# The Bitter Lesson for the Bitter Lesson:

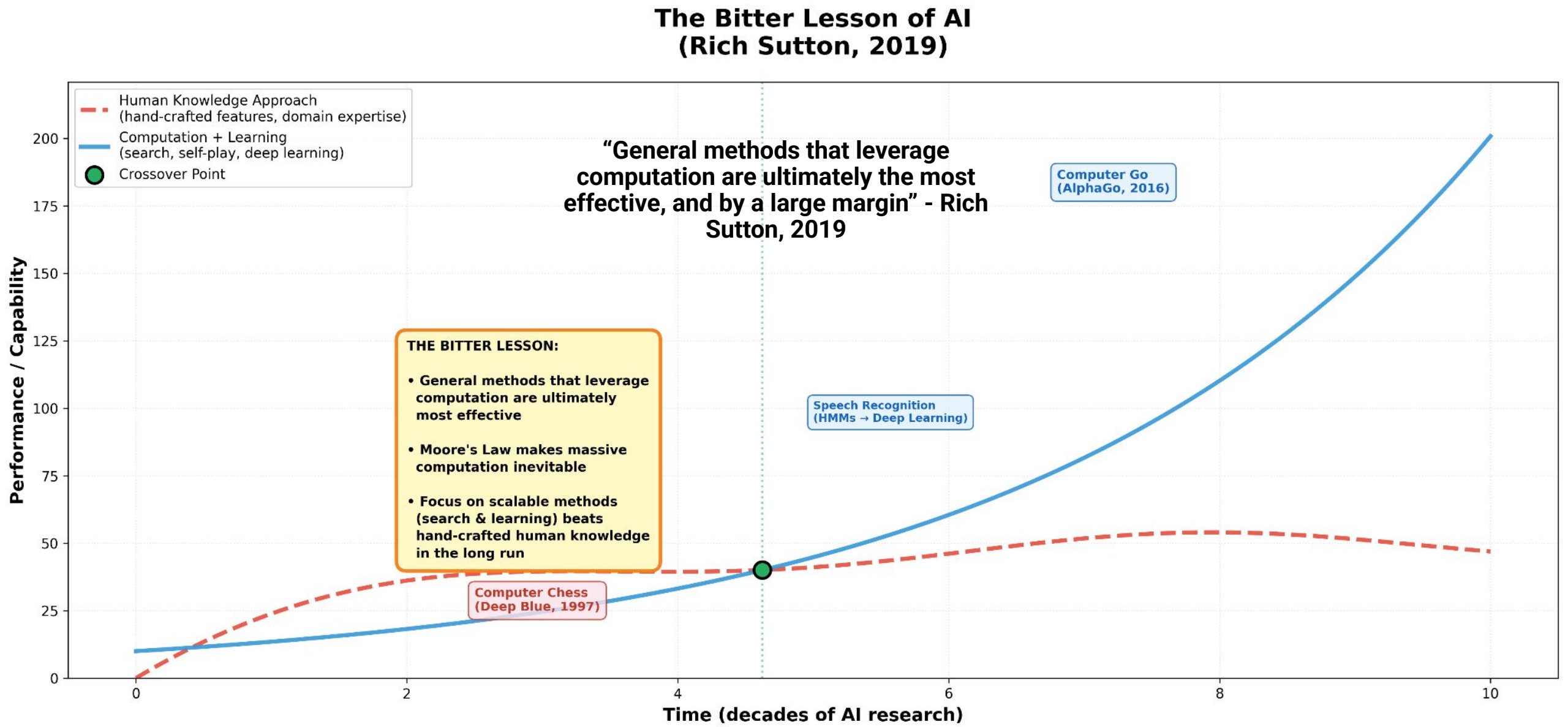
# Where Human Mathematicians and Engineers go in the Age of AI

**Chris Rackauckas**

VP of Modeling and Simulation,
*JuliaHub*

Research Affiliate, Co-PI of Julia Lab,
*Massachusetts Institute of Technology, CSAIL*

Director of Scientific Research,
*Pumas-AI*


The Bitter Lesson of AI
(Rich Sutton, 2019)
Human Knowledge Approach
(hand-crafted features, domain expertise)
Computation + Learning
(search, self-play, deep learning)
Crossover Point
"General methods that leverage computation are ultimately the most effective, and by a large margin" - Rich Sutton, 2019
Computer Go
(AlphaGo, 2016)
THE BITTER LESSON:
• General methods that leverage computation are ultimately most effective
• Moore's Law makes massive computation inevitable
• Focus on scalable methods (search & learning) beats hand-crafted human knowledge in the long run
Speech Recognition
(HMMs → Deep Learning)
Computer Chess
(Deep Blue, 1997)
Performance / Capability
0
25
50
75
100
125
150
175
200
0
2
4
6
8
10
Time (decades of AI research)

**Looking more closely at the “AI successes” may suggest the bitter lesson needs to learn a lesson.**

# Potential Problem: Out of Sample Data (Generalization Issues)


**Traffic light, or moon?**

**“It’s only an edge case that happens on planets with moons”**

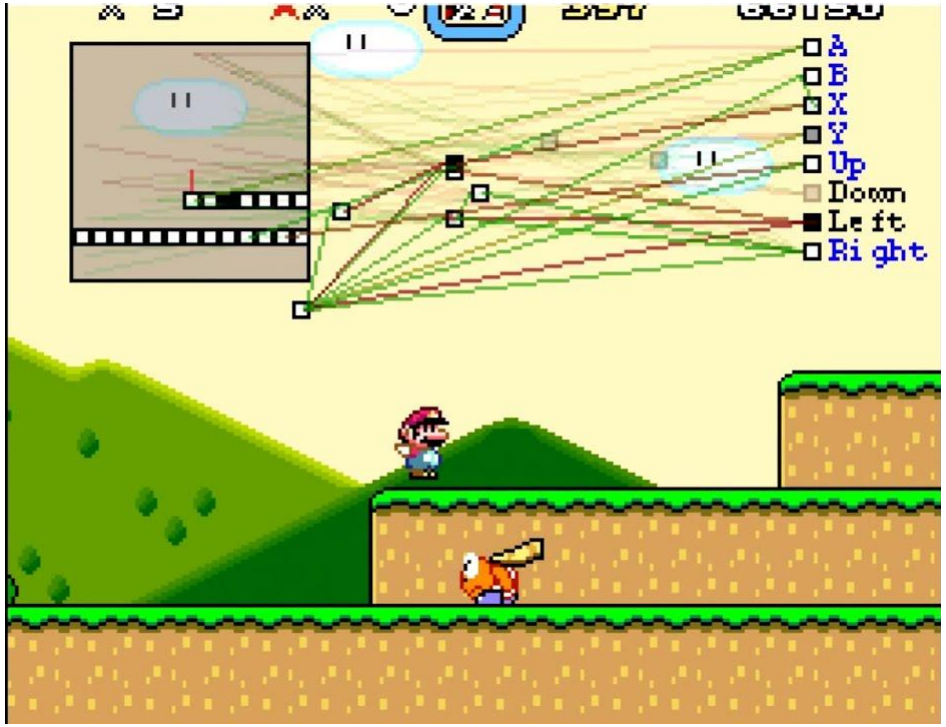
# Poten ample Data (Generalization Issues)


**Traffic lights,
Or truck carrying
traffic lights?**

**ML/AI has issues with generalization beyond the dataset.**

**Anything beyond the training dataset needs some form of human intervention.**

# Reinforcement Learning For Game AI: Outperforming All Humans?




**Game AI**
**(OpenAI, DeepMind, …)**

**If data is the issue,**
**Why not make the data infinite?**

**Won’t machine learning be perfect there?**

**=**

**Reinforcement Learning**

# Reinforcement Learning Demonstrates a Overfitting Issue

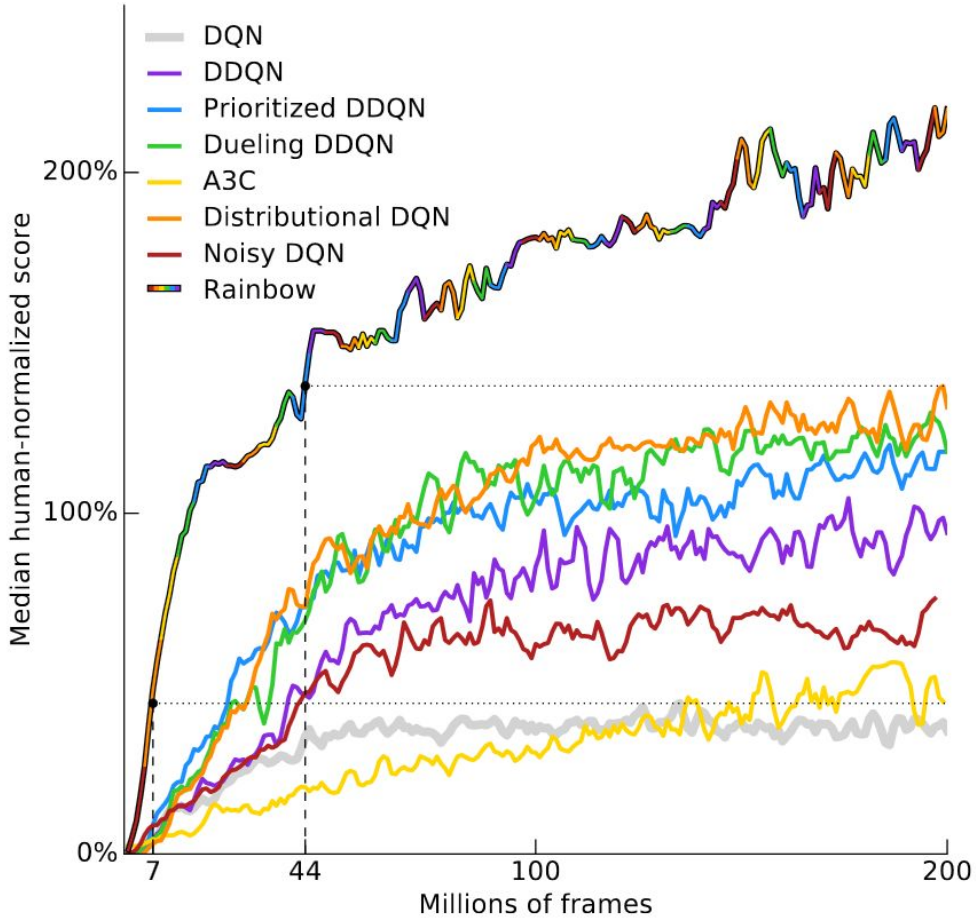
- A coworker is teaching an agent to navigate a room. The episode terminates if the agent walks out of bounds. He didn't add any penalty if the episode terminates this way. The final policy learned to be suicidal, because negative reward was plentiful, positive reward was too hard to achieve, and a quick death ending in 0 reward was preferable to a long life that risked negative reward.
- A friend is training a simulated robot arm to reach towards a point above a table. It turns out the point was defined *with respect to the table*, and the table wasn't anchored to anything. The policy learned to slam the table really hard, making the table fall over, which moved the target point too. The target point *just so happened* to fall next to the end of the arm.
- A researcher gives a talk about using RL to train a simulated robot hand to pick up a hammer and hammer in a nail. Initially, the reward was defined by how far the nail was pushed into the hole. Instead of picking up the hammer, the robot used its own limbs to punch the nail in. So, they added a reward term to encourage picking up the hammer, and retrained the policy. They got the policy to pick up the hammer…but then it threw the hammer at the nail instead of actually using it.

