
Four lectures, four ways to bake structure into a neural network:



The first four lectures asked "what should the network and loss look like?" Today: "what should we train it on, and how broadly?"

Where we've been

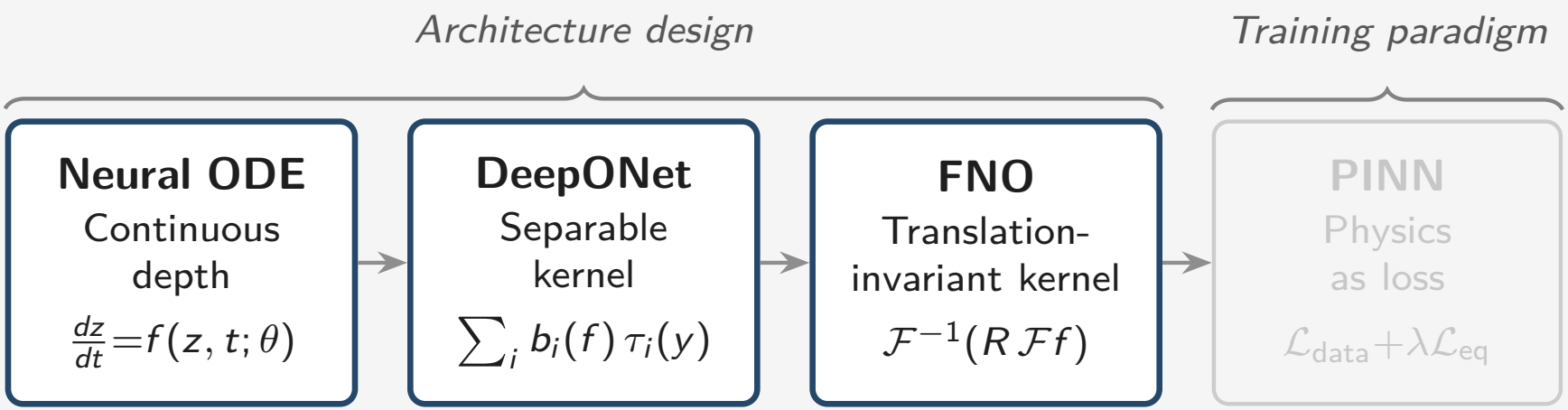
Four lectures, four ways to bake structure into a neural network:



The first four lectures asked "what should the network and loss look like?" Today: "what should we train it on, and how broadly?"

Where we've been

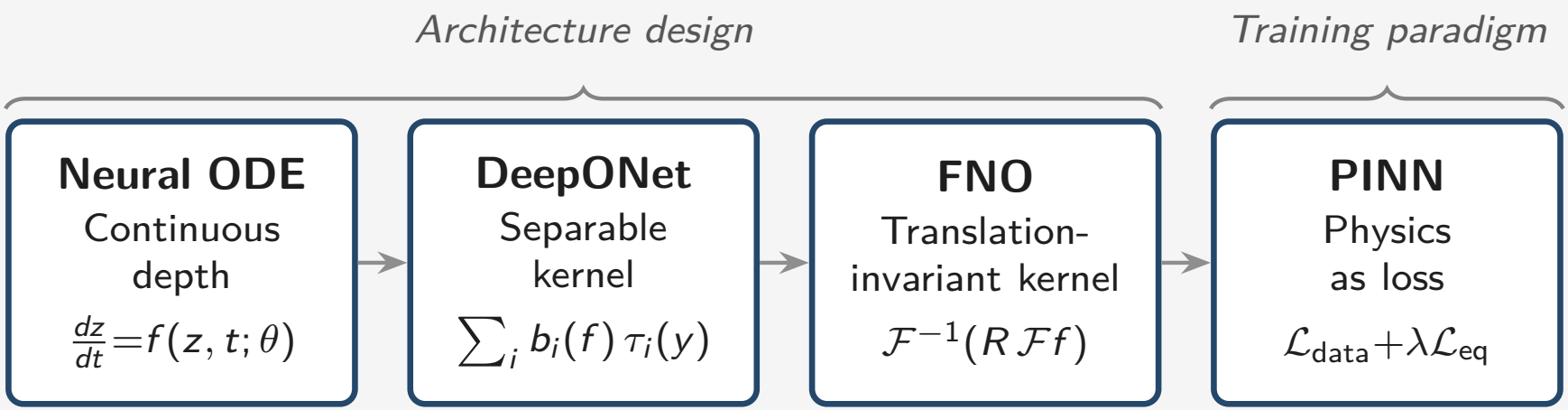
Four lectures, four ways to bake structure into a neural network:



The first four lectures asked "what should the network and loss look like?" Today: "what should we train it on, and how broadly?"

Where we've been

Four lectures, four ways to bake structure into a neural network:



The first four lectures asked "what should the network and loss look like?" Today: "what should we train it on, and how broadly?"

Where we've been: every model is bespoke

Each method we've built solves *one* problem family. Step outside, and the model doesn't help:

Every architecture in this series is a **bespoke surrogate**.
What if we wanted *one* model that handles all of them?

Where we've been: every model is bespoke

Each method we've built solves *one* problem family. Step outside, and the model doesn't help:

- **Neural ODE** trained on one dynamical system
  $\Rightarrow$ doesn't transfer to a different robot or a different fluid.

Every architecture in this series is a **bespoke surrogate**.
What if we wanted *one* model that handles all of them?

Where we've been: every model is bespoke

Each method we've built solves *one* problem family. Step outside, and the model doesn't help:

- **Neural ODE** trained on one dynamical system
  $\Rightarrow$ doesn't transfer to a different robot or a different fluid.
- **DeepONet** trained on advection–diffusion
  $\Rightarrow$ doesn't help with Helmholtz or wave equations.

Every architecture in this series is a **bespoke surrogate**.
What if we wanted *one* model that handles all of them?

Where we've been: every model is bespoke

Each method we've built solves *one* problem family. Step outside, and the model doesn't help:

- **Neural ODE** trained on one dynamical system
  $\Rightarrow$ doesn't transfer to a different robot or a different fluid.
- **DeepONet** trained on advection–diffusion
  $\Rightarrow$ doesn't help with Helmholtz or wave equations.
- **FNO** trained on Navier–Stokes at $\mathrm{Re} = 100$
  $\Rightarrow$ degrades sharply at $\mathrm{Re} = 10{,}000$; useless at a different geometry.

Every architecture in this series is a **bespoke surrogate**.
What if we wanted *one* model that handles all of them?

Where we've been: every model is bespoke

Each method we've built solves *one* problem family. Step outside, and the model doesn't help:

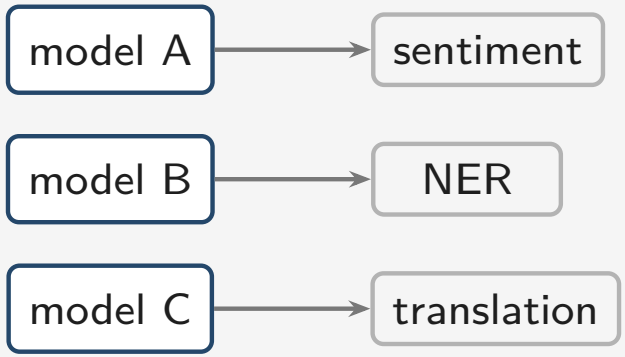
- **Neural ODE** trained on one dynamical system
  $\Rightarrow$ doesn't transfer to a different robot or a different fluid.
- **DeepONet** trained on advection–diffusion
  $\Rightarrow$ doesn't help with Helmholtz or wave equations.
- **FNO** trained on Navier–Stokes at $\mathrm{Re} = 100$
  $\Rightarrow$ degrades sharply at $\mathrm{Re} = 10{,}000$; useless at a different geometry.
- **PINN** trained for Burgers' equation
  $\Rightarrow$ Schrödinger needs a new network and a new training run.

Every architecture in this series is a **bespoke surrogate**.
What if we wanted *one* model that handles all of them?

An analogy: what happened in NLP

Pre-2018 NLP: every task was a bespoke model. **Post-GPT:** one model, prompted or lightly fine-tuned for many tasks.

Before: bespoke



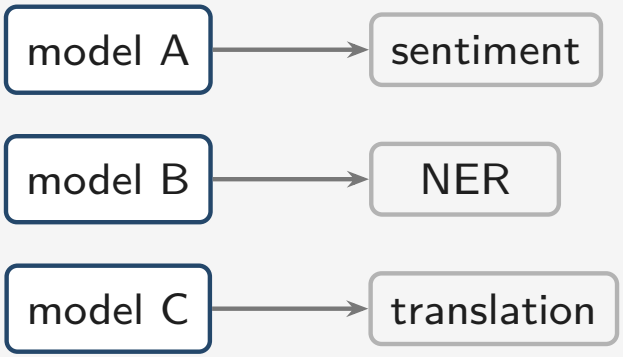
Modern SciML is trying to copy this recipe: pretrain broadly, adapt cheaply. Today, though, every new PDE family still demands its own bespoke architecture and training run.

The motivating question: can we have a **"GPT for PDEs"**?

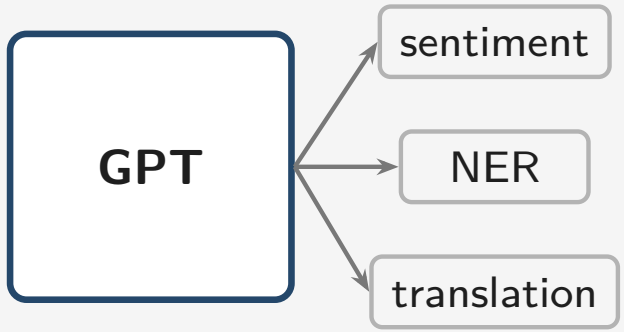
An analogy: what happened in NLP

Pre-2018 NLP: every task was a bespoke model. **Post-GPT:** one model, prompted or lightly fine-tuned for many tasks.

Before: bespoke



After: one model, many tasks



Modern SciML is trying to copy this recipe: pretrain broadly, adapt cheaply. Today, though, every new PDE family still demands its own bespoke architecture and training run.

The motivating question: can we have a **"GPT for PDEs"**?

What is a foundation model?

A **foundation model** is a model pretrained on *broad* data that can be *adapted* to many downstream tasks.
(paraphrasing Bommasani et al., Stanford CRFM, 2021)

Broad data. Diverse problems, not one task. Many physics in one training corpus, instead of one model per equation.

Adapt. Three modes, increasing cost:

What is a foundation model?

A **foundation model** is a model pretrained on *broad* data that can be *adapted* to many downstream tasks.
(paraphrasing Bommasani et al., Stanford CRFM, 2021)

Broad data. Diverse problems, not one task. Many physics in one training corpus, instead of one model per equation.

Adapt. Three modes, increasing cost:

What is a foundation model?

A **foundation model** is a model pretrained on *broad* data that can be *adapted* to many downstream tasks.
(paraphrasing Bommasani et al., Stanford CRFM, 2021)

Broad data. Diverse problems, not one task. Many physics in one training corpus, instead of one model per equation.

Adapt. Three modes, increasing cost:

What is a foundation model?

A **foundation model** is a model pretrained on *broad* data that can be *adapted* to many downstream tasks.
(paraphrasing Bommasani et al., Stanford CRFM, 2021)

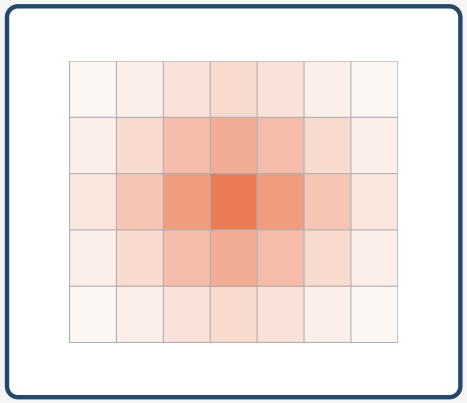
Broad data. Diverse problems, not one task. Many physics in one training corpus, instead of one model per equation.

Adapt. Three modes, increasing cost:

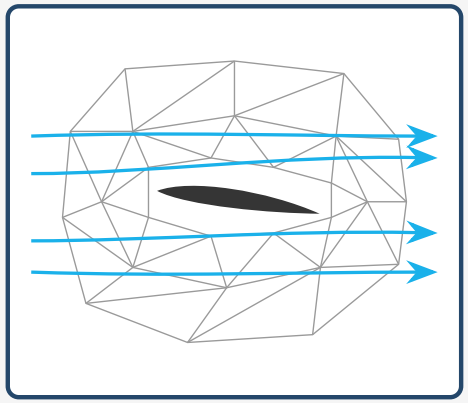
- **Zero-shot** — run the pretrained model on a new task, no weight updates.
- **Few-shot / in-context** — show the model a handful of examples at inference time.
- **Fine-tuning** — a short, targeted training run on task-specific data.

Why this is hard for PDEs

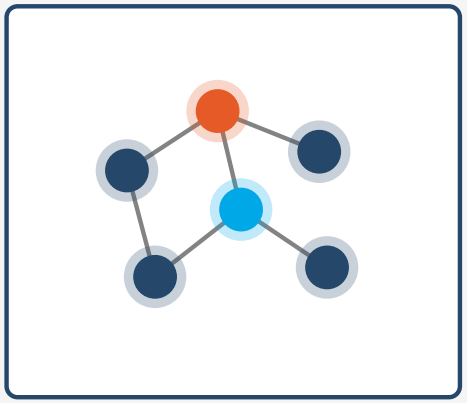
There is no unifying representation in physics



Heat equation (parabolic)
scalar field, regular grid



Navier–Stokes
vector field, unstructured mesh

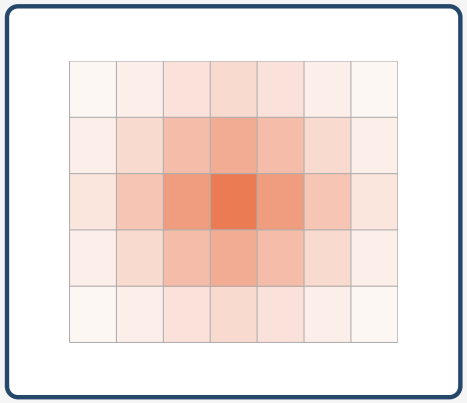


Weather over earth surface
point clouds, sensors

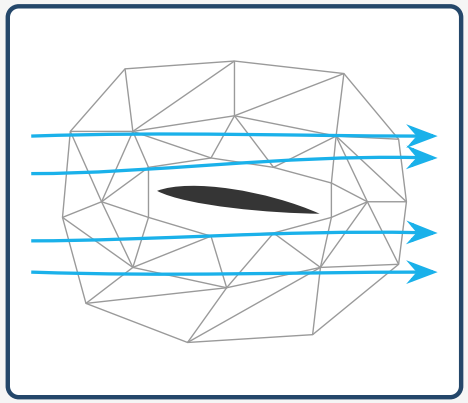
These are all "physics" — but they don't share a representation:

Why this is hard for PDEs

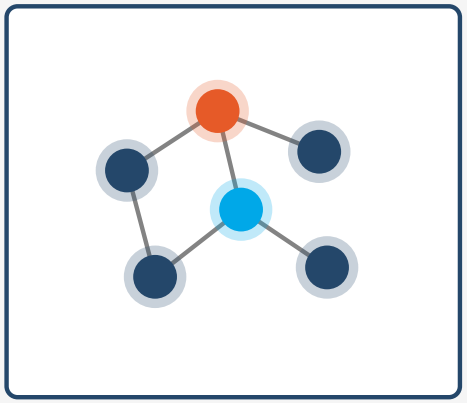
There is no unifying representation in physics