**ML results are good for humor.**

**Deep Reinforcement Learning Doesn't Work Yet**

https://www.alexirpan.com/2018/02/14/rl-hard.html

# Reinforcement Learning Demonstrates a Major Training Data Issue




## Other con of reinforcement learning:

## Requires an accurate simulator so it can generate hundreds of millions of training points!

The y-axis is “median human-normalized score”. This is computed by training 57 DQNs, one for each Atari game, normalizing the score of each agent such that human performance is 100%, then plotting the median performance across the 57 games. RainbowDQN passes the 100% threshold at about *18 million* frames. This corresponds to about 83 hours of play experience, plus however long it takes to train the model. A lot of time, for an Atari game that most humans pick up within a few minutes.

**Deep Reinforcement Learning Doesn't Work Yet**
https://www.alexirpan.com/2018/02/14/rl-hard.html

# Reinforcement Learning Demonstrates a Major Cost Issue


**AlphaStar: Closed world reinforcement learning agent takes $1 billion of research and training! (Still needed human intervention, beat at later AI events by a university team)**

**Cost:**

**AlphaGo – learns to play Go**
**$35 million**

**GPT-3: Natural Language Processing**
**$10 million**

**AlphaStar: Starcraft Bot**
**$12 million for the final**
**>$1billion in research tuning**

**GPT-5: $500 million to train, total human cost in tens of billions**

**Claude Opus: “Billions of dollars range”, total cost high tens of billions**

http://ceur-ws.org/Vol-2903/IUI21WS-TExSS-5.pdf

# THE COST OF AI

*Billions spent chasing artificial intelligence*

Pennsylvania's dormant Three Mile Island nuclear plant would be brought back to life to feed the voracious energy needs of Microsoft under an unprecedented deal in which the tech giant would buy 100 percent of its power for 20 years.

***— Washington Post, Sept 2024***

Training a single AI model can emit as much carbon as five cars in their lifetimes. The training of GPT-3 consumed millions of dollars worth of energy.

***— Team-GPT Analysis***

OpenAI CEO Sam Altman vaguely pegged the training cost at 'more than $100 million' for GPT-4. By 2025 we may have a $10 billion model.

***— Anthropic CEO Dario Amodei, 2024***

Microsoft spent $34.9 billion in just one quarter while Meta plans to spend up to $72 billion this year on AI infrastructure.

***— CNBC, Oct 2025***

The Stargate Project intends to invest $500 billion over the next four years building new AI infrastructure for OpenAI in the United States.

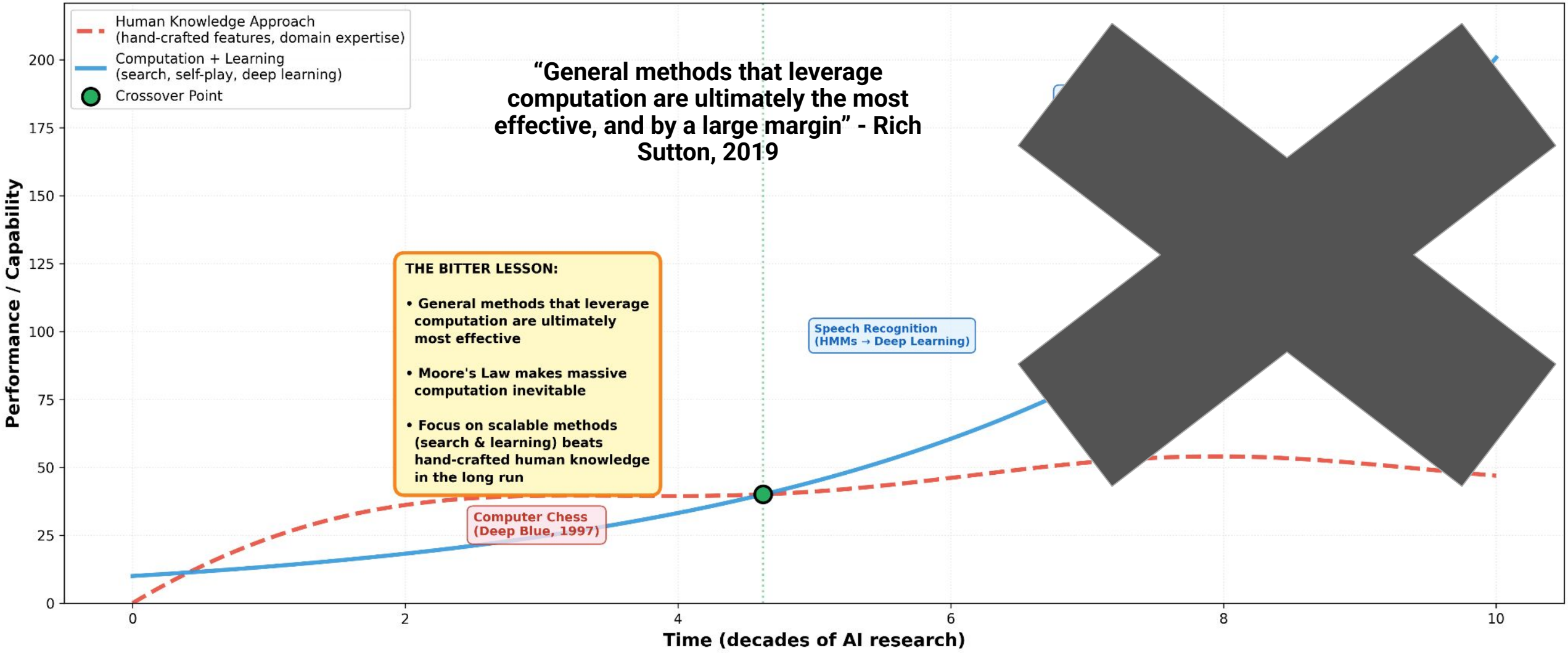
***— OpenAI, Jan 2025***

A single ChatGPT query consumes almost ten times the electricity of a standard Google search. Data centers could consume 10% of all US electricity by 2030.

***— Goldman Sachs & IEA***

*Tech giants are reshaping global power infrastructure to fuel the AI race*

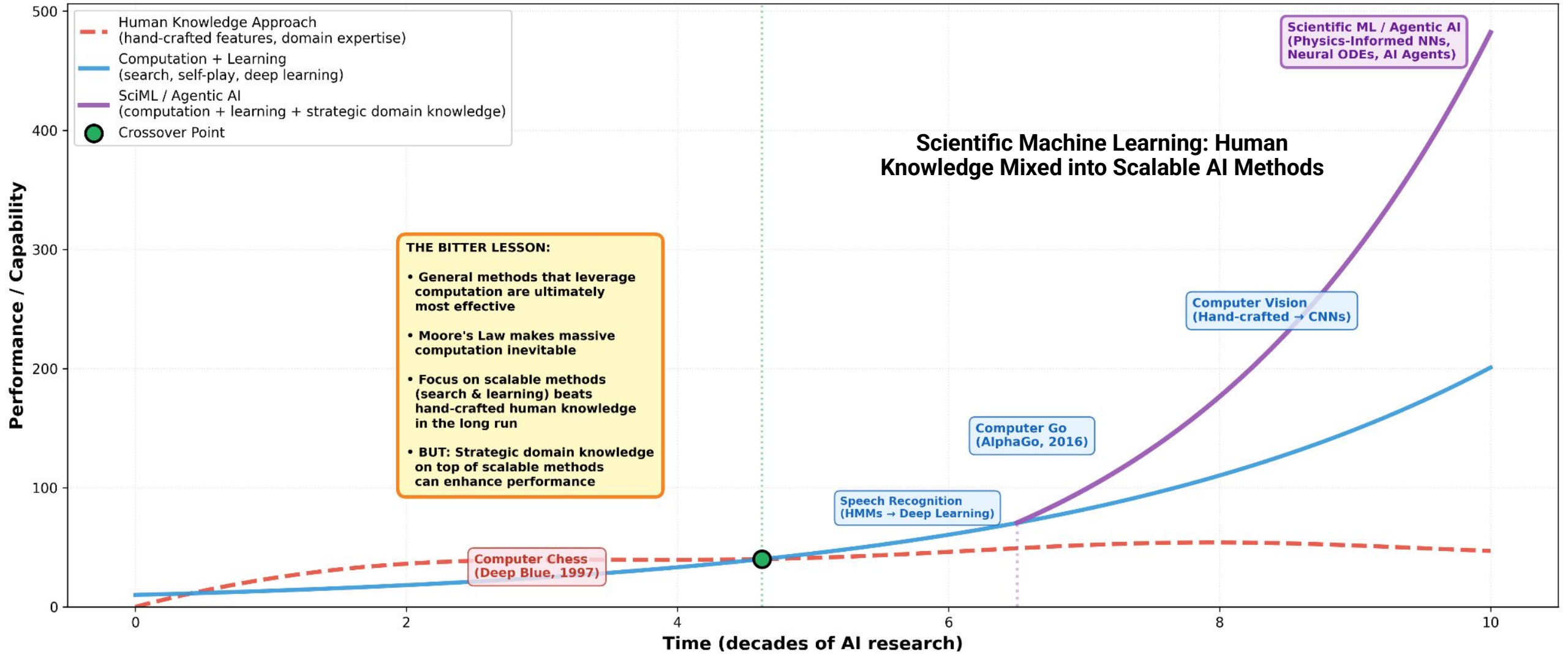
# The Bitter Lesson of AI
# (Rich Sutton, 2019)



**ML/AI scaling costs are growing exponentially**

**Is there a way around these issues?**

# The Bitter Lesson of AI + The SciML/Agentic AI Synthesis
# (Rich Sutton, 2019 + Modern Extensions)



# Two Solutions to The Bitter Lesson

Complementary Approaches to Overcoming AI's Limitations

## Scientific Machine Learning

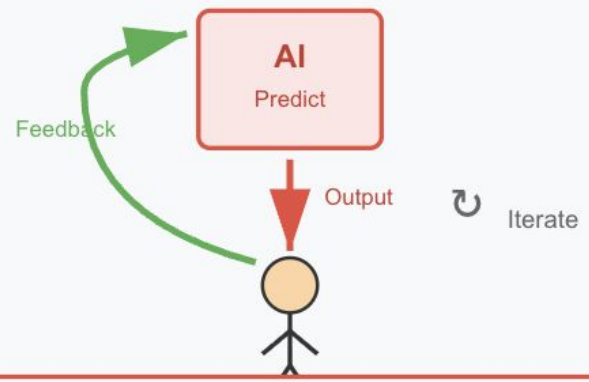
**Adding Scientific Knowledge to Scalable AI Methods**

$$\partial u/\partial t = f(u, t)$$
$$F = ma$$
$$\nabla^2\phi = \varrho$$
$$E = mc^2$$

**Key Concept:**
Embed physical laws, conservation principles, and domain knowledge directly into neural network architectures and training processes

## Agentic AI

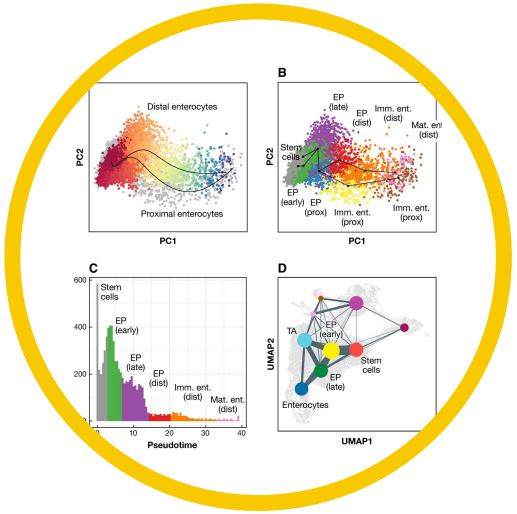
**Agentic Loops with Humans in the Loop**

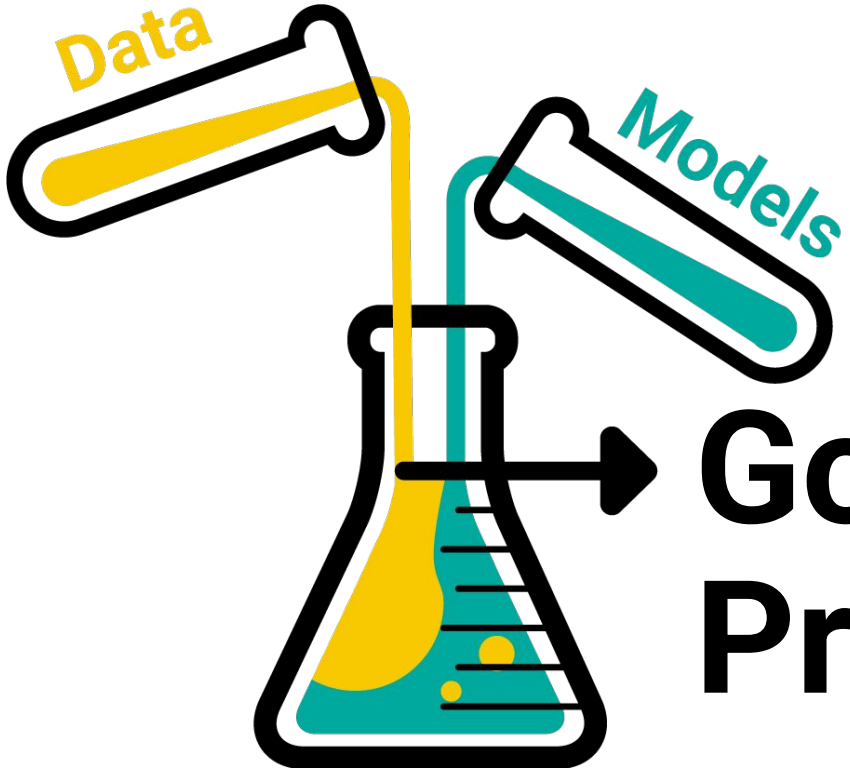



**Key Concept:**
Use AI as a tool in an iterative workflow where human expertise validates, corrects, and guides predictions in a continuous improvement loop

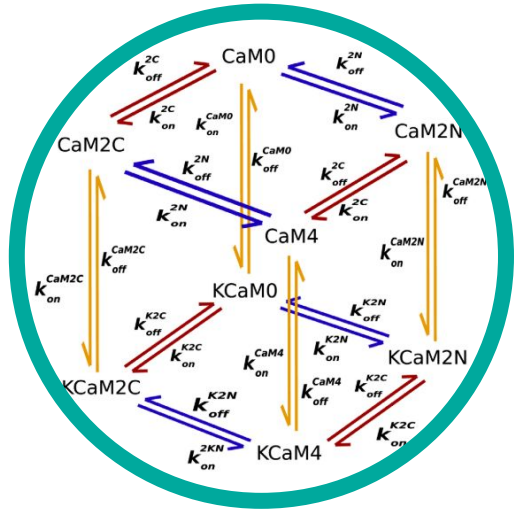
# Scientific Machine Learning is model-based data-efficient machine learning

How do we simultaneously use both sources of knowledge?



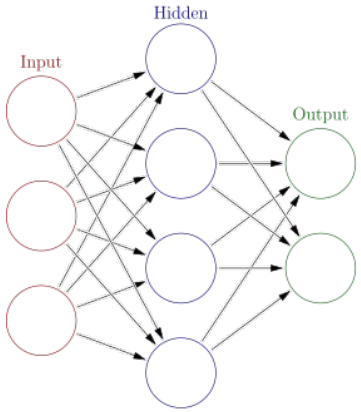

Good Predictions



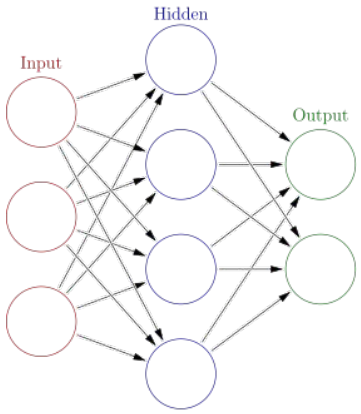
**Core Point: the more structure that is encoded into the model, the less there is to learn and the more data efficient the algorithm is!**