Heat equation (parabolic)
scalar field, regular grid



Navier–Stokes
vector field, unstructured mesh

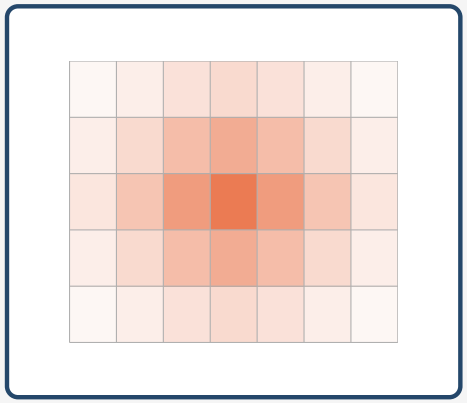


Weather over earth surface
point clouds, sensors

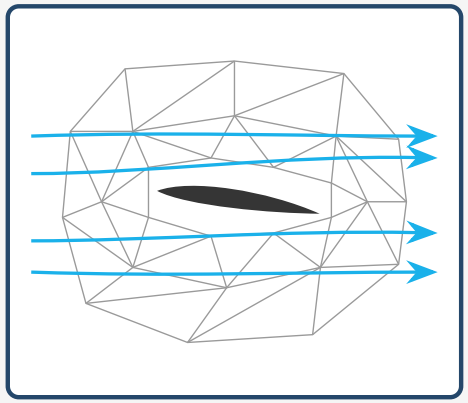
These are all "physics" — but they don't share a representation:

Why this is hard for PDEs

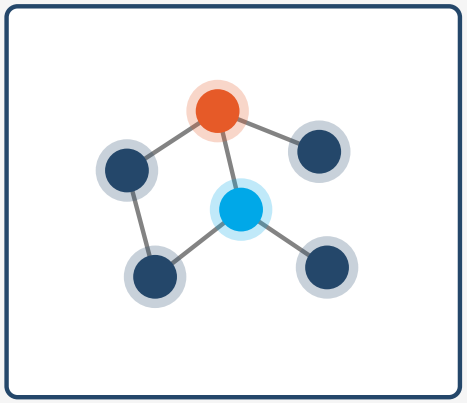
There is no unifying representation in physics



Heat equation (parabolic)
scalar field, regular grid



Navier–Stokes
vector field, unstructured mesh



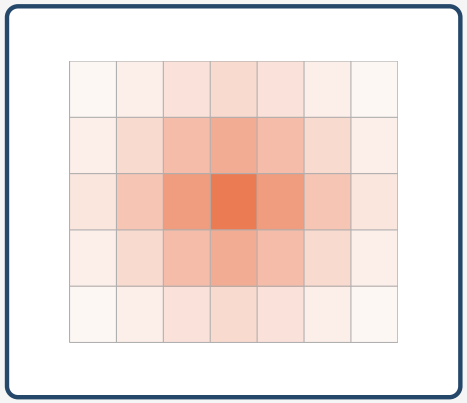
Weather over earth surface
point clouds, sensors

These are all "physics" — but they don't share a representation:

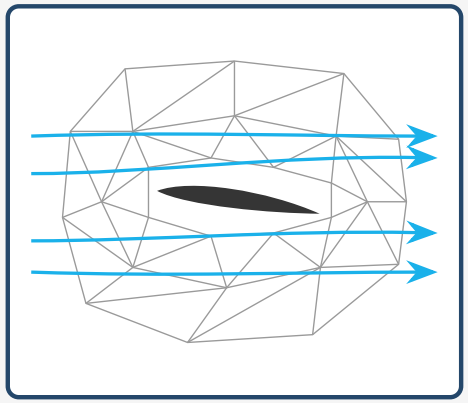
- **Different representations.** Scalar temperature, vector velocity, point-cloud atoms — different shapes, different units.

Why this is hard for PDEs

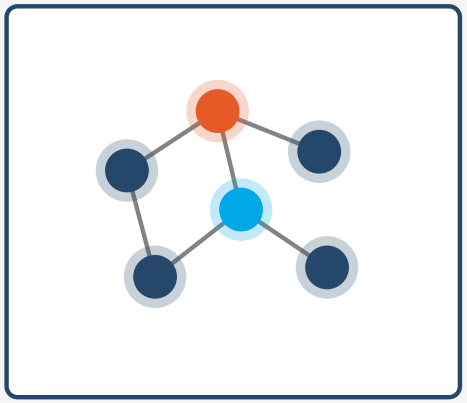
There is no unifying representation in physics



Heat equation (parabolic)
scalar field, regular grid



Navier–Stokes
vector field, unstructured mesh



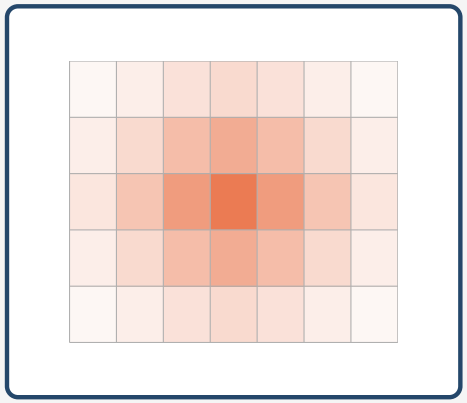
Weather over earth surface
point clouds, sensors

These are all "physics" — but they don't share a representation:

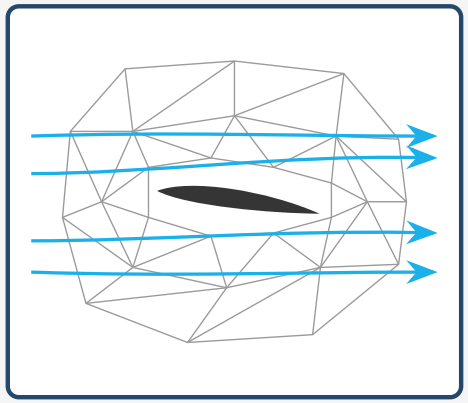
- **Different representations.** Scalar temperature, vector velocity, point-cloud atoms — different shapes, different units.
- **Different geometries.** Regular grids, airfoil meshes, spherical domains. No canonical layout.

Why this is hard for PDEs

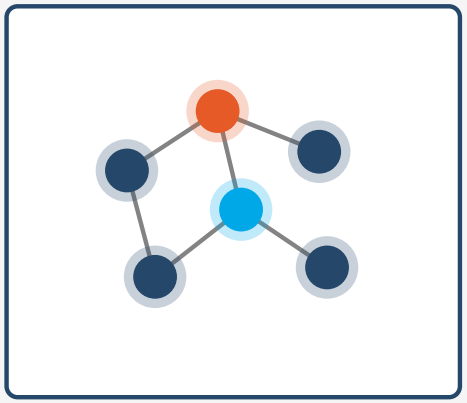
There is no unifying representation in physics



Heat equation (parabolic)
scalar field, regular grid



Navier–Stokes
vector field, unstructured mesh



Weather over earth surface
point clouds, sensors

These are all "physics" — but they don't share a representation:

- **Different representations.** Scalar temperature, vector velocity, point-cloud atoms — different shapes, different units.
- **Different geometries.** Regular grids, airfoil meshes, spherical domains. No canonical layout.
- **Different scales.** Viscosity ranges over 10 orders of magnitude across applications.

Training paradigm: what makes it "foundation"

The pipeline is borrowed from NLP. What's actually new for SciML is *how* and *on what* we train.

Idea of foundation models: Pretrain *once* on many physics, adapt *cheaply* to your specific task.

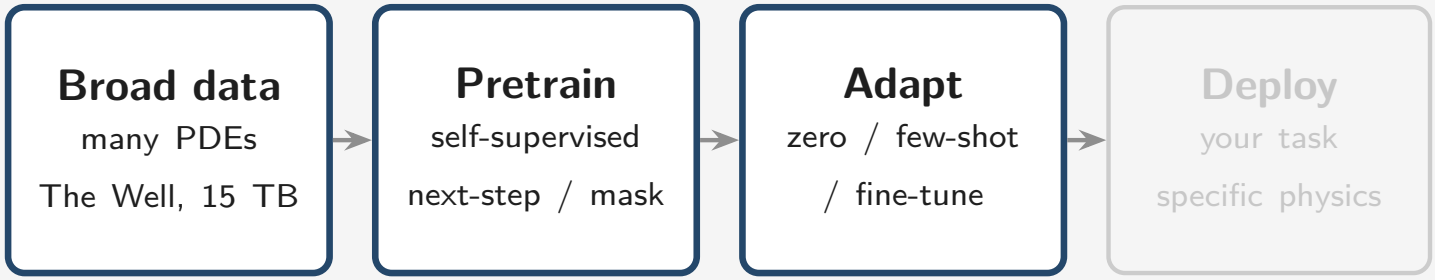
Training paradigm: what makes it "foundation"

The pipeline is borrowed from NLP. What's actually new for SciML is *how* and *on what* we train.

Idea of foundation models: Pretrain *once* on many physics, adapt *cheaply* to your specific task.

Training paradigm: what makes it "foundation"

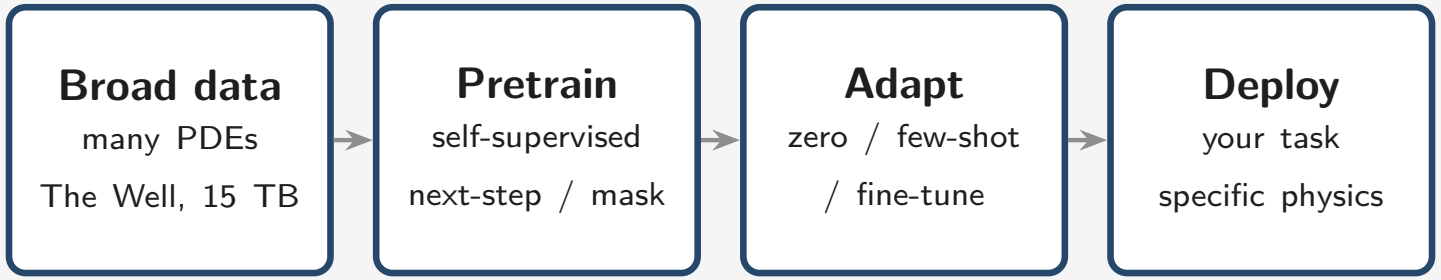
The pipeline is borrowed from NLP. What's actually new for SciML is *how* and *on what* we train.



Idea of foundation models: Pretrain *once* on many physics, adapt *cheaply* to your specific task.

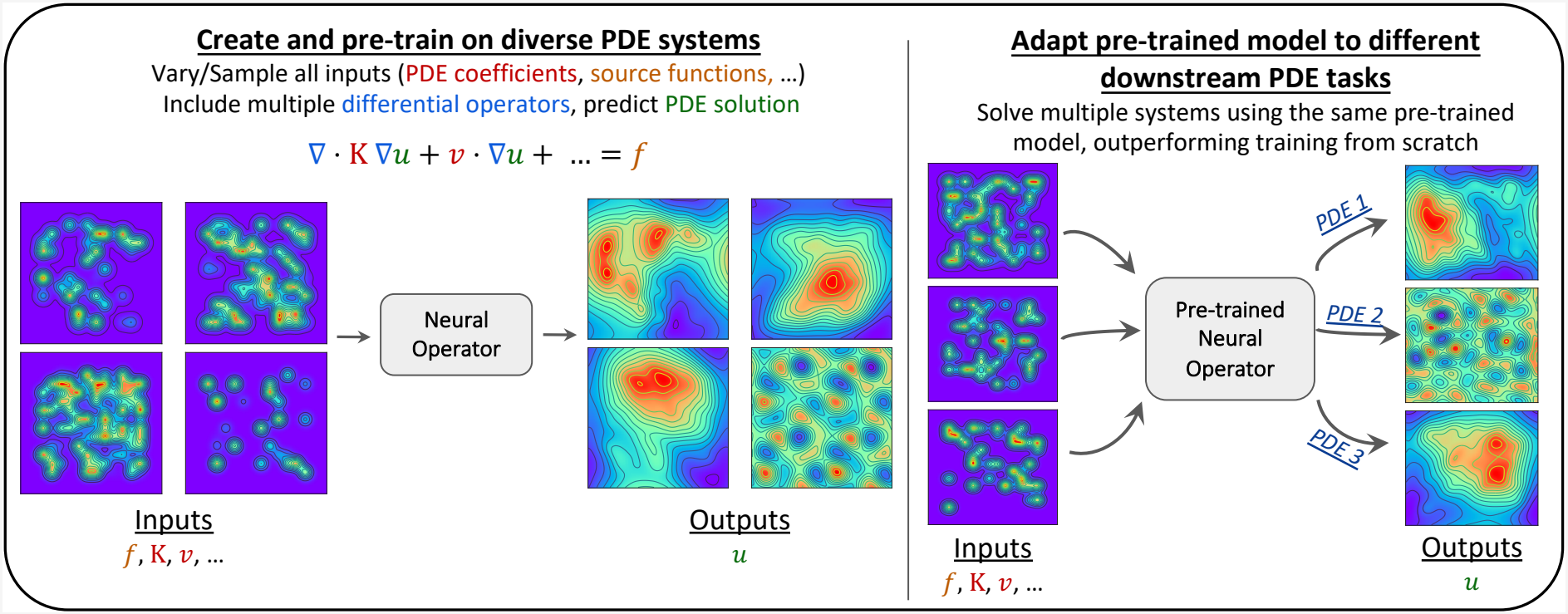
Training paradigm: what makes it "foundation"

The pipeline is borrowed from NLP. What's actually new for SciML is *how* and *on what* we train.



Idea of foundation models: Pretrain *once* on many physics, adapt *cheaply* to your specific task.

How do foundation models work?



Why physics is harder than NLP

Three things NLP got for free that physics does not:

Foundation models are genuinely an open question of whether they will work for physics.

Why physics is harder than NLP

Three things NLP got for free that physics does not:

- **No web-scale corpus.**
  NLP: Common Crawl, ∼petabytes of text. Physics: simulations you have to run yourself.

Foundation models are genuinely an open question of whether they will work for physics.

Why physics is harder than NLP

Three things NLP got for free that physics does not:

- **No web-scale corpus.**
  NLP: Common Crawl, ∼petabytes of text. Physics: simulations you have to run yourself.
- **No canonical token.**
  NLP: text is already a sequence of discrete tokens. Physics: $u(x, t)$ is a continuous field on some geometry.

Foundation models are genuinely an open question of whether they will work for physics.