# Universal (Approximator) Differential Equations

$$u' = f(u, \quad , t)$$

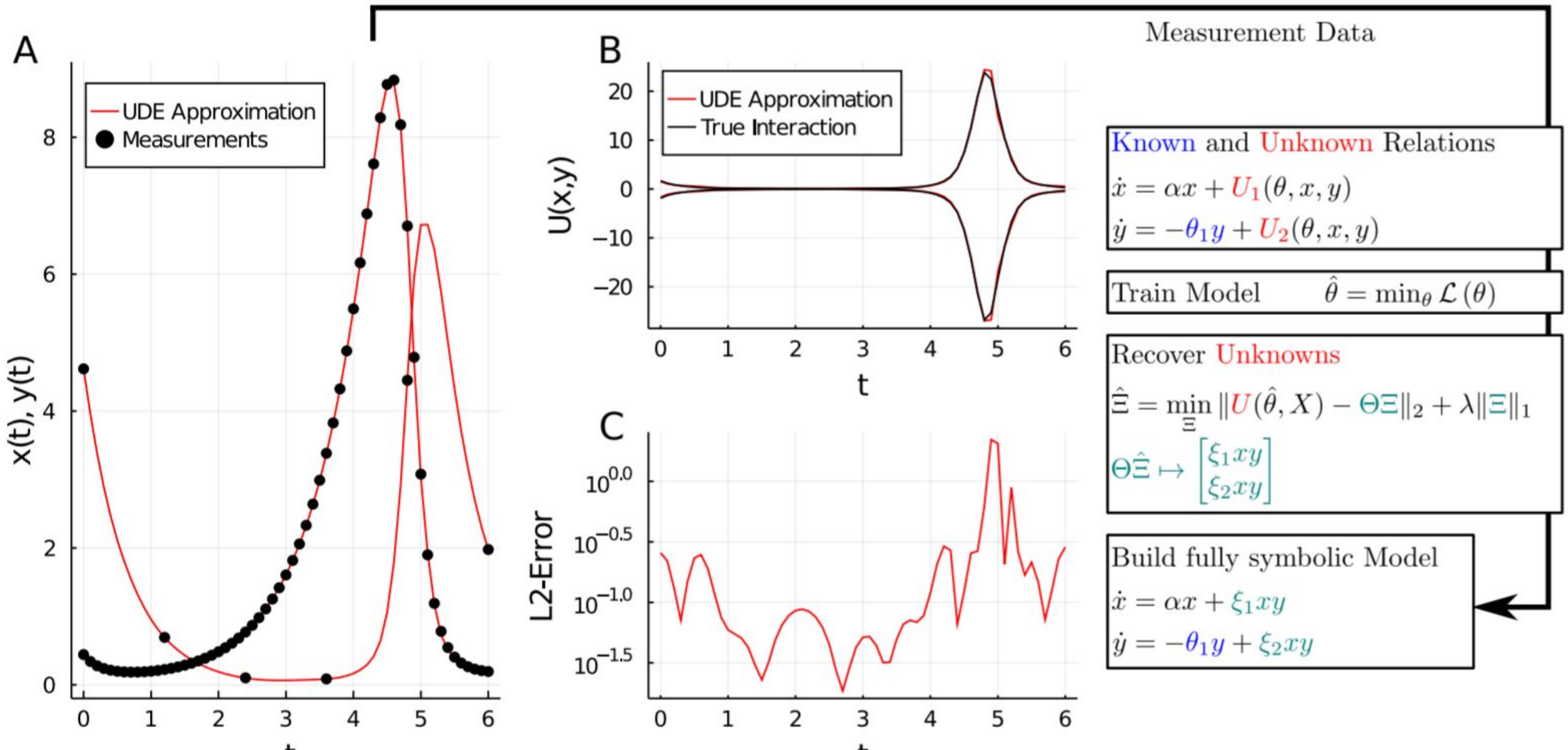



$$x' = \alpha x \quad +$$
$$y' = -\beta y +$$




Rackauckas, Christopher, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. "Universal differential equations for scientific machine learning." *arXiv preprint arXiv:2001.04385* (2020).

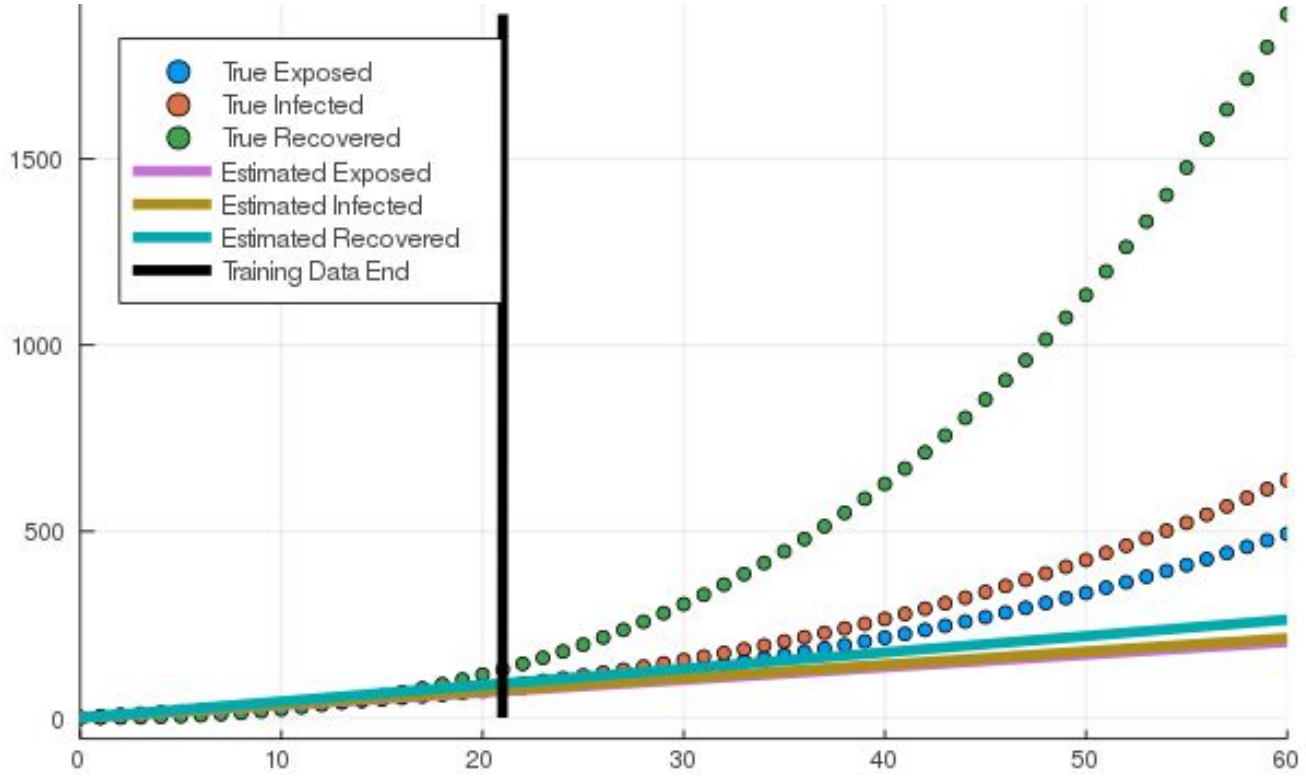
# Universal (Approximator) Differential Equations



Julia SIM

# Let's dive in a bit!

## Standard Machine Learning: Learn the whole model




u'=NN(u) trained on 21 days of data

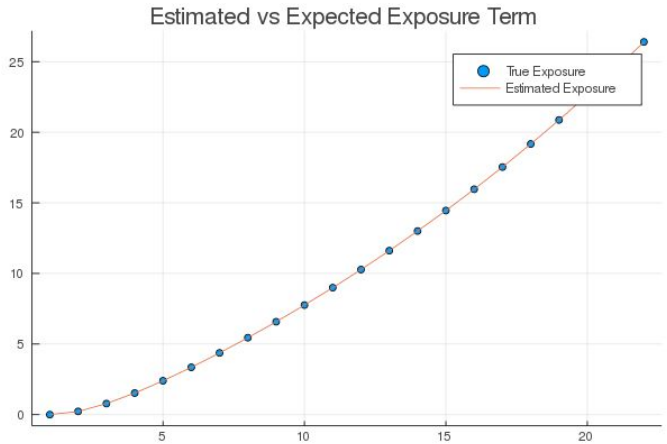
Can fit, but not enough information to accurately extrapolate

**Does not have the correct asymptotic behavior**

More examples of this issue:

Ridderbusch et al. "Learning ODE Models with Qualitative Structure Using Gaussian Processes."

# Universal ODE




$$
\begin{aligned}
S' &= -\frac{\beta_0 S F}{N} - \mu S, \\
E' &= \frac{\beta_0 S F}{N} - \left(\sigma + \mu\right) E, \\
I' &= \sigma E - (\gamma + \mu) I, \\
R' &= \gamma I - \mu R, \\
N' &= -\mu N, \\
D' &= d\,\gamma I - \lambda D, \quad \text{and} \\
C' &= \sigma E,
\end{aligned}
$$

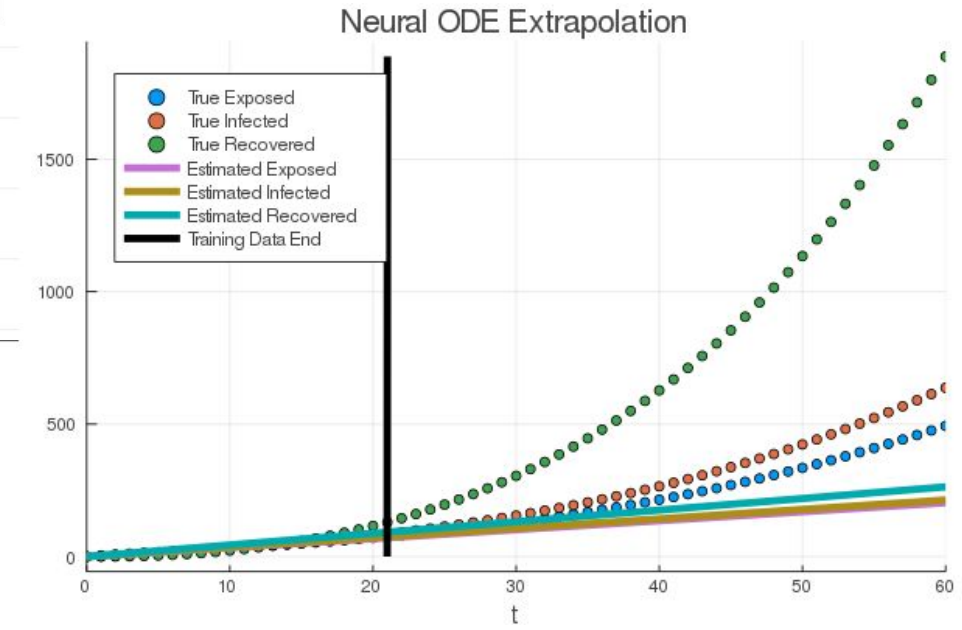
Replace Unknown Portion

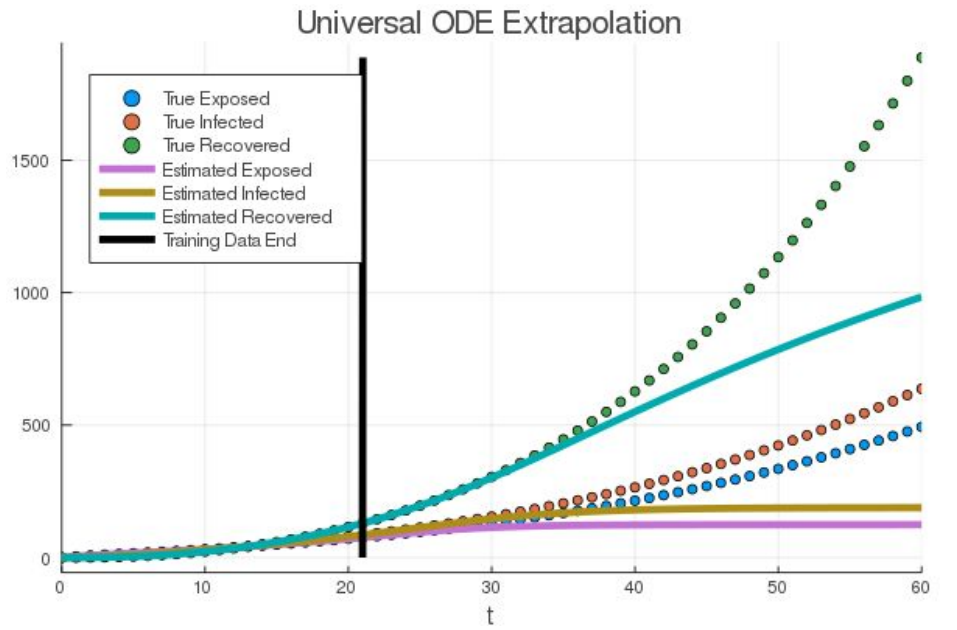
Replace Unknown Portion

Exposure: **Unknown**

Infection rates: known From disease quantities

Percentage of cases known to be severe, can be estimated



# Universal ODE -> Internal Sparse Regression

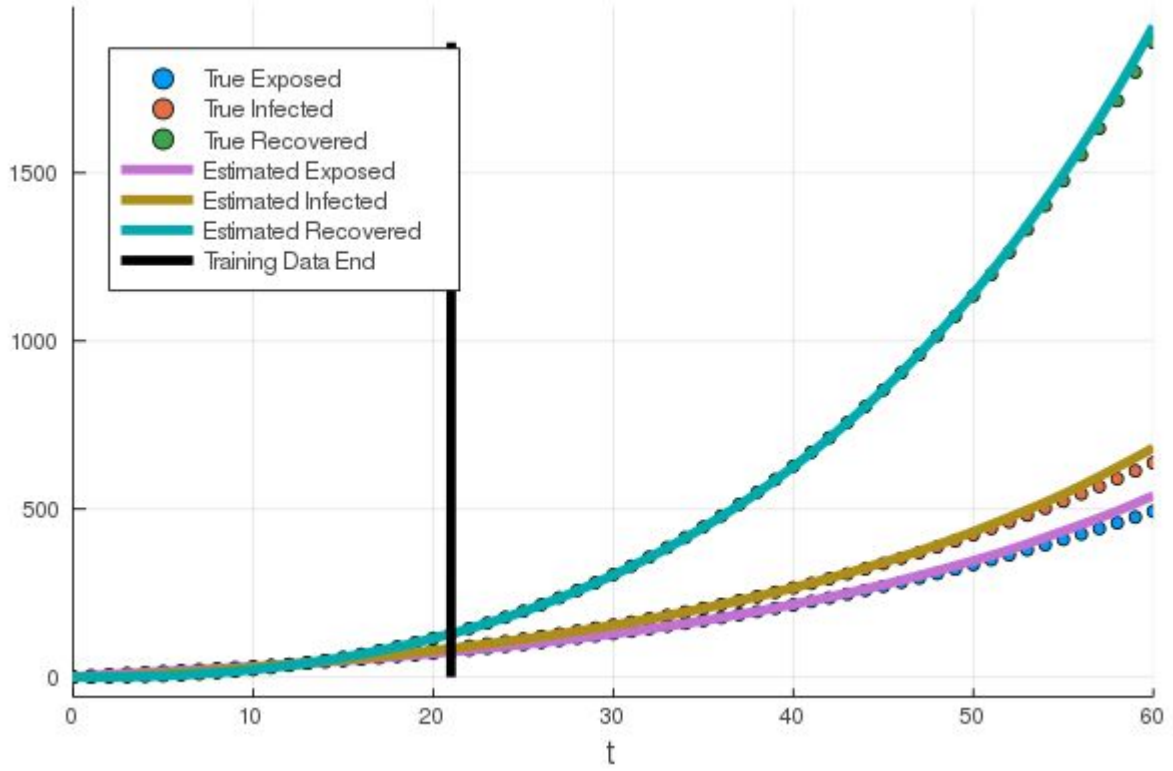
Sparse Identification on only the missing term:
I * 0.10234428543435758 + S/N * I * 0.11371750552005416 + (S/N) ^ 2 * I * 0.1263545979855597

$$
\begin{aligned}
S' &= -\frac{\beta_0 SF}{N} - \mu S, \\
E' &= \frac{\beta_0 SF}{N} + \left(\sigma + \mu\right) E, \\
I' &= \sigma E - (\gamma + \mu) I, \\
R' &= \gamma I - \mu R, \\
N' &= -\mu N, \\
D' &= d\,\gamma I - \lambda D, \quad \text{and} \\
C' &= \sigma E,
\end{aligned}
$$

Replace Unknown Portion

Replace Unknown Portion

Sparsity improves generalizability!



# Accurate Model Extrapolation Mixing in Physical Knowledge

Upon denoting $\mathbf{x} = (\phi, \chi, p, e)$, we propose the following family of UDEs to describe the two-body relativistic dynamics:

$$\dot{\phi} = \frac{(1 + e\cos(\chi))^2}{Mp^{3/2}}\big(1 + \mathcal{F}_1(\cos(\chi), p, e)\big), \tag{5a}$$

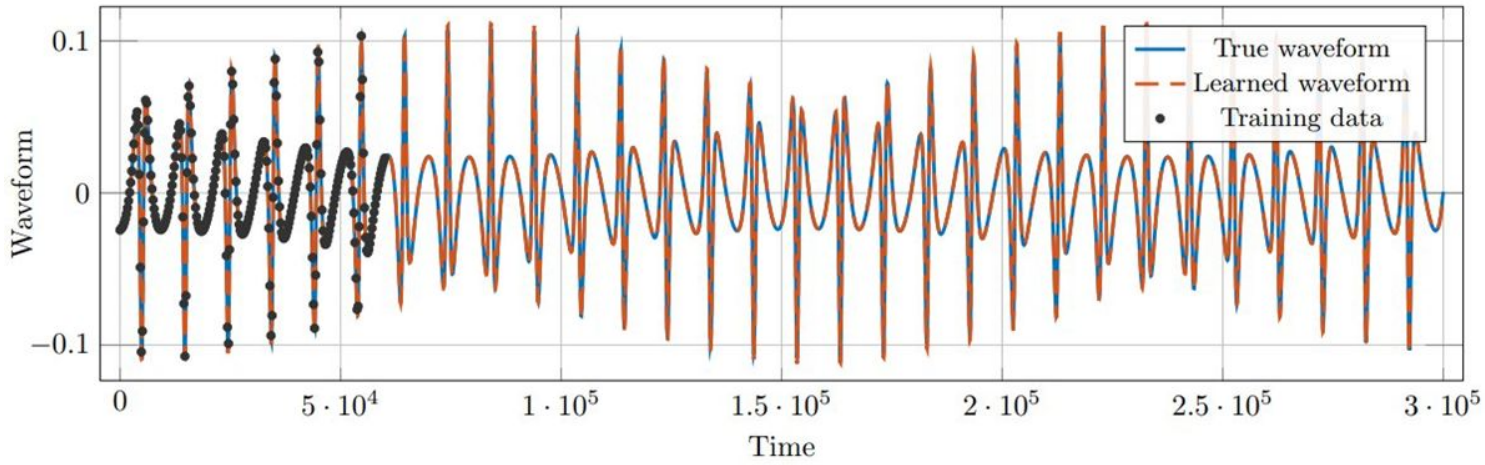
$$\dot{\chi} = \frac{(1 + e\cos(\chi))^2}{Mp^{3/2}}\big(1 + \mathcal{F}_2(\cos(\chi), p, e)\big), \tag{5b}$$