Why physics is harder than NLP

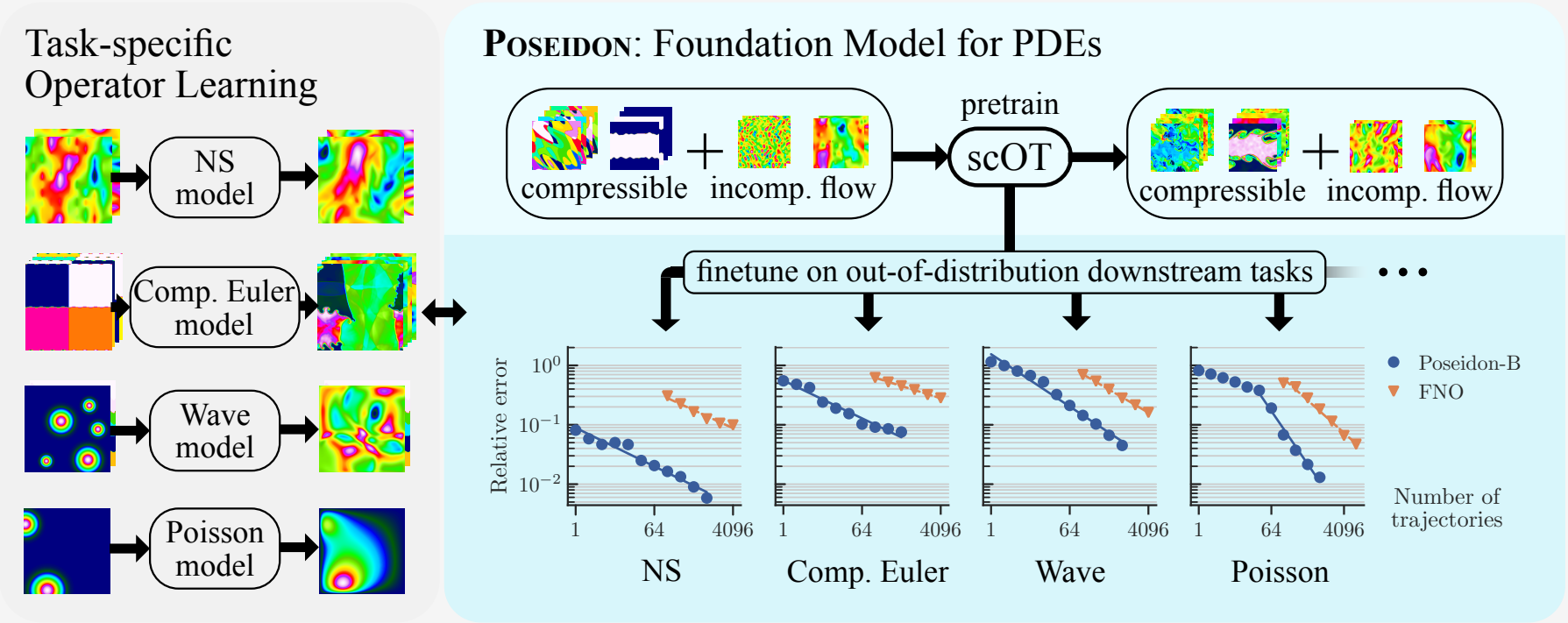
Three things NLP got for free that physics does not:

- **No web-scale corpus.**
  NLP: Common Crawl, ∼petabytes of text. Physics: simulations you have to run yourself.
- **No canonical token.**
  NLP: text is already a sequence of discrete tokens. Physics: $u(x, t)$ is a continuous field on some geometry.
- **Chaotic dynamics.**
  Language is inherently structured — grammar, syntax, semantics. Physics can be chaotic: nearby trajectories diverge exponentially.

Foundation models are genuinely an open question of whether they will work for physics.

A case study: Poseidon

Poseidon (paper).
Pretrained on fluid dynamics. Evaluated on **15 PDE tasks**, 9 of which involve physics never seen at pretraining.



The puzzle. How does a model trained on Euler + Navier–Stokes learn anything useful about Poisson or Helmholtz?

Problem formulation

The choice. 6 operators, all fluid dynamics, fixed geometries:

- 4 from compressible Euler PDEs
- 2 from incompressible Navier–Stokes PDEs
- **77,840 trajectories total.**
- Downstream transfer to other PDEs such as Helmholtz or Poisson.

Argument for:

- Fluids are well-understood — known solvers, clean data.
- Rich phenomenology: shocks, shear, vortex roll-ups, mixing.
- 6 operators cover many regimes in one PDE family.

Argument against:

- Why would fluids transfer to Helmholtz or Poisson?
- No elliptic, parabolic, or wave physics at all.
- NLP analog: pretraining only on news, expecting poetry.

Problem formulation

The choice. 6 operators, all fluid dynamics, fixed geometries:

- 4 from compressible Euler PDEs
- 2 from incompressible Navier–Stokes PDEs
- **77,840 trajectories total.**
- Downstream transfer to other PDEs such as Helmholtz or Poisson.

Discuss for 60 seconds.

Pretraining on *only* fluids — good idea or bad idea?
What's the strongest argument on each side?

Argument for:

- Fluids are well-understood — known solvers, clean data.
- Rich phenomenology: shocks, shear, vortex roll-ups, mixing.
- 6 operators cover many regimes in one PDE family.

Argument against:

- Why would fluids transfer to Helmholtz or Poisson?
- No elliptic, parabolic, or wave physics at all.
- NLP analog: pretraining only on news, expecting poetry.

How to get more data???

Key trick. Maximize what you have. From each trajectory of $K+1$ snapshots, use *every* ordered pair $(t_k, t_{\bar{k}})$ with $k \leq \bar{k}$ as a training example.

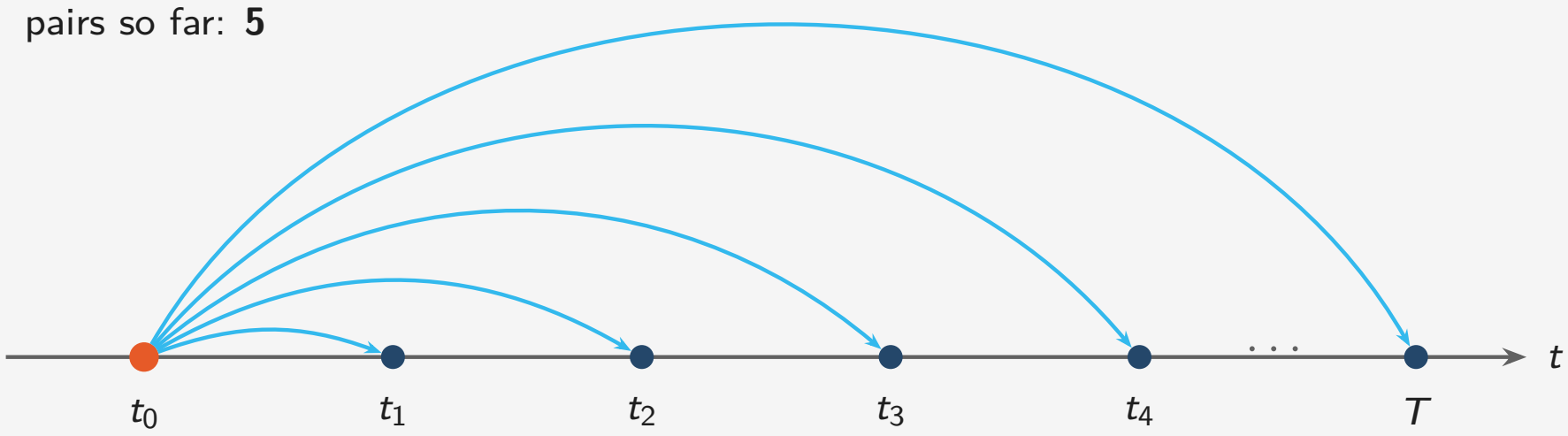
pairs so far: 0

K snapshots, but $\binom{K+1}{2} \approx K^2/2$ training pairs. **Quadratic data multiplier, essentially for free.**

The catch. The input is $u(t_k)$, but the target depends on $\Delta t = t_{\bar{k}} - t_k$ — which varies across pairs. The model needs Δt as an input, not just u.

How to get more data???

Key trick. Maximize what you have. From each trajectory of $K+1$ snapshots, use *every* ordered pair $(t_k, t_{\bar{k}})$ with $k \leq \bar{k}$ as a training example.

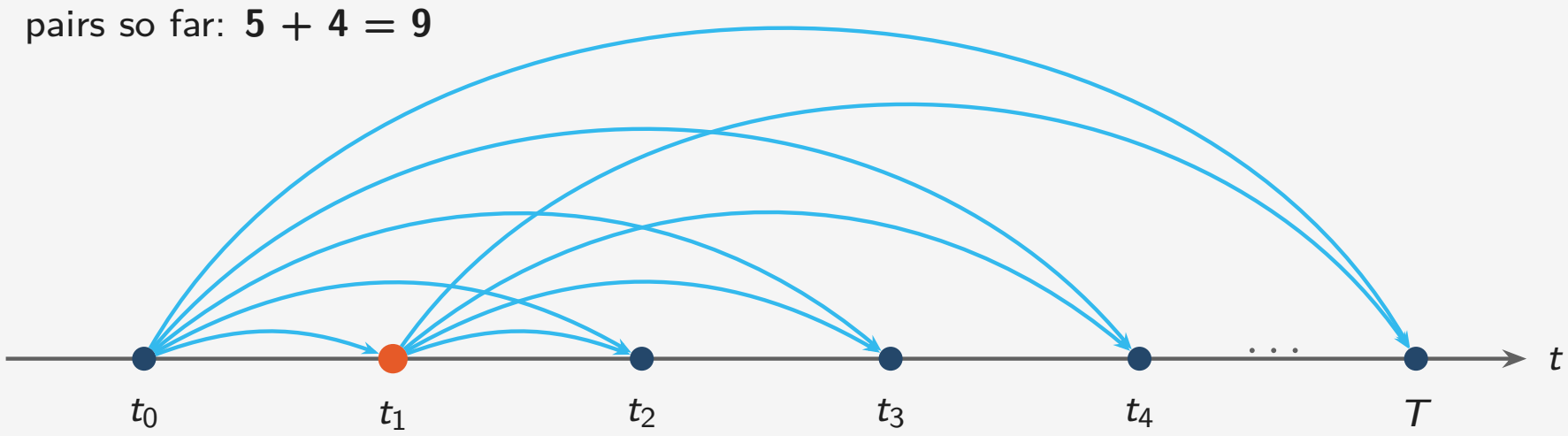


K snapshots, but $\binom{K+1}{2} \approx K^2/2$ training pairs. **Quadratic data multiplier, essentially for free.**

The catch. The input is $u(t_k)$, but the target depends on $\Delta t = t_{\bar{k}} - t_k$ — which varies across pairs. The model needs Δt as an input, not just u.

How to get more data???

Key trick. Maximize what you have. From each trajectory of $K+1$ snapshots, use *every* ordered pair $(t_k, t_{\bar{k}})$ with $k \leq \bar{k}$ as a training example.

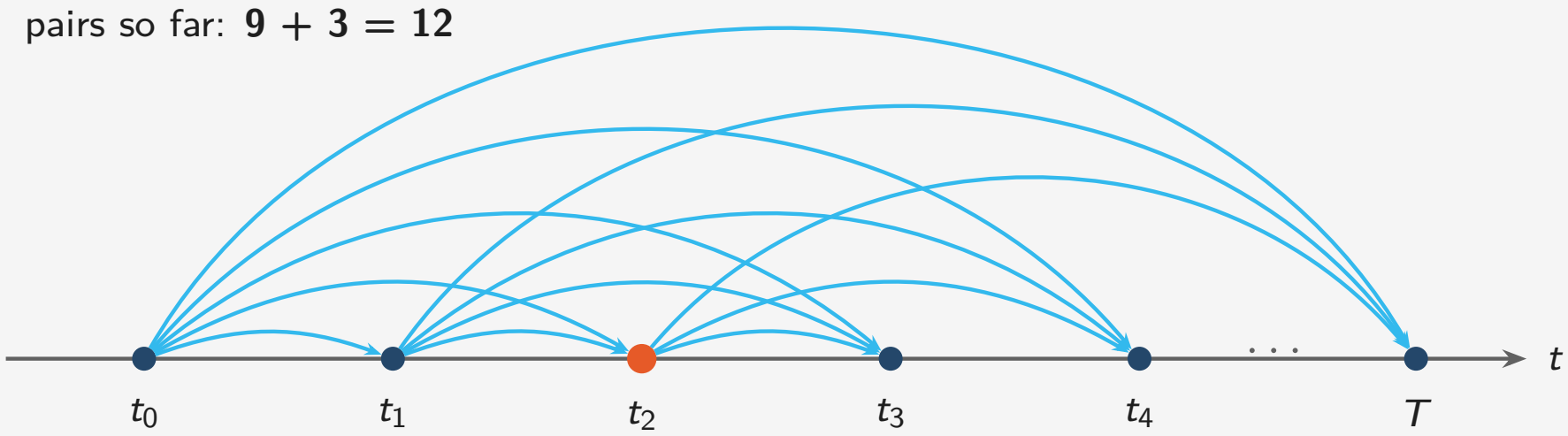


K snapshots, but $\binom{K+1}{2} \approx K^2/2$ training pairs. **Quadratic data multiplier, essentially for free.**

The catch. The input is $u(t_k)$, but the target depends on $\Delta t = t_{\bar{k}} - t_k$ — which varies across pairs. The model needs Δt as an input, not just u.

How to get more data???

Key trick. Maximize what you have. From each trajectory of $K+1$ snapshots, use *every* ordered pair $(t_k, t_{\bar{k}})$ with $k \leq \bar{k}$ as a training example.

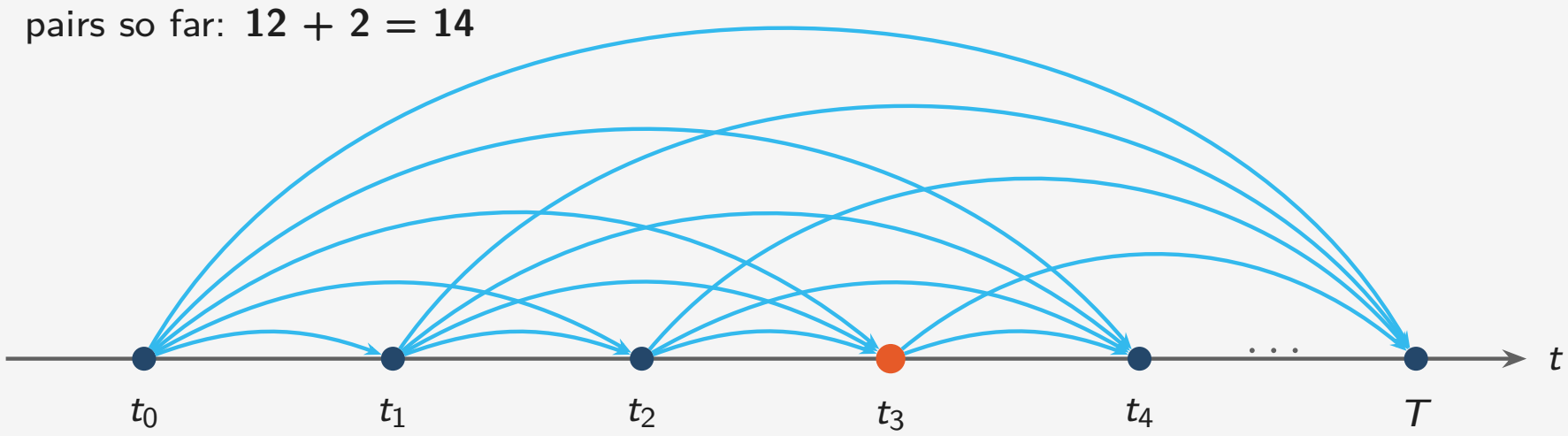


K snapshots, but $\binom{K+1}{2} \approx K^2/2$ training pairs. **Quadratic data multiplier, essentially for free.**

The catch. The input is $u(t_k)$, but the target depends on $\Delta t = t_{\bar{k}} - t_k$ — which varies across pairs. The model needs Δt as an input, not just u.

How to get more data???

Key trick. Maximize what you have. From each trajectory of $K+1$ snapshots, use *every* ordered pair $(t_k, t_{\bar{k}})$ with $k \leq \bar{k}$ as a training example.

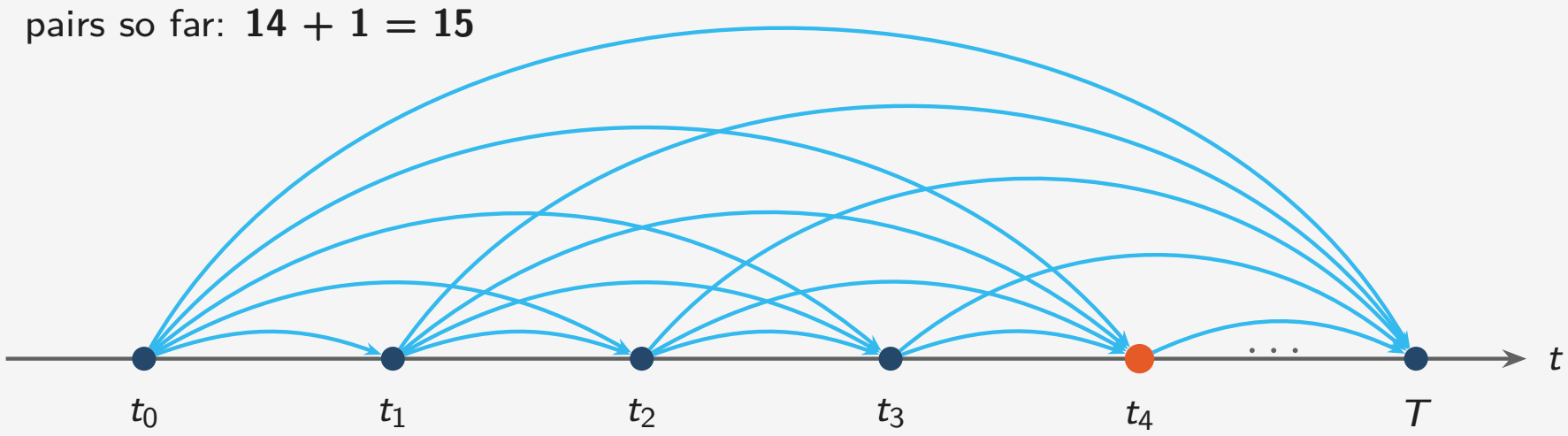


K snapshots, but $\binom{K+1}{2} \approx K^2/2$ training pairs. **Quadratic data multiplier, essentially for free.**

The catch. The input is $u(t_k)$, but the target depends on $\Delta t = t_{\bar{k}} - t_k$ — which varies across pairs. The model needs Δt as an input, not just u.

How to get more data???

Key trick. Maximize what you have. From each trajectory of $K+1$ snapshots, use *every* ordered pair $(t_k, t_{\bar{k}})$ with $k \leq \bar{k}$ as a training example.

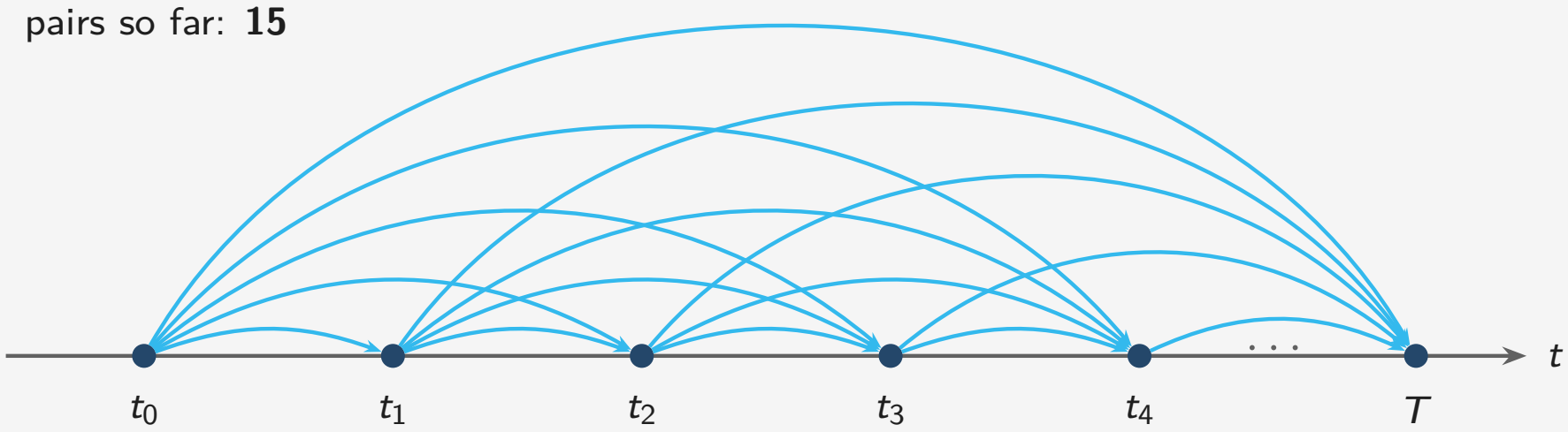


K snapshots, but $\binom{K+1}{2} \approx K^2/2$ training pairs. **Quadratic data multiplier, essentially for free.**

The catch. The input is $u(t_k)$, but the target depends on $\Delta t = t_{\bar{k}} - t_k$ — which varies across pairs. The model needs Δt as an input, not just u.

How to get more data???

Key trick. Maximize what you have. From each trajectory of $K+1$ snapshots, use *every* ordered pair $(t_k, t_{\bar{k}})$ with $k \leq \bar{k}$ as a training example.

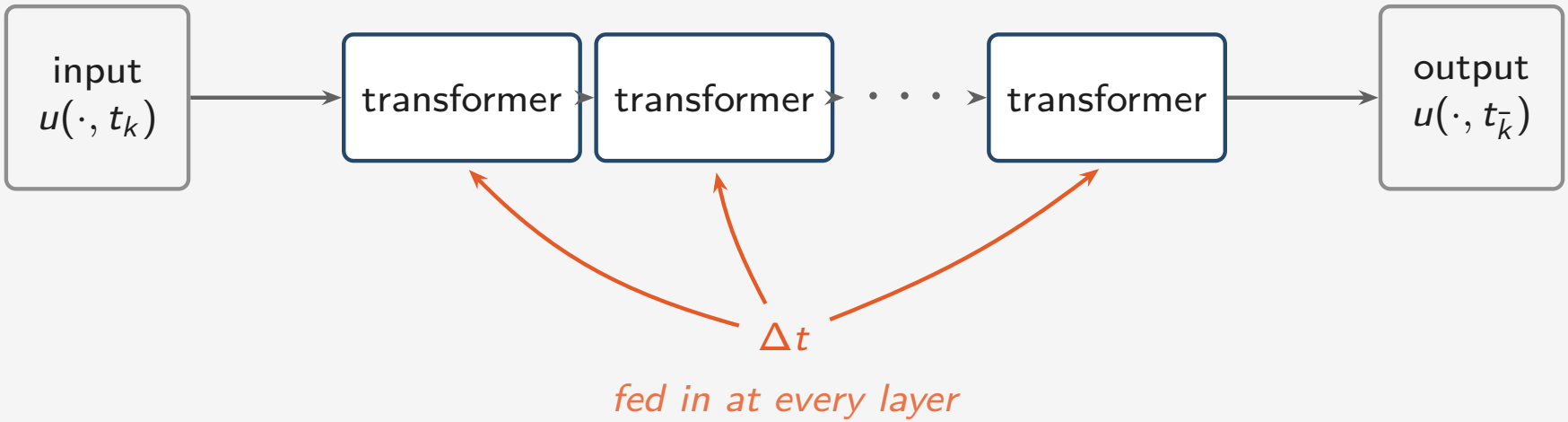


K snapshots, but $\binom{K+1}{2} \approx K^2/2$ training pairs. **Quadratic data multiplier, essentially for free.**

The catch. The input is $u(t_k)$, but the target depends on $\Delta t = t_{\bar{k}} - t_k$ — which varies across pairs. The model needs Δt as an input, not just u.

A simple architecture

A deep stack of transformer layers, applied to the input field.



The one detail that matters: Δt will enter as an input at every layer.

How is Δt actually fed in?

We said Δt enters at every layer. How?

The standard LayerNorm rescales activations by learnable parameters α, β:

$$\mathrm{LN}(v) \;=\; \alpha \odot \frac{v - \mu_v}{\sigma_v} + \beta$$

where μ_v, σ_v are the mean and standard deviation of v's channels.

Poseidon's twist: make α and β *functions of* Δt:

$$\alpha(\tau) = a\,\tau + \bar{a}, \qquad \beta(\tau) = b\,\tau + \bar{b}$$

A tiny affine map of $\tau = \Delta t$. Two extra vectors per layer.

Why not just concatenate Δt to the input?

That injects Δt once, at the start. Conditioning LayerNorm re-injects it at *every* layer — the model never has to "remember" the lead time through the depth of the network.

How to evaluate (and what happened)

The protocol. For each of 15 downstream tasks:

- Take the pretrained Poseidon. Fine-tune on N task-specific examples.
- Sweep N from 1 to $\sim$2000. Report test error vs. N.
- Compare against FNO trained from scratch on the same N.

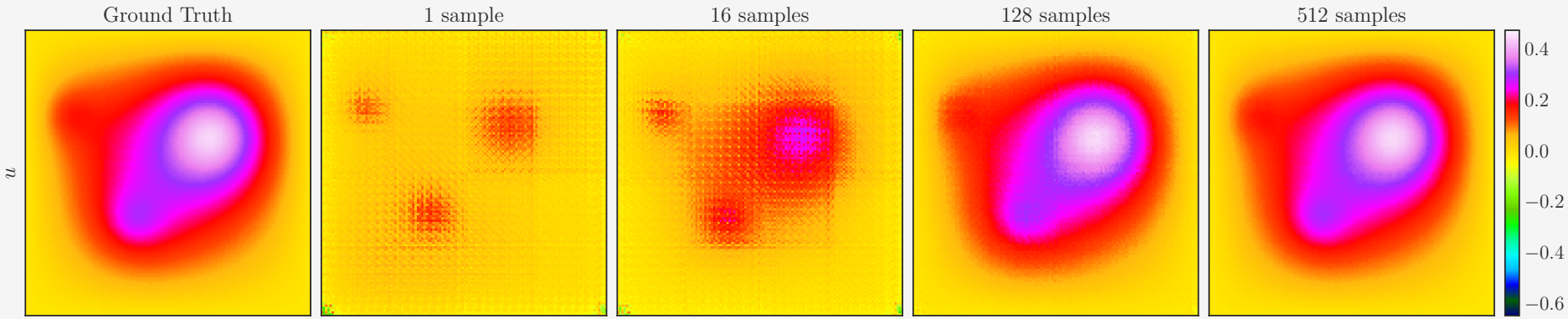
The 15 tasks span three tiers:

- 6 same PDE, new initial distributions
- 6 new physical effects (gravity, forcing, transport)
- 3 completely unrelated PDEs (Wave, Helmholtz, Poisson)

The headline. Median **20 samples** suffice for Poseidon-L to match FNO's accuracy at **1024** samples.

A $\sim 50\times$ **sample-efficiency gain**, even on PDEs Poseidon never saw during pretraining.

Example of performance increase: Elliptic-Poisson PDE



Lesson 1: scaling the model helps

Poseidon comes in three sizes: **T** (21M), **B** (158M), **L** (629M parameters).

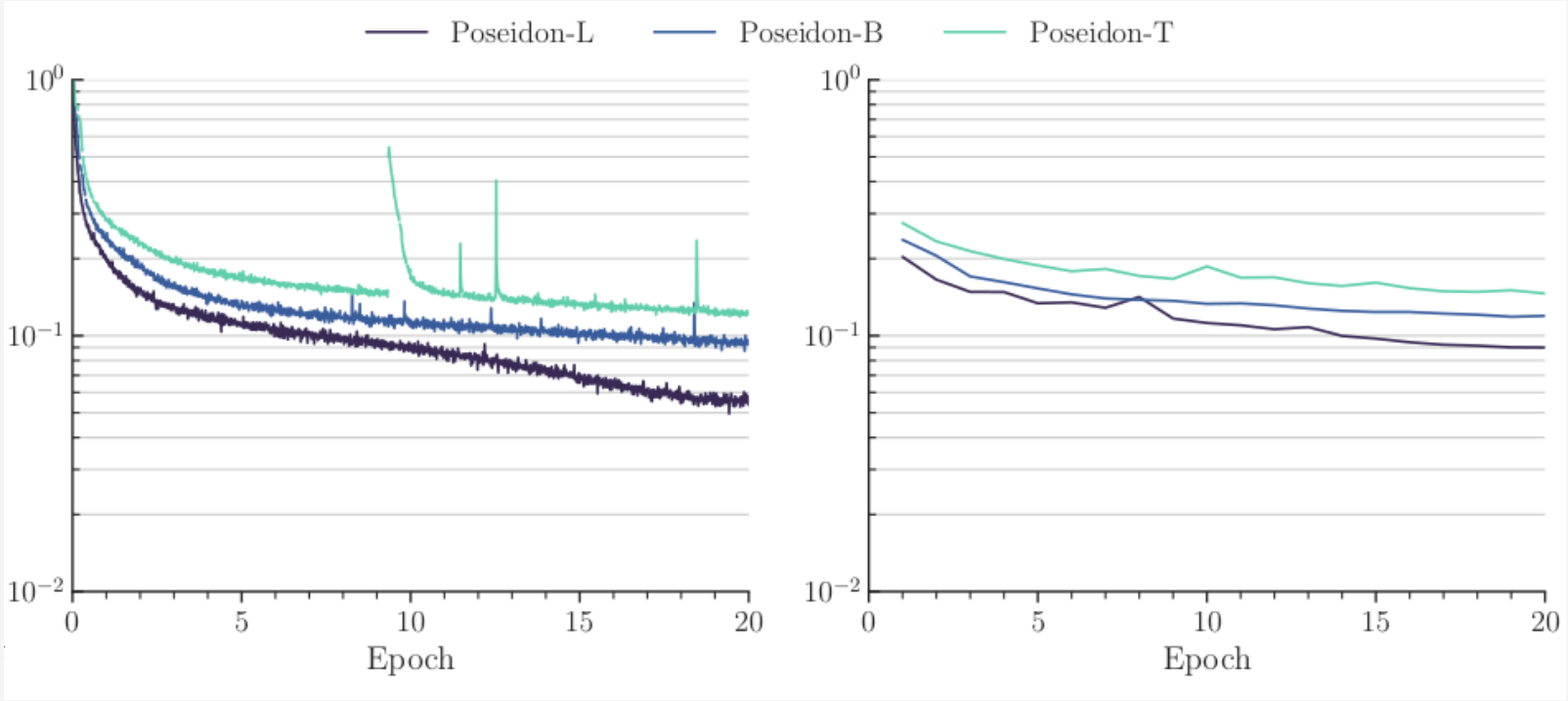


Figure: MSE training error and MSE testing error

Bigger model, lower loss. Nothing surprising here.

Lesson 2: Diversity matters more than dataset size

The ablation: train Poseidon-B on three pretraining datasets of *equal total size*, but different diversity.