$$\dot{p} = \mathcal{F}_3(p, e), \tag{5c}$$

$$\dot{e} = \mathcal{F}_4(p, e), \tag{5d}$$

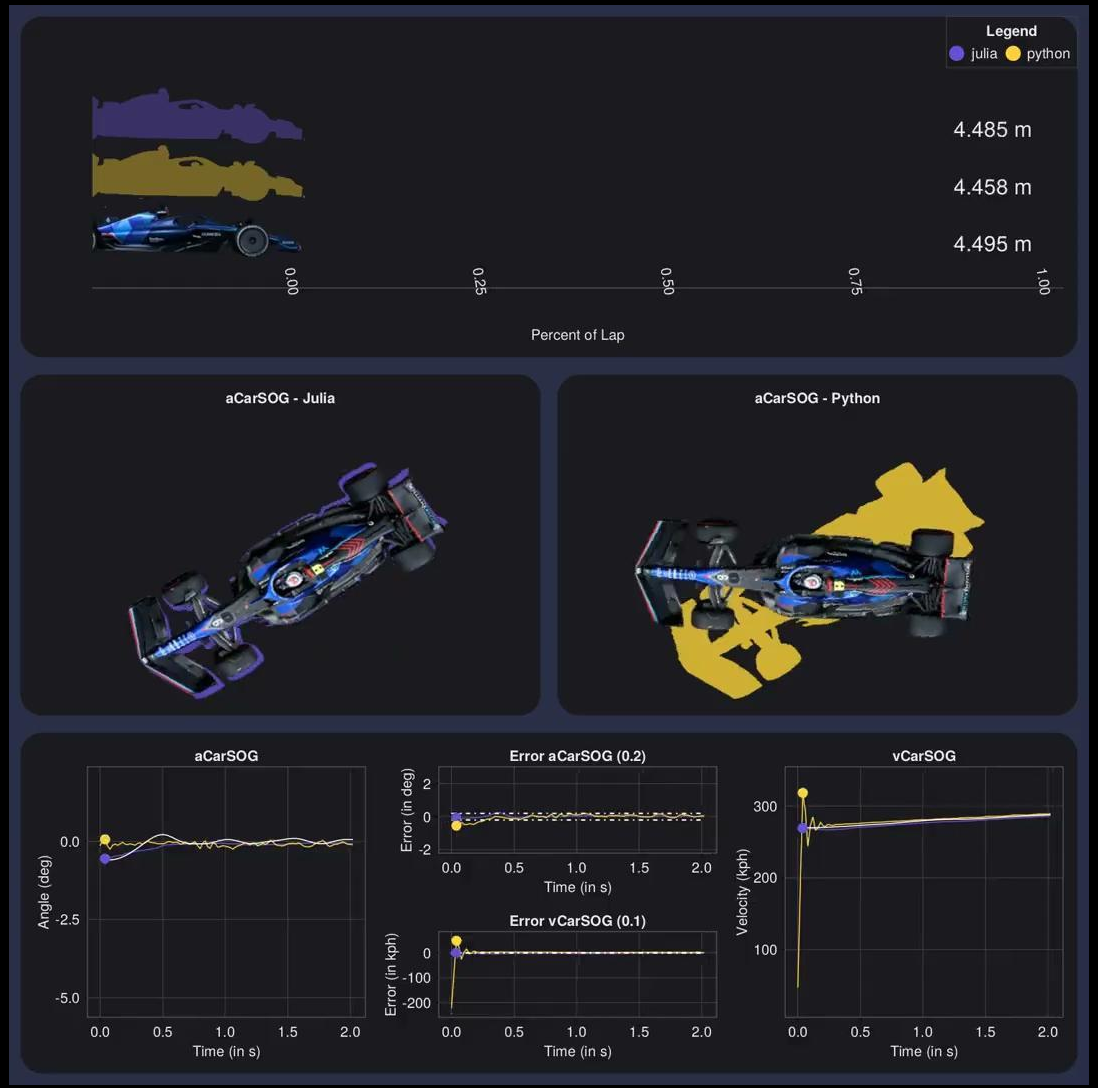
Automated discovery of geodesic equations from LIGO black hole data: run the code yourself!

https://github.com/Astroinformatics/ScientificMachineLearning/blob/main/neuralode_gw.ipynb



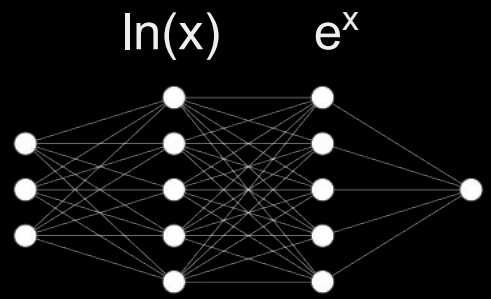

Keith, B., Khadse, A., & Field, S. E. (2021). Learning orbital dynamics of binary black hole systems from gravitational wave measurements. Physical Review Research, 3(4), 043101.

# Scientific Machine Learning Digital Twins: More Realistic Results than Pure ML




## Physically-Informed Machine Learning

$$\dot{\beta} \approx \frac{a_y}{u} - \frac{a_x}{u} - r$$




Using knowledge of the physical forms as part of the design of the neural networks.

Formulation: Koopman + SIREN + Physics

**Smoother, more accurate results**

# Scientific Machine Learning Digital Twins: More Realistic Results than Pure ML

WILLIAMS RACING


**Result:**

**Math is cool!**

# SciML Shows how to build Earthquake-Safe Buildings

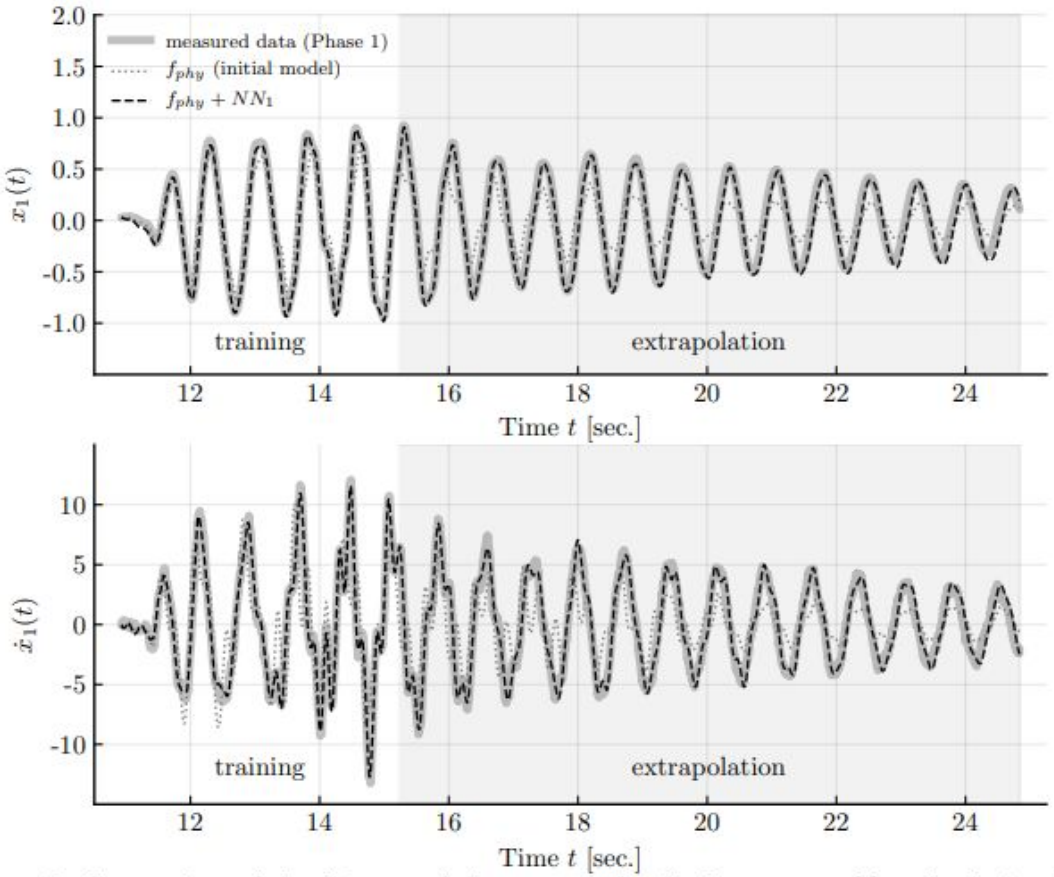



Figure 12: Comparison of time history of the response for displacement $x_1(t)$ and velocity $\dot{x}_1(t)$ for the NSD experiment (Phase 1).

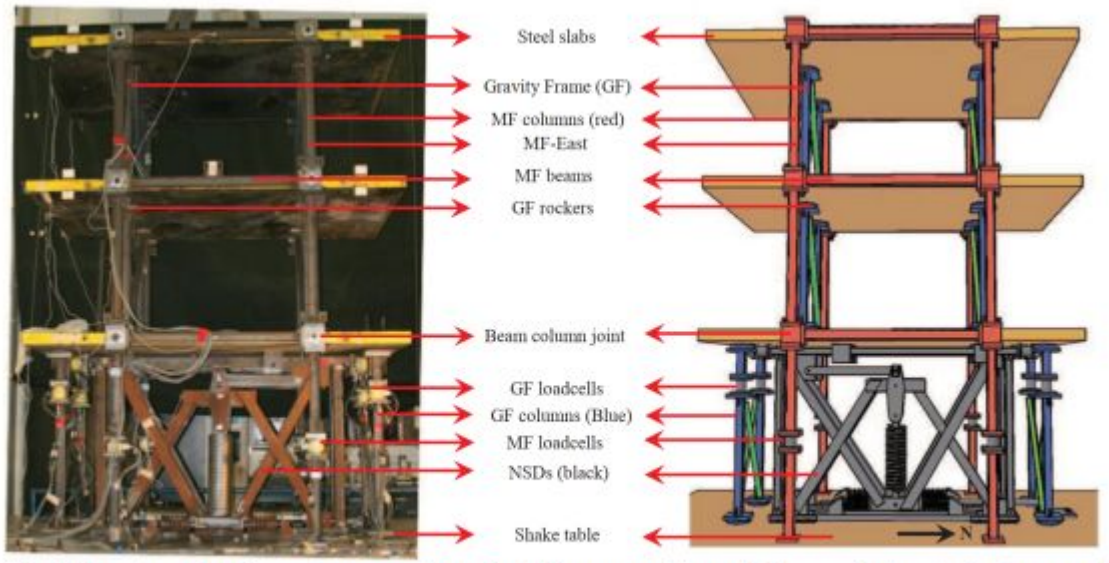



Figure 10: The structural system equipped with a negative stiffness device in between the first floor and the shake table.