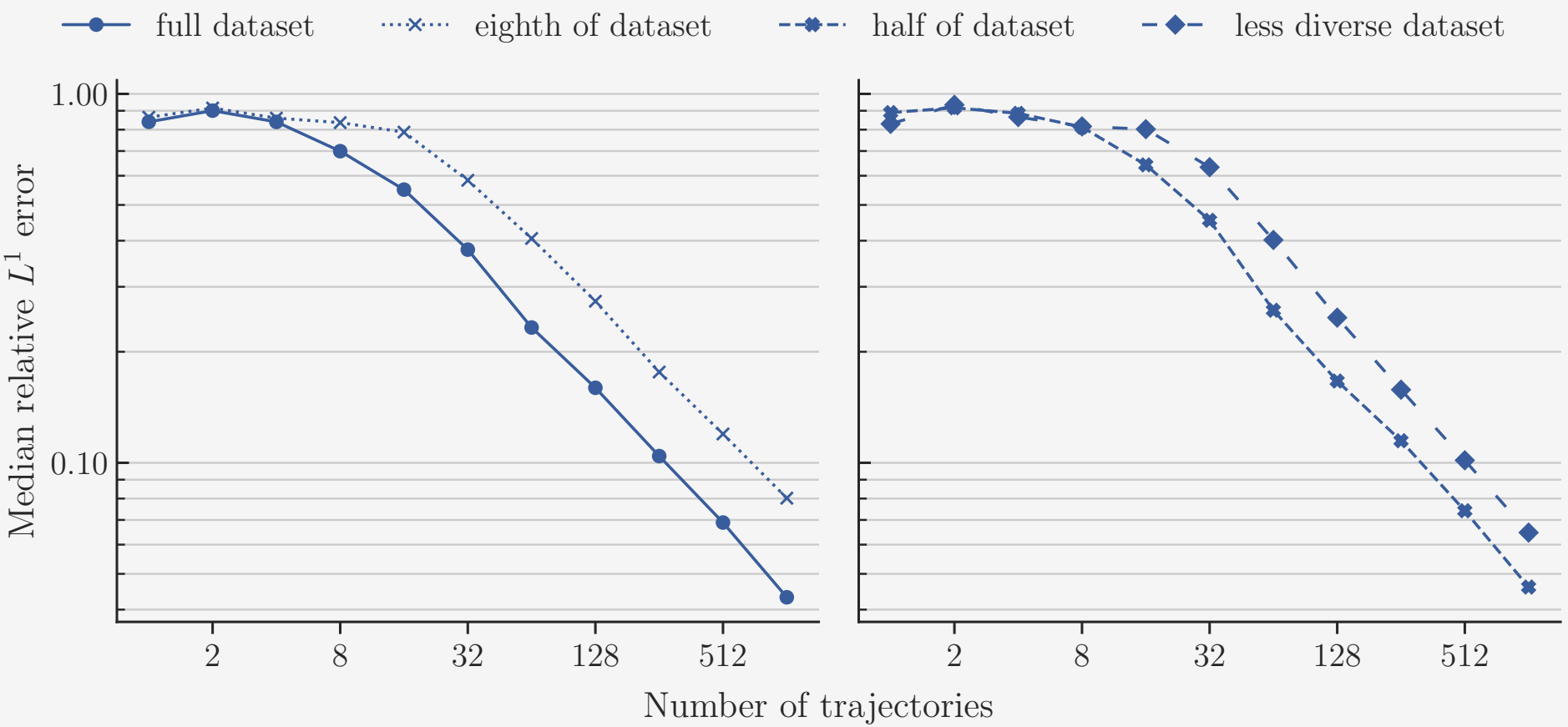


Figure: Testing error when trained with various datasets on downstream wave PDE

Lesson 3: fine-tuning is dramatically more sample-efficient

The actual payoff. For each downstream task: median relative L^1 error vs. number of task-specific fine-tuning samples.

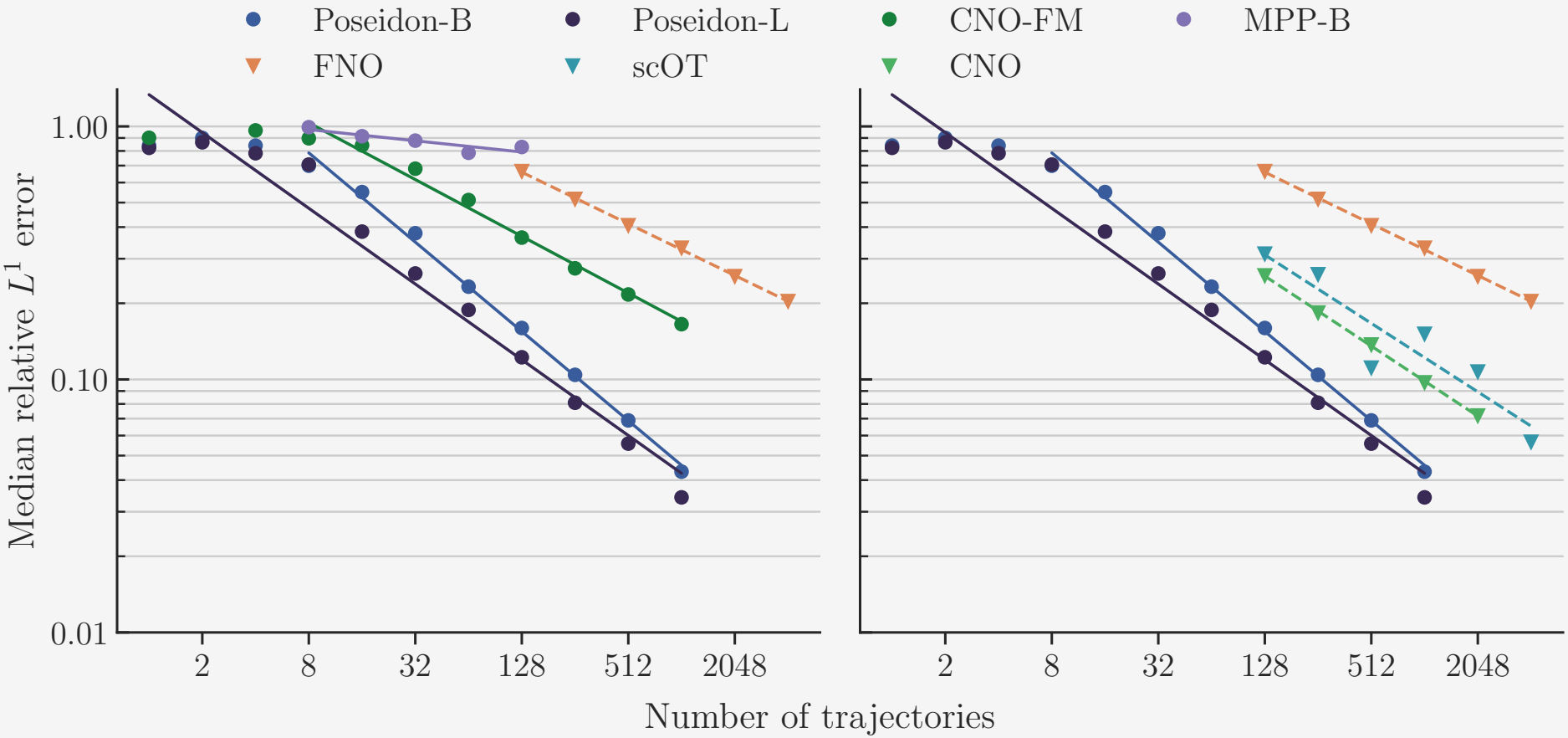


Figure: Testing error when trained with various models on downstream wave PDE

Design question for foundation models

Poseidon was trained on snapshot pairs $(t_k, t_{\bar{k}})$ with $\Delta t = t_{\bar{k}} - t_k$ drawn from inside a trajectory.

At inference, a user gives you $u(\cdot, 0)$ and asks for $u(\cdot, T)$ where T is *much larger* than any Δt seen during training.

Discuss with the person next to you for 90 seconds.

What goes wrong?

How would you fix it?

Large Δt: out of distribution

What goes wrong. The pair $(u(\cdot, 0), \Delta t = T)$ is unlike anything Poseidon saw during training. Quality degrades.

The fix: autoregressive rollout. Chain κ small steps, each with a Δt inside the training distribution:

$$u(\cdot, 0) \;\to\; u_1 \;\to\; u_2 \;\to\; \cdots \;\to\; u_\kappa \;\approx\; u(\cdot, T)$$

Pros

- Each step stays in-distribution.
- No retraining needed.

Cons

- Errors compound across steps.
- κ times the inference cost.

Comparison on training with large T versus autoregressive design

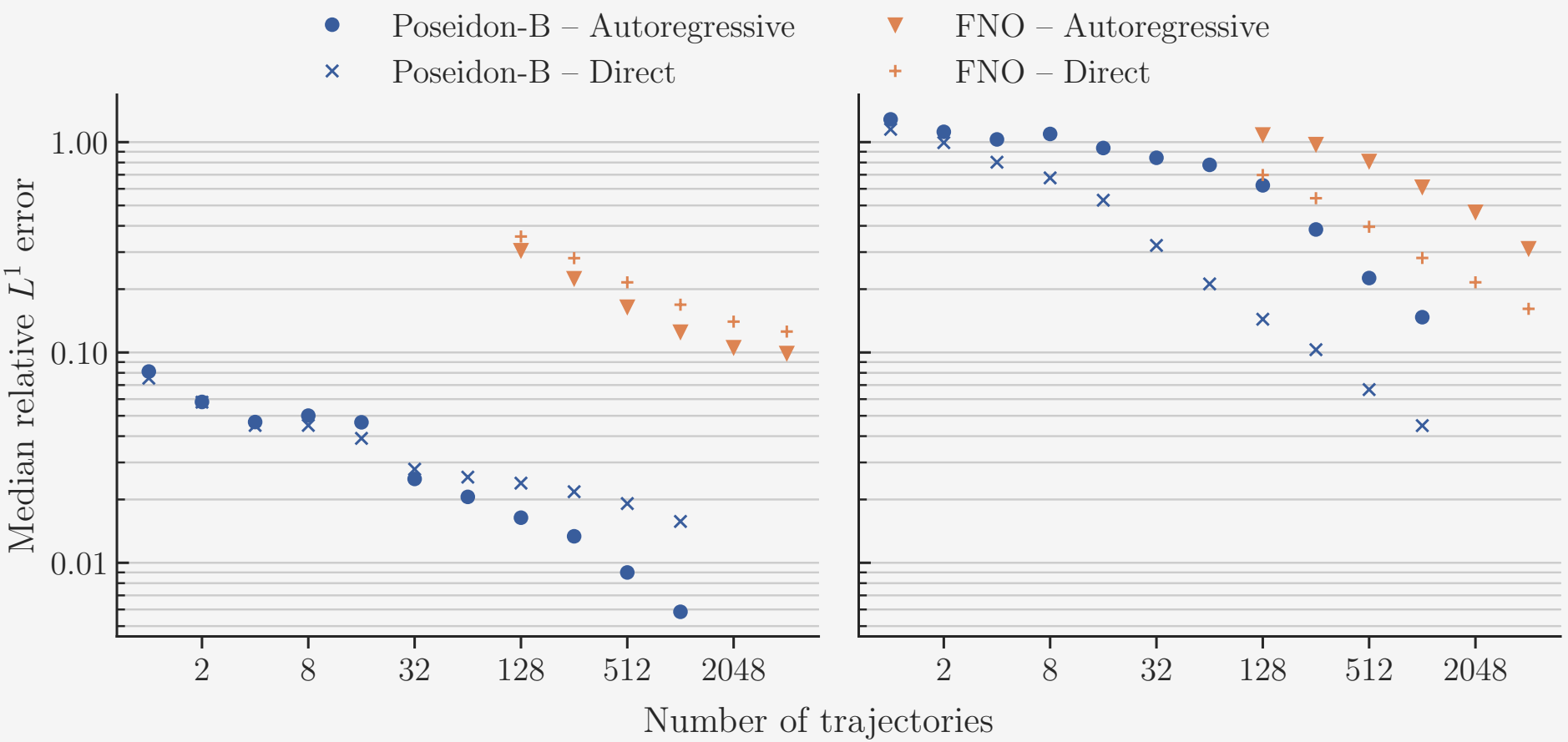


Figure: L_1 error compared from direct and autoregressive design

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

Three lessons:

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

Three lessons:

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

- **Pretrain on fluid dynamics only** diverse, but narrow.

Three lessons:

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

- **Pretrain on fluid dynamics only** diverse, but narrow.
- **More data with a trick** every snapshot pair is a training example.

Three lessons:

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

- **Pretrain on fluid dynamics only** diverse, but narrow.
- **More data with a trick** every snapshot pair is a training example.
- **Transformer with Δt-conditioned LayerNorm** time enters at every layer.

Three lessons:

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

- **Pretrain on fluid dynamics only** diverse, but narrow.
- **More data with a trick** every snapshot pair is a training example.
- **Transformer with Δt-conditioned LayerNorm** time enters at every layer.
- **Fine-tune on each downstream task** N task-specific samples, N small.

Three lessons:

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

- **Pretrain on fluid dynamics only** diverse, but narrow.
- **More data with a trick** every snapshot pair is a training example.
- **Transformer with Δt-conditioned LayerNorm** time enters at every layer.
- **Fine-tune on each downstream task** N task-specific samples, N small.

Three lessons:

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

- **Pretrain on fluid dynamics only** diverse, but narrow.
- **More data with a trick** every snapshot pair is a training example.
- **Transformer with Δt-conditioned LayerNorm** time enters at every layer.
- **Fine-tune on each downstream task** N task-specific samples, N small.

Three lessons:

- Scaling the model helps — but not by as much as you'd hope.

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

- **Pretrain on fluid dynamics only** diverse, but narrow.
- **More data with a trick** every snapshot pair is a training example.
- **Transformer with Δt-conditioned LayerNorm** time enters at every layer.
- **Fine-tune on each downstream task** N task-specific samples, N small.

Three lessons:

- Scaling the model helps — but not by as much as you'd hope.
- **Diversity matters more than dataset size.**

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

- **Pretrain on fluid dynamics only** diverse, but narrow.
- **More data with a trick** every snapshot pair is a training example.
- **Transformer with Δt-conditioned LayerNorm** time enters at every layer.
- **Fine-tune on each downstream task** N task-specific samples, N small.

Three lessons:

- Scaling the model helps — but not by as much as you'd hope.
- **Diversity matters more than dataset size.**
- Fine-tuning is $\sim 50\times$ more sample-efficient than training from scratch.

Poseidon Case Study: Recap

The model. A transformer trained on 6 fluid-dynamics operators, evaluated on 15 PDE tasks — 9 of which involve physics it never saw.

Four design choices that mattered:

- **Pretrain on fluid dynamics only** diverse, but narrow.
- **More data with a trick** every snapshot pair is a training example.
- **Transformer with Δt-conditioned LayerNorm** time enters at every layer.
- **Fine-tune on each downstream task** N task-specific samples, N small.

Three lessons:

- Scaling the model helps — but not by as much as you'd hope.
- **Diversity matters more than dataset size.**
- Fine-tuning is $\sim 50\times$ more sample-efficient than training from scratch.

The surprise: a model trained on Euler and Navier–Stokes learned representations that transferred to Poisson, Helmholtz, and Wave equations — physics it never saw.

A more modern design (Walrus)

Same approach as Poseidon, but the key is to emphasize diversity.

A more modern design (Walrus)

Same approach as Poseidon, but the key is to emphasize diversity.

- **1.3B parameter** space-time transformer.

A more modern design (Walrus)

Same approach as Poseidon, but the key is to emphasize diversity.

- **1.3B parameter** space-time transformer.
- Pretrained on **19 physical domains**:
  astrophysics (MHD turbulence, supernovae), fluid dynamics, active matter, biology *with varying domains*.

A more modern design (Walrus)

Same approach as Poseidon, but the key is to emphasize diversity.

- **1.3B parameter** space-time transformer.
- Pretrained on **19 physical domains**:
  astrophysics (MHD turbulence, supernovae), fluid dynamics, active matter, biology *with varying domains*.
- Predicts **state changes**, not absolute state — mirrors residual learning, echoes Neural ODEs from Lecture 12.

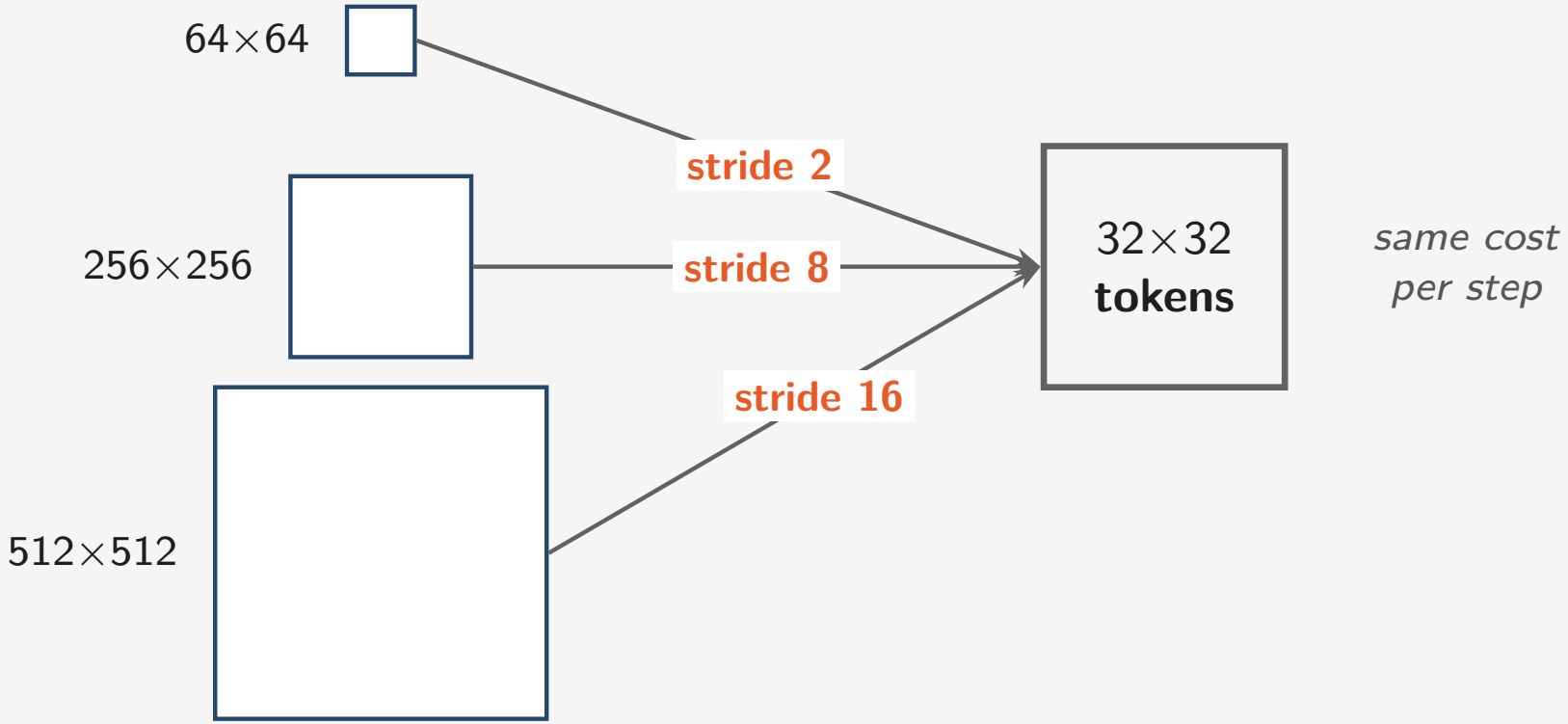
A more modern design (Walrus)

Same approach as Poseidon, but the key is to emphasize diversity.

- **1.3B parameter** space-time transformer.
- Pretrained on **19 physical domains**:
  astrophysics (MHD turbulence, supernovae), fluid dynamics, active matter, biology *with varying domains*.
- Predicts **state changes**, not absolute state — mirrors residual learning, echoes Neural ODEs from Lecture 12.
- New design called convolutional stride modulation for discretization invariance.

Multi-resolutions: convolutional stride modulation (CSM)

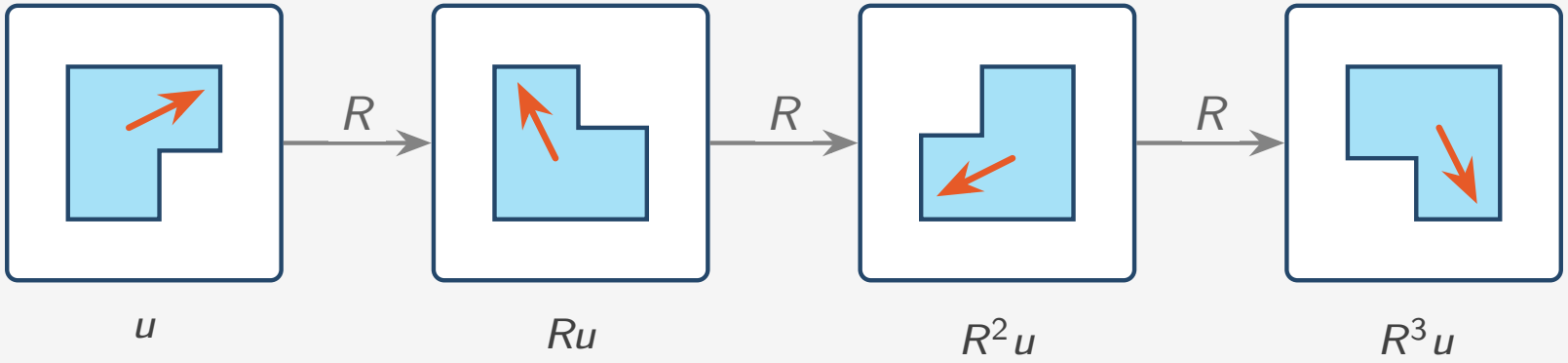
The problem. Walrus's 19 pretraining datasets have native resolutions from 64^3 to 1024×256. With a *fixed* patch size, a 512^2 snapshot produces $\sim 16\times$ more tokens than a 128^2 snapshot. How to resolve?



The fix. Choose convolution kernel and stride *per dataset* so every input lands at the same target token count (32^2 for 2D), regardless of native resolution.

How to get more data?

The trick. Physics is invariant under rotations of the frame. Rotate a fluid simulation 90° — still valid, fresh example.



The catch. Tensor fields must rotate *with* the frame: $\boldsymbol{u} \to R\boldsymbol{u}$ for vectors, $\boldsymbol{T} \to R\boldsymbol{T}R^\top$ for stresses. Naive image augmentation silently breaks every vector field.

Bonus: pad 2D snapshots with a singleton z-axis and apply the *same* 3D rotation group — 2D data gets the geometric diversity of 3D for free.

Example performance: Advection example

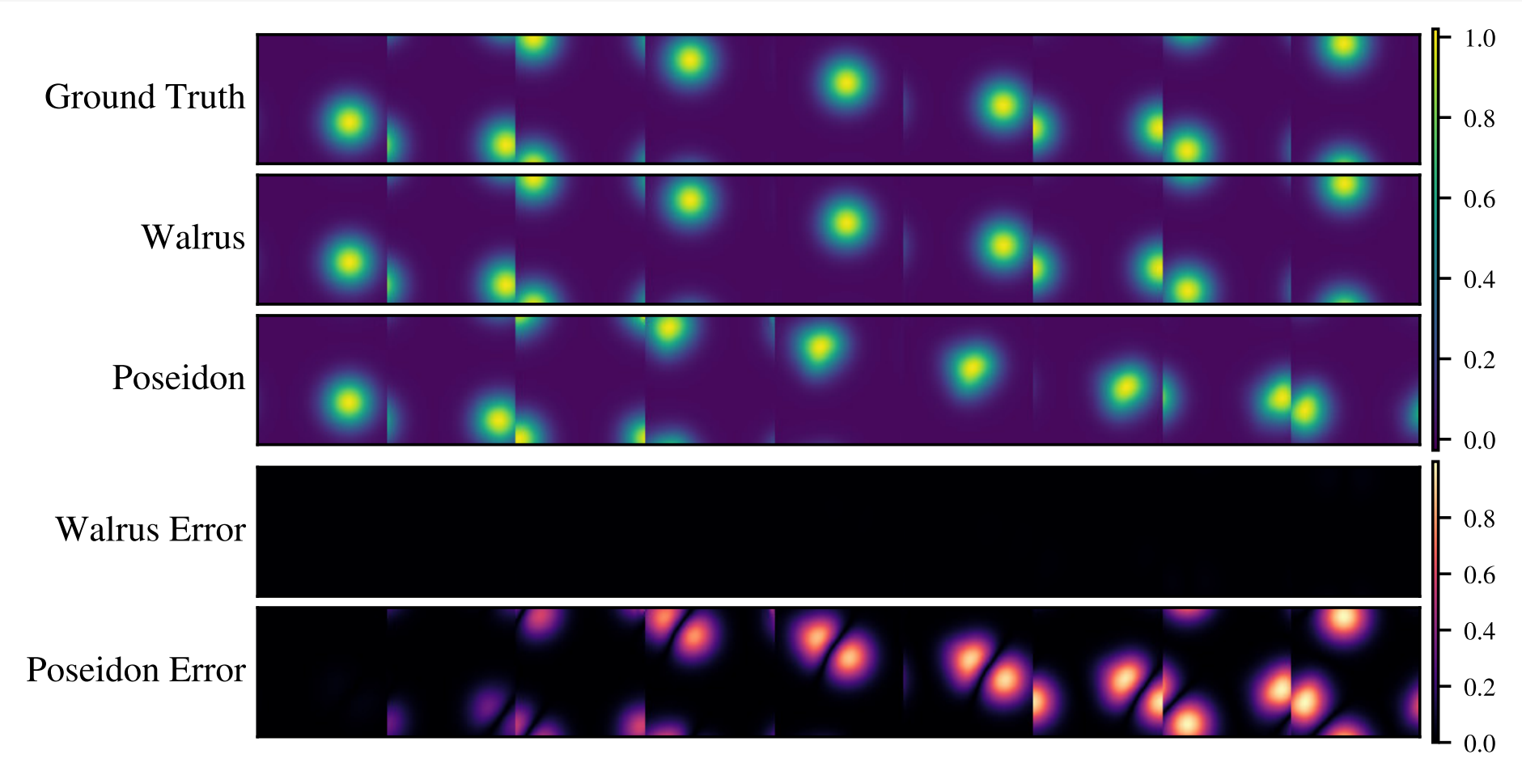


Figure: Comparison on simple advection-diffusion PDE as a downstream task.

Example performance: Quantitative

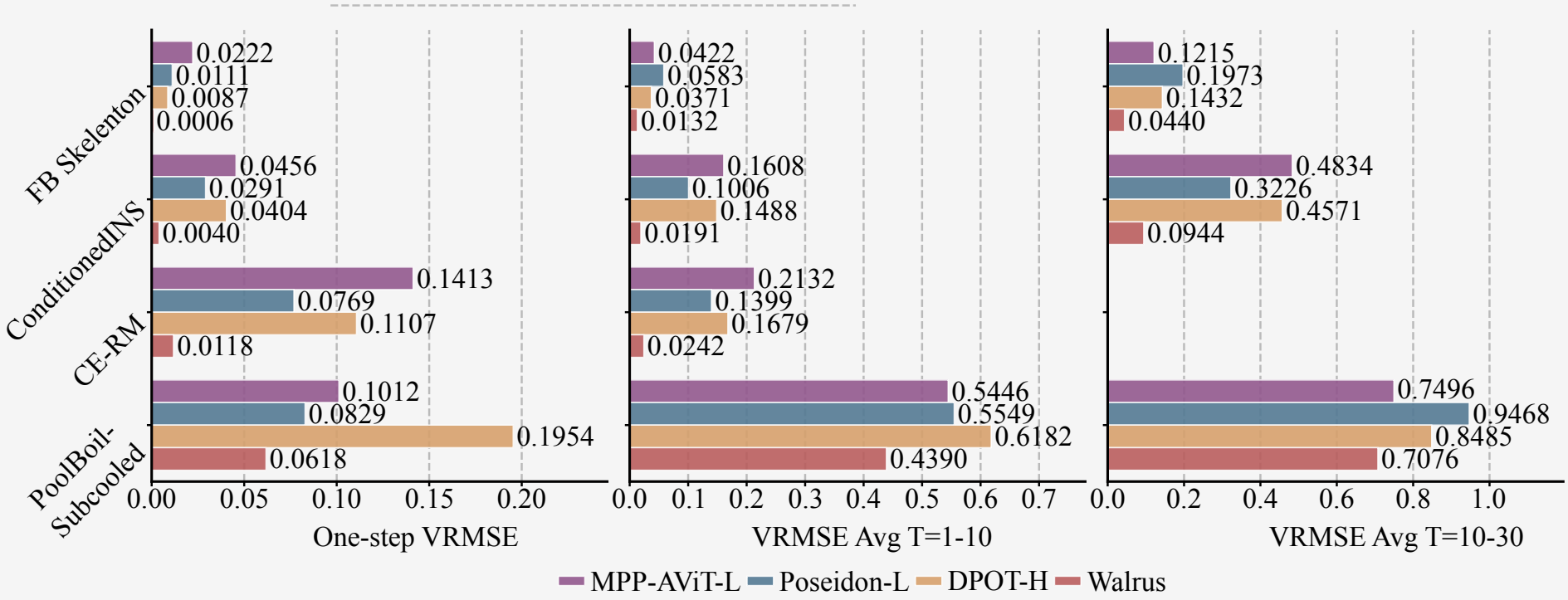


Figure: Comparison across various downstream PDE tasks

Emerging themes: Walrus and Poseidon

- **Squeeze more supervision from scarce data.** Poseidon: all-to-all time pairing. Walrus: rotation-equivariant augmentation. Both invent data multipliers essentially for free.

Emerging themes: Walrus and Poseidon

- **Squeeze more supervision from scarce data.** Poseidon: all-to-all time pairing. Walrus: rotation-equivariant augmentation. Both invent data multipliers essentially for free.
- **Transformers scale, but need strong physics priors.** Δt-conditioned LayerNorm (Poseidon), convolutional stride modulation (Walrus).

Emerging themes: Walrus and Poseidon

- **Squeeze more supervision from scarce data.** Poseidon: all-to-all time pairing. Walrus: rotation-equivariant augmentation. Both invent data multipliers essentially for free.
- **Transformers scale, but need strong physics priors.** Δt-conditioned LayerNorm (Poseidon), convolutional stride modulation (Walrus).
- **Transfer to unseen physics does work.** Fluids $\rightarrow$ Poisson / Helmholtz / Wave. Astrophysics $\rightarrow$ biology. Pretraining on one regime buys you another.

Scaling laws for pretraining in PDEs

Poseidon and Walrus showed foundation models for SciML *work*. Two questions we haven't answered:

An old 2023 paper sweeps these knobs independently on three classical 2D PDEs — *Poisson*, *Advection–Diffusion*, *Helmholtz* — using an FNO backbone.

Scaling laws for pretraining in PDEs

Poseidon and Walrus showed foundation models for SciML *work*. Two questions we haven't answered:

- How much of the win comes from **pretraining**, from **model size**, from **data diversity**?

An old 2023 paper sweeps these knobs independently on three classical 2D PDEs — *Poisson*, *Advection–Diffusion*, *Helmholtz* — using an FNO backbone.