Structural identification with physics-informed neural ordinary differential equations
Lai, Zhilu, Mylonas, Charilaos, Nagarajaiah, Satish, Chatzi, Eleni

**For a detailed walkthrough of UDEs and applications watch on Youtube:**

**Chris Rackauckas: Accurate and Efficient Physics-Informed Learning Through Differentiable Simulation**

# Fundamental of Clinical Pharmacology: Nonlinear Mixed Effects (NLME) Modeling

***Goal*: Learn to predict patient behavior (dynamics) from simple data (covariates)**

$$Z_i = \begin{bmatrix} wt_i, \\ sex_i, \end{bmatrix}$$

**Covariates**

Math: Find $(\theta, \eta)$ such that $\mathrm{E}[\eta] = 0$

Requires special fitting procedures (Pumas)

$$g_i = \begin{bmatrix} Ka \\ CL \\ V \end{bmatrix} = \begin{bmatrix} \theta_1 e^{\eta_{i,1} \kappa_{i,k,1}}, \\ \theta_2 (\frac{wt_i}{70})^{0.75} \theta_4^{sex_i} e^{\eta_{i,2}}, \\ \theta_3 e^{\eta_{i,3}}, \end{bmatrix}$$

**Structural Model (pre)**

$$\frac{d[\text{Depot}]}{dt} = -Ka[\text{Depot}],$$
$$\frac{d[\text{Central}]}{dt} = Ka[\text{Depot}] - \frac{CL}{V}[\text{Central}].$$

**Dynamics**

Intution: $\eta$ (the random effects) are a fudge factor

Find $\theta$ (the fixed effect, or average effect) such that you can predict new patient dynamics as good as possible

# The Impact of Pumas (PharmacUtical Modeling And Simulation)

## We have been using Pumas software for our pharmacometric needs to support our development decisions and regulatory submissions.

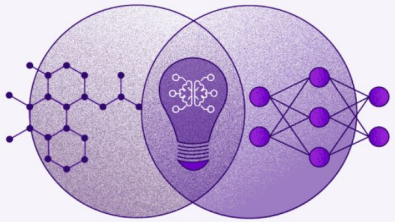
Pumas software has surpassed our expectations on its accuracy and ease of use. We are encouraged by its capability of supporting different types of pharmacometric analyses within one software. **Pumas has emerged as our "go-to" tool for most of our analyses in recent months.** We also work with Pumas-AI on drug development consulting. We are impressed by the quality and breadth of the experience of Pumas-AI scientists in collaborating with us on modeling and simulation projects across our pipeline spanning investigational therapeutics and vaccines at various stages of clinical development

**Husain A. PhD (2020)**
Director, Head of Clinical Pharmacology and Pharmacometrics,
*Moderna Therapeutics, Inc*

moderna™
messenger therapeutics

Built on SciML

DeepPumas



# DeepPumas: Deep Learning Nonlinear Mixed Effects Software

Method and apparatus for automating models for individualized administration of medicaments, Christopher Vincent Rackauckas, Vijay Ivaturi, US63/136,719

**Typical values**

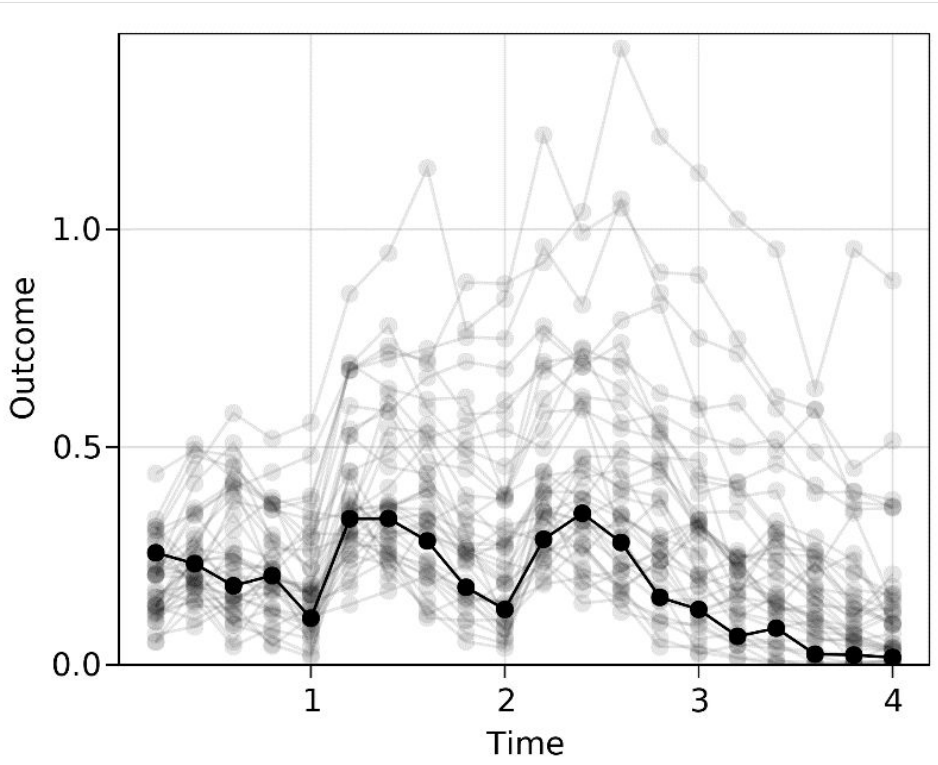
$$\theta \in \mathbb{R}^3_+$$
$$\Omega \in \mathbb{R}^3_+$$

**Patient data**

Age
Weight

**Random effects**

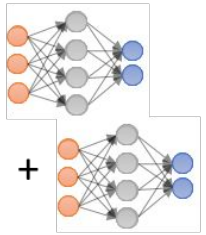
$$\eta \sim \text{MvNormal}(\Omega)$$




**Individual parameters**

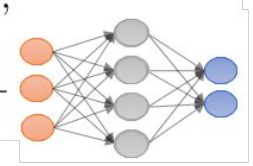
$$\begin{aligned} Ka_i &= \theta_1 \cdot e^{\eta_{i,1}} + c_1 \cdot Age_i \\ CL_i &= \theta_2 \cdot e^{\eta_{i,2}} \\ V_i &= \theta_3 \cdot e^{\eta_{i,3}} + c_2 \cdot Weight_1^{c_3} \end{aligned}$$

+



**Dynamics**

$$\begin{aligned} \frac{d[\text{Depot}]}{dt} &= -Ka[\text{Depot}], \\ \frac{d[\text{Central}]}{dt} &= Ka[\text{Depot}] - \end{aligned}$$



**Error model**

$$Outcome \sim \text{Normal}\left(Central, \sqrt{Central} \cdot \sigma\right)$$

# Two Solutions to The Bitter Lesson

Complementary Approaches to Overcoming AI's Limitations

## Scientific Machine Learning

**Adding Scientific Knowledge to Scalable AI Methods**

$$\partial u/\partial t = f(u, t)$$
$$F = ma$$
$$\nabla^2\phi = \varrho$$
$$E = mc^2$$

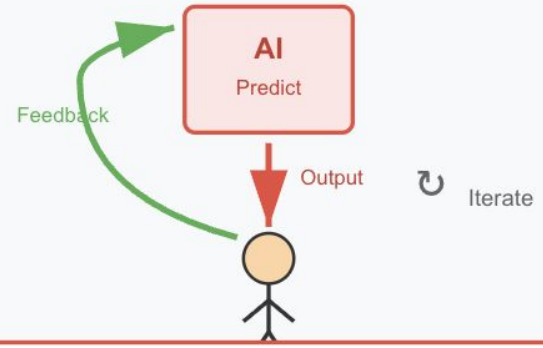
**Key Concept:**
Embed physical laws, conservation principles, and domain knowledge directly into neural network architectures and training processes

**Benefits:**
- ✓ Better generalization with less data
- ✓ Physical consistency guaranteed
- ✓ Interpretable predictions
- ✓ Efficient for scientific applications

*Example: Universal Differential Equations*

## Agentic AI

**Agentic Loops with Humans in the Loop**




**Key Concept:**
Use AI as a tool in an iterative workflow where human expertise validates, corrects, and guides predictions in a continuous improvement loop

**Benefits:**
- ✓ Catches and corrects AI errors quickly
- ✓ Leverages human domain expertise
- ✓ Improves with each iteration
- ✓ Builds trust through transparency

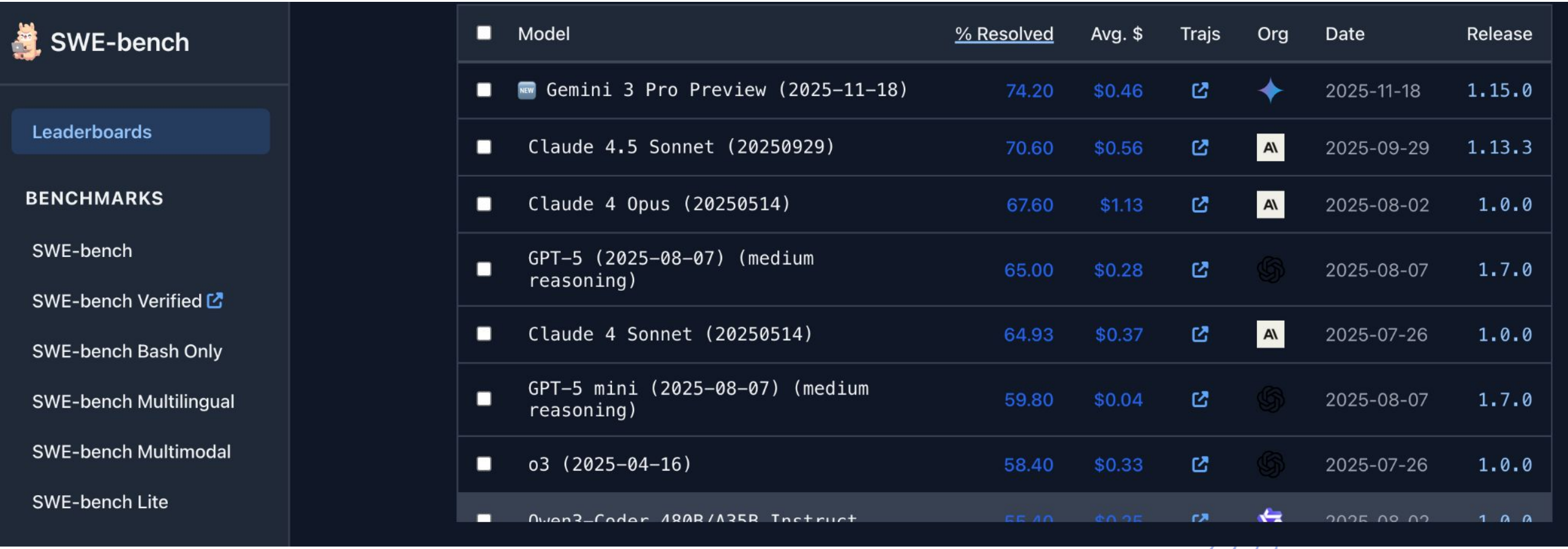
*Example: AI-Assisted Engineering Design*