Scaling laws for pretraining in PDEs

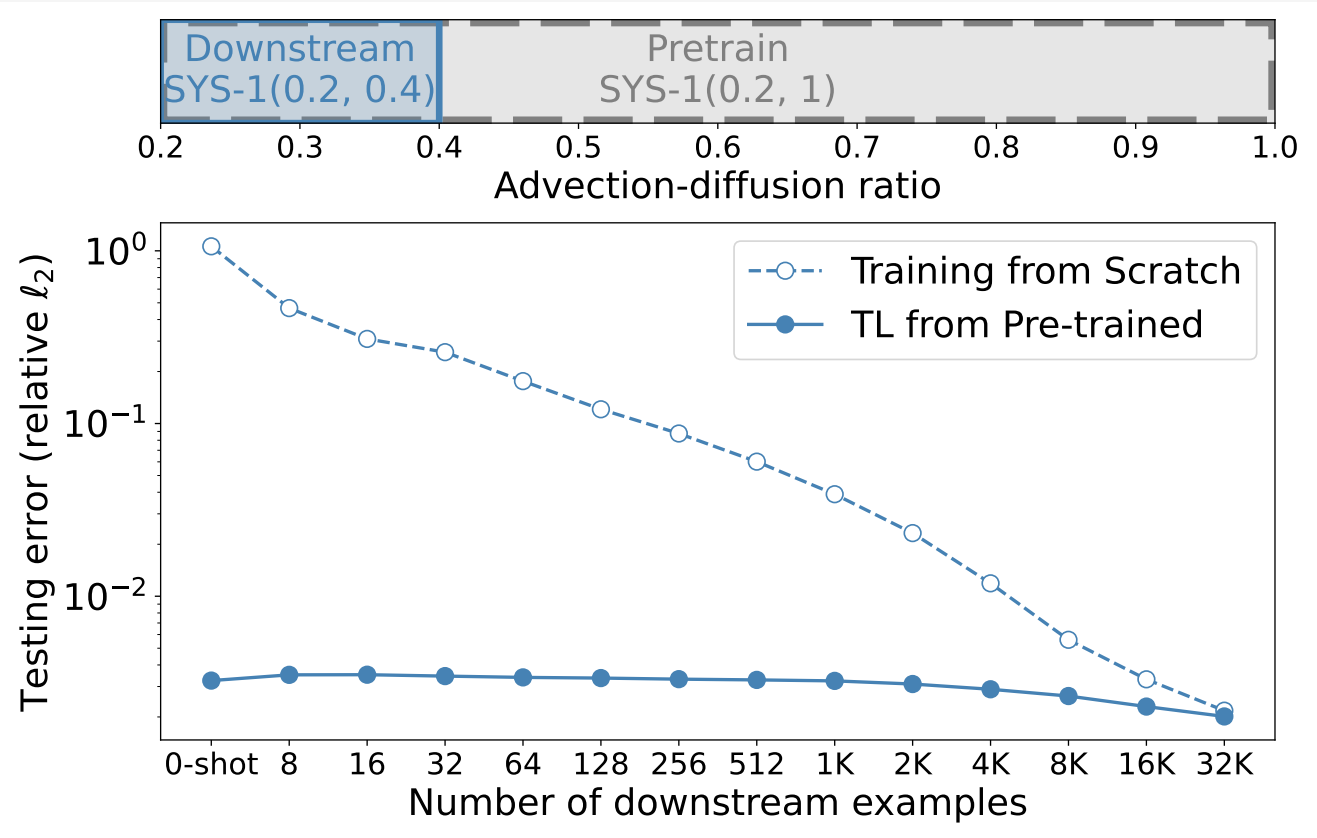
Poseidon and Walrus showed foundation models for SciML *work*. Two questions we haven't answered:

- How much of the win comes from **pretraining**, from **model size**, from **data diversity**?
- When does pretraining **stop helping**?

An old 2023 paper sweeps these knobs independently on three classical 2D PDEs — *Poisson*, *Advection–Diffusion*, *Helmholtz* — using an FNO backbone.

Q1: How much does fine-tuning data matter?

Sweep the number of fine-tuning examples from 0 (zero-shot) to $32K$. Compare fine-tuning the pretrained FNO vs. training from scratch.



To reach 10^{-2} error: pretraining needs **64** examples; scratch needs **8K**. A $\sim 100\times$ **data savings.**

Q2: How much does model size matter?

Sweep the FNO from $64K$ to $256M$ parameters — a $4000\times$ range. Repeat the data-scaling experiment at each size.

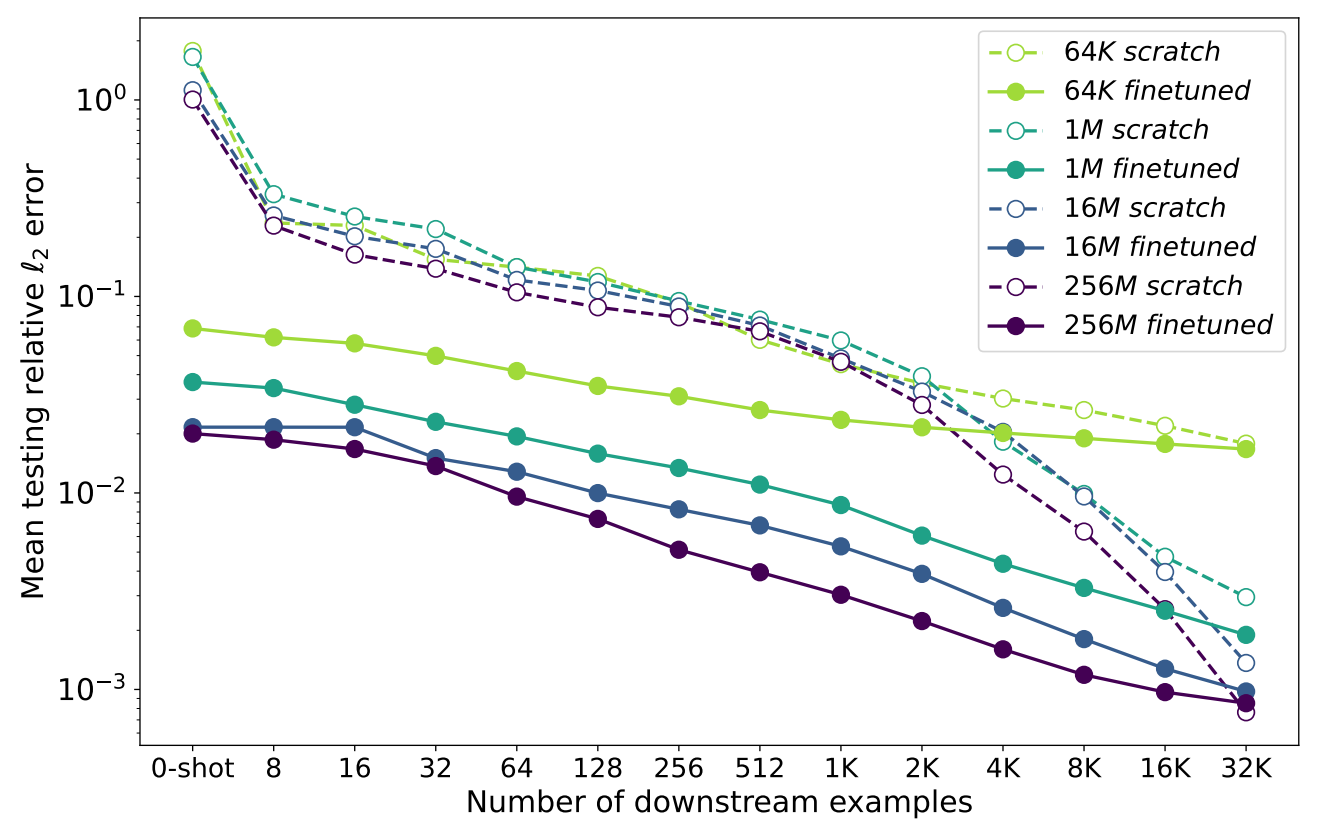
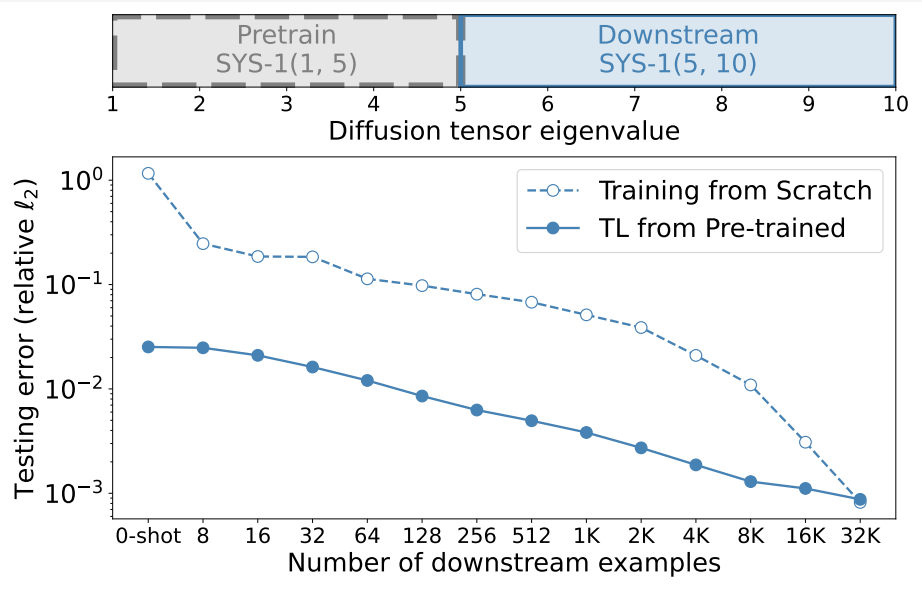
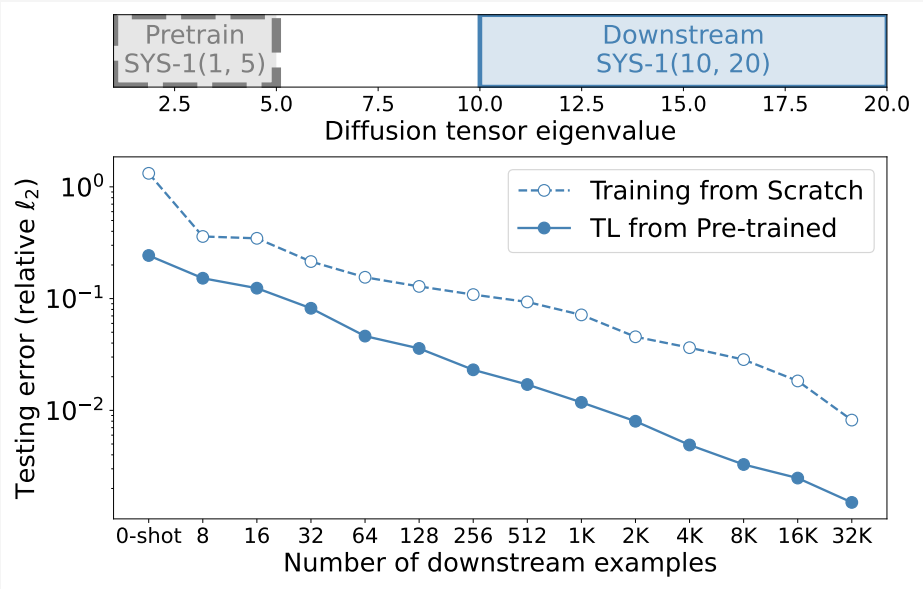


Figure: Scaling on Poisson PDE (Weaker than the scaling laws in NLP).

Q3: How far OOD can you push it?

Define two levels of distribution shift by sliding the downstream PDE coefficients away from the pretraining range:





Foundation models for PDEs: Takeaways

Key takeaway: Scale, data, and architecture design is not as easy in physical systems like it is in NLP! One needs to think.

Foundation models for PDEs: Takeaways

1. **Data is scarce, so reuse matters:** Poseidon creates more supervision with all-to-all time pairing, Walrus uses the rotation trick, and pretraining can improve sample efficiency by roughly 100×.

Key takeaway: Scale, data, and architecture design is not as easy in physical systems like it is in NLP! One needs to think.

Foundation models for PDEs: Takeaways

1. **Data is scarce, so reuse matters:** Poseidon creates more supervision with all-to-all time pairing, Walrus uses the rotation trick, and pretraining can improve sample efficiency by roughly $100\times$.
2. **Scaling helps, but less than in NLP:** PDE scaling laws are shallower and task-dependent, so pretraining mostly shifts error curves downward rather than changing their slope.

Key takeaway: Scale, data, and architecture design is not as easy in physical systems like it is in NLP! One needs to think.

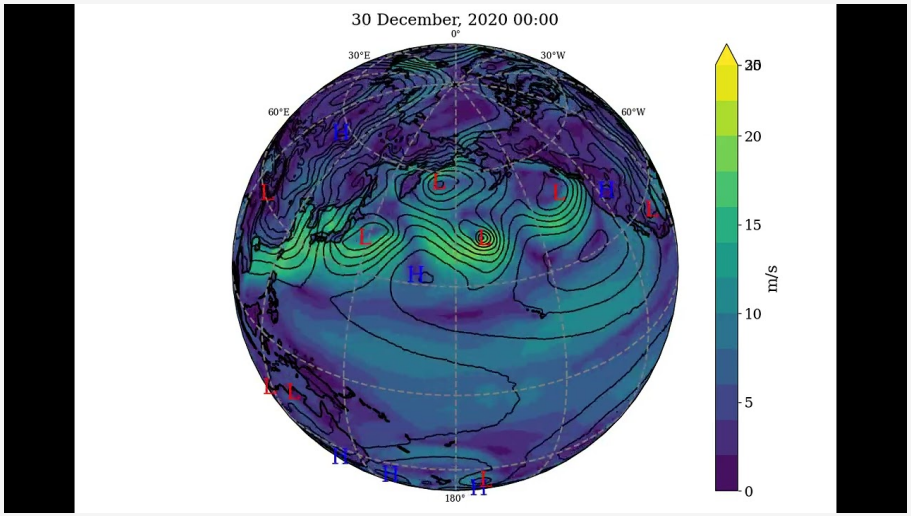
Foundation models for PDEs: Takeaways

1. **Data is scarce, so reuse matters:** Poseidon creates more supervision with all-to-all time pairing, Walrus uses the rotation trick, and pretraining can improve sample efficiency by roughly 100×.
2. **Scaling helps, but less than in NLP:** PDE scaling laws are shallower and task-dependent, so pretraining mostly shifts error curves downward rather than changing their slope.
3. **Architecture matters:** PDE models need to handle many spatial and temporal scales, which is why hierarchical, multi-scale designs need to *consider the physics*.

Key takeaway: Scale, data, and architecture design is not as easy in physical systems like it is in NLP! One needs to think.

A real world example

Weather forecasting is one of the largest scientific computing problems in the world.

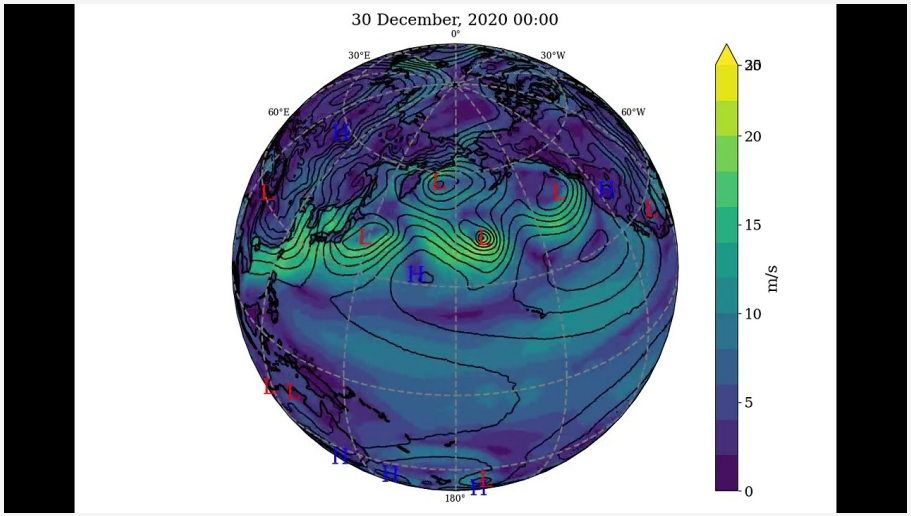


A global wind field forecast.

A real world example

Weather forecasting is one of the largest scientific computing problems in the world.

- **The physics.** Solve atmospheric dynamics on a global grid. ECMWF's IFS runs at ∼9 km, four times daily.

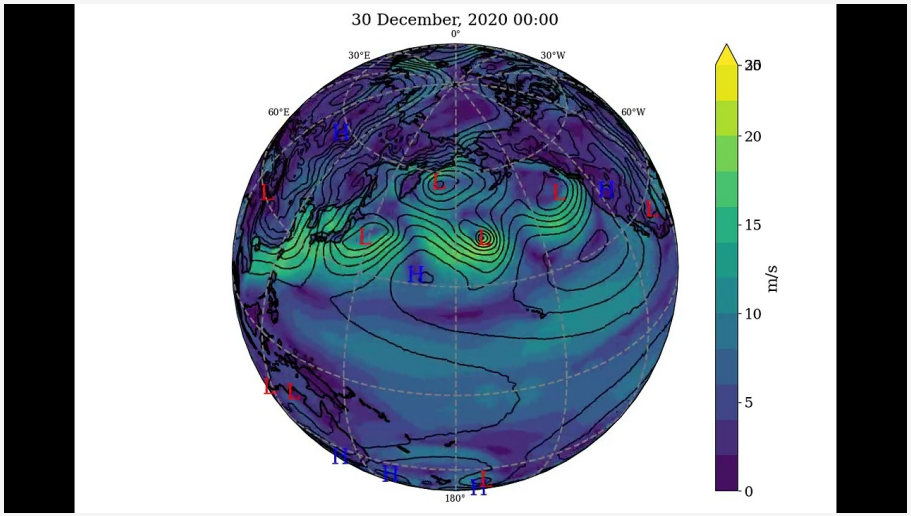


A global wind field forecast.

A real world example

Weather forecasting is one of the largest scientific computing problems in the world.

- **The physics.** Solve atmospheric dynamics on a global grid. ECMWF's IFS runs at ∼9 km, four times daily.
- **The cost.** Each cycle takes **hours on a supercomputer.** Major agencies (ECMWF, NOAA, JMA) run dedicated HPC facilities.

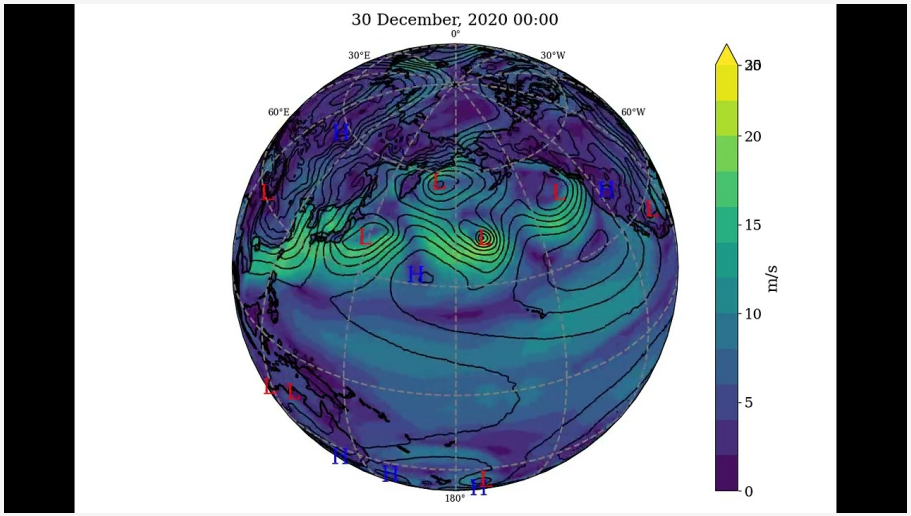


A global wind field forecast.

A real world example

Weather forecasting is one of the largest scientific computing problems in the world.

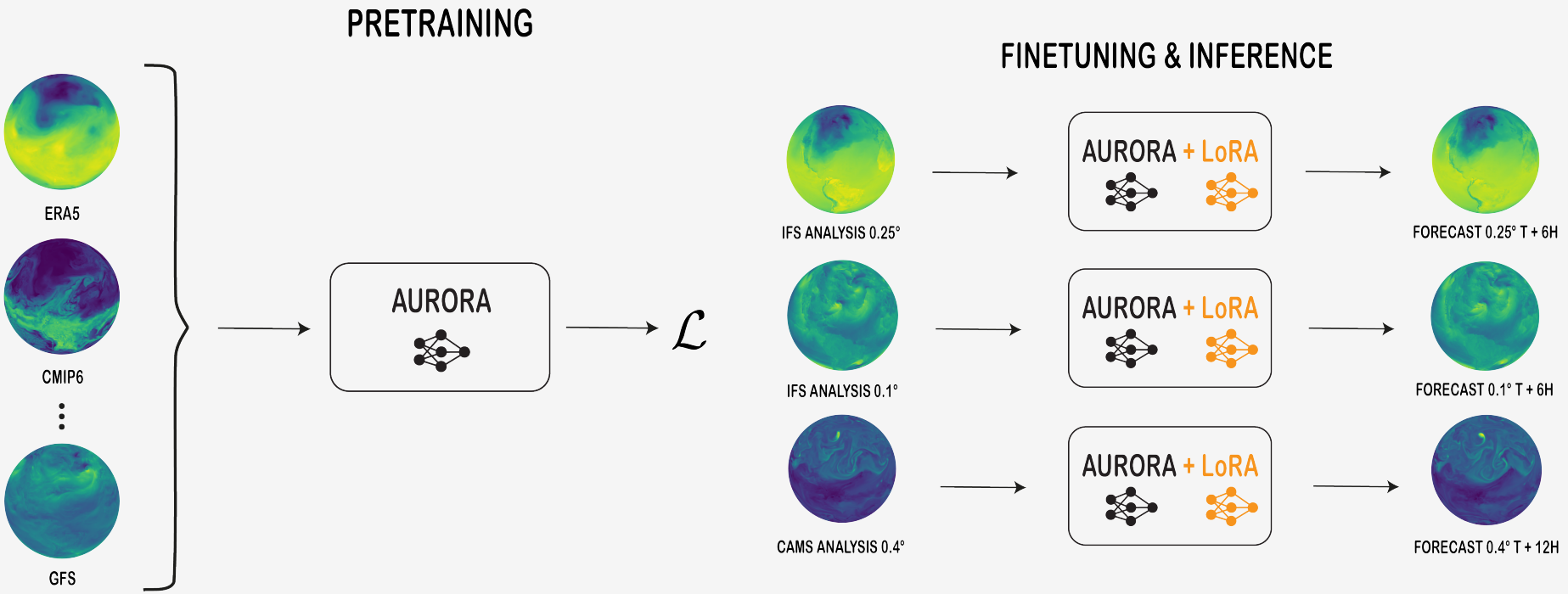
- **The physics.** Solve atmospheric dynamics on a global grid. ECMWF's IFS runs at ∼9 km, four times daily.
- **The cost.** Each cycle takes **hours on a supercomputer.** Major agencies (ECMWF, NOAA, JMA) run dedicated HPC facilities.
- **Why it matters.** Better forecasts save lives (hurricanes, floods) and billions of dollars (agriculture, aviation, energy).



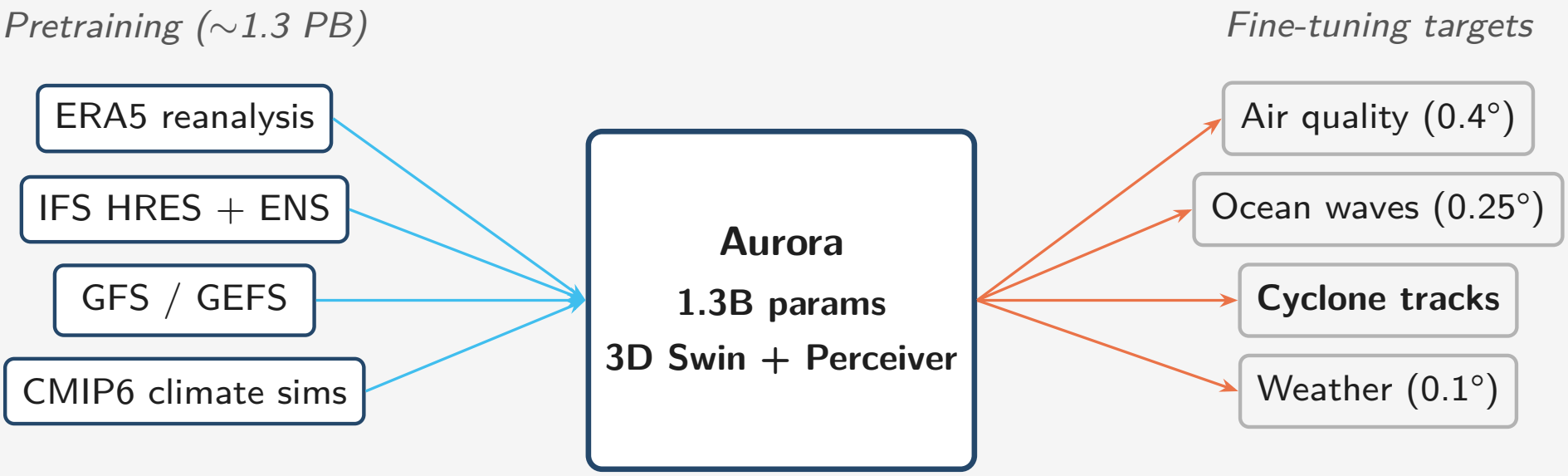
A global wind field forecast.

Aurora (paper). Nature, Microsoft AI

A 1.3B-parameter foundation model trained on **over 1 million hours** of heterogeneous Earth-system data — then fine-tuned for four operational tasks.



Aurora: pretraining corpus and downstream tasks



Pretraining: diverse by design.

- **Reanalysis** (ERA5): best estimate of past weather.
- **Operational forecasts** (IFS, GFS) + ensembles.
- **Climate simulations** (CMIP6): exposure to extremes not in the historical record.

∼1M+ hours of Earth-system data, 10+ heterogeneous sources.

Fine-tuning: four operational tasks.

- **Air quality** — 5 pollutants + particulates.
- **Ocean waves** — height, period, direction.
- **Cyclone tracks** — global, all storms.
- **High-res weather** — 0.1° (the IFS HRES grid).

The headline result: tropical cyclone tracks

Aurora beats the official forecast from **7 operational agencies** (NHC, JTWC, JMA, CMA, . . .) on **100%** of lead times, across all tropical cyclones globally in 2022–2023.

First time a single AI model has done this.

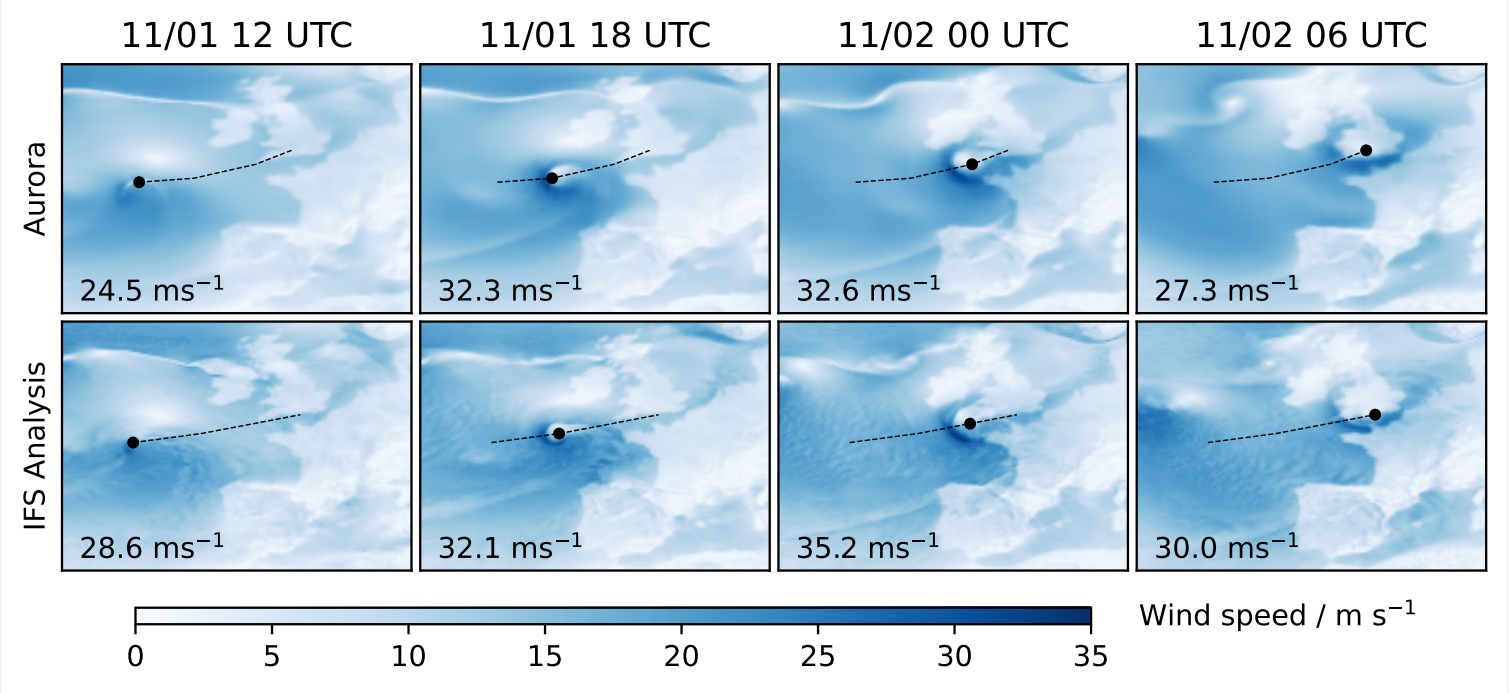


Figure: Example of Aurora and Ground truth on the Ciarán cyclone.

Where this leaves us

We started with a question: *can we have a "GPT for PDEs"?*

Four case studies later, the same recipe keeps showing up:

Aurora is the first foundation model for physics that's genuinely operational. But the recipe is fragile and the theory is thin.

The field is very open.

Where this leaves us

We started with a question: *can we have a "GPT for PDEs"?*

Four case studies later, the same recipe keeps showing up:

Aurora is the first foundation model for physics that's genuinely operational. But the recipe is fragile and the theory is thin.

The field is very open.

Where this leaves us

We started with a question: *can we have a "GPT for PDEs"?*

Four case studies later, the same recipe keeps showing up:

- **Diversity beats size.** (in the pretraining dataset)

Aurora is the first foundation model for physics that's genuinely operational. But the recipe is fragile and the theory is thin.

The field is very open.

Where this leaves us

We started with a question: *can we have a "GPT for PDEs"?*

Four case studies later, the same recipe keeps showing up:

- **Diversity beats size.** (in the pretraining dataset)
- **Transfer is does work** fluids help with Poisson, atmosphere helps with ocean waves.

Aurora is the first foundation model for physics that's genuinely operational. But the recipe is fragile and the theory is thin.

The field is very open.

Where this leaves us

We started with a question: *can we have a "GPT for PDEs"?*

Four case studies later, the same recipe keeps showing up:

- **Diversity beats size.** (in the pretraining dataset)
- **Transfer is does work** fluids help with Poisson, atmosphere helps with ocean waves.
- **Architecture has to be modified to the data** Scale alone does not solve everything.

Aurora is the first foundation model for physics that's genuinely operational. But the recipe is fragile and the theory is thin.

The field is very open.