**Agentic AI in science isn’t tomorrow.**

**It’s today.**

**How do you get started today?**

# AI for Software Engineering is Progressing



# WHY AGENTIC AI ACTUALLY WORKS

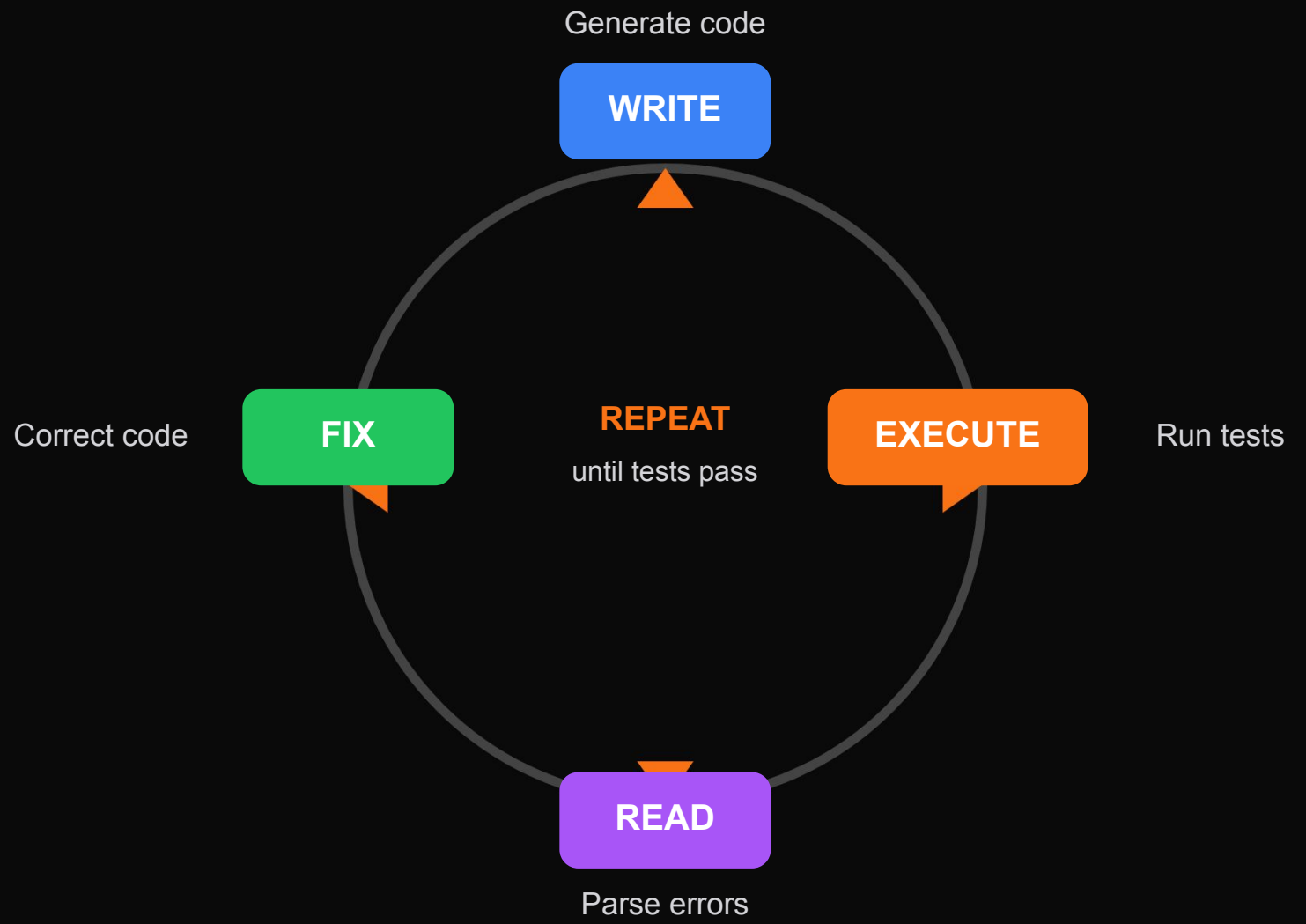
**It's not about LLM accuracy. It's about the feedback loop.**

**THE MYTH**

"LLMs are getting accurate enough to generate correct code on the first try."

**THE REALITY**

Agentic AI iterates until correct. The compiler is the oracle.




| ~65% | 95%+ |
|---|---|
| First try | After loop |

The compiler provides ground truth. The loop provides convergence.

# AI CLI CODING AGENTS: HEAD-TO-HEAD

Terminal-based AI assistants for software development (2025)

## Claude Code

ANTHROPIC

✓ PROS

- Best code quality and accuracy
- Superior agentic navigation
- Rich customization (CLAUDE.md)
- Parallel task execution

✗ CONS

- $20-200/month pricing

BEST FOR

**Enterprise & Complex Projects**

## Codex CLI

OPENAI

✓ PROS

- GitHub Actions integration
- ChatGPT subscription included

✗ CONS

- Complex setup process
- Verbose code output
- Less informative CLI UX
- Fewer advanced features

BEST FOR

**CI/CD & Collaborative Teams**

## Gemini CLI

GOOGLE

✓ PROS

- Generous free tier
- 1M token context window
- GCP/Google ecosystem
- Built-in web search grounding

✗ CONS

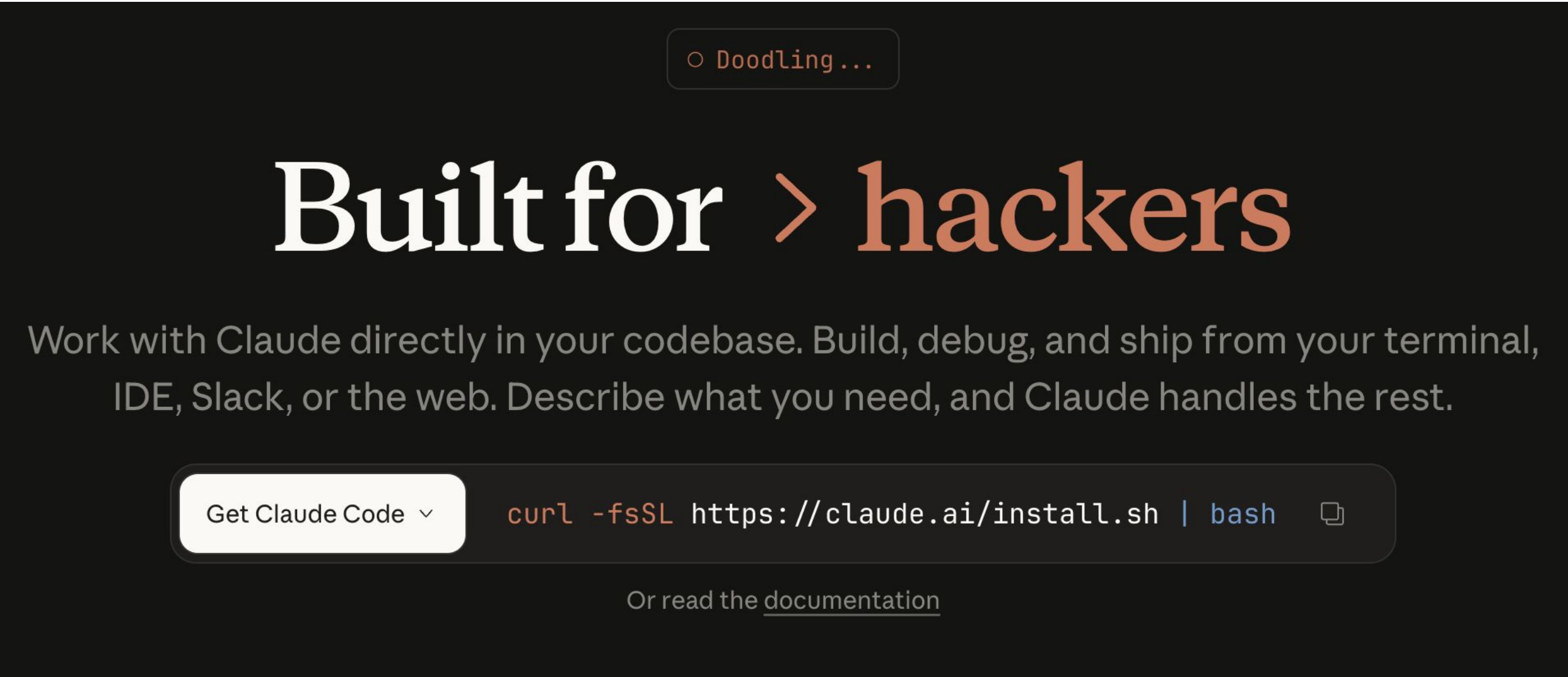
- Requires more guidance
- GCP setup for Workspace
- Less polished output

BEST FOR

**Budget & Large Codebases**

Sources: Terminal-Bench, CodeAnt.AI, Render Blog, EducativeIO (2025)

# Step 1: Get Claude Code Running



## Demonstration:

1. Setting up Claude Code
2. Local orchestration in VS Code
3. Sandboxing and “agentic freedom”
    a. Claude Code CLI
    b. gh bot authentication
4. Multi-agent bot swarm organization

JuliaHub

# With the fundamental accuracy challenges of LLMs, what can be done to improve Agentic AI for engineering?

# WHY AGENTIC AI ACTUALLY WORKS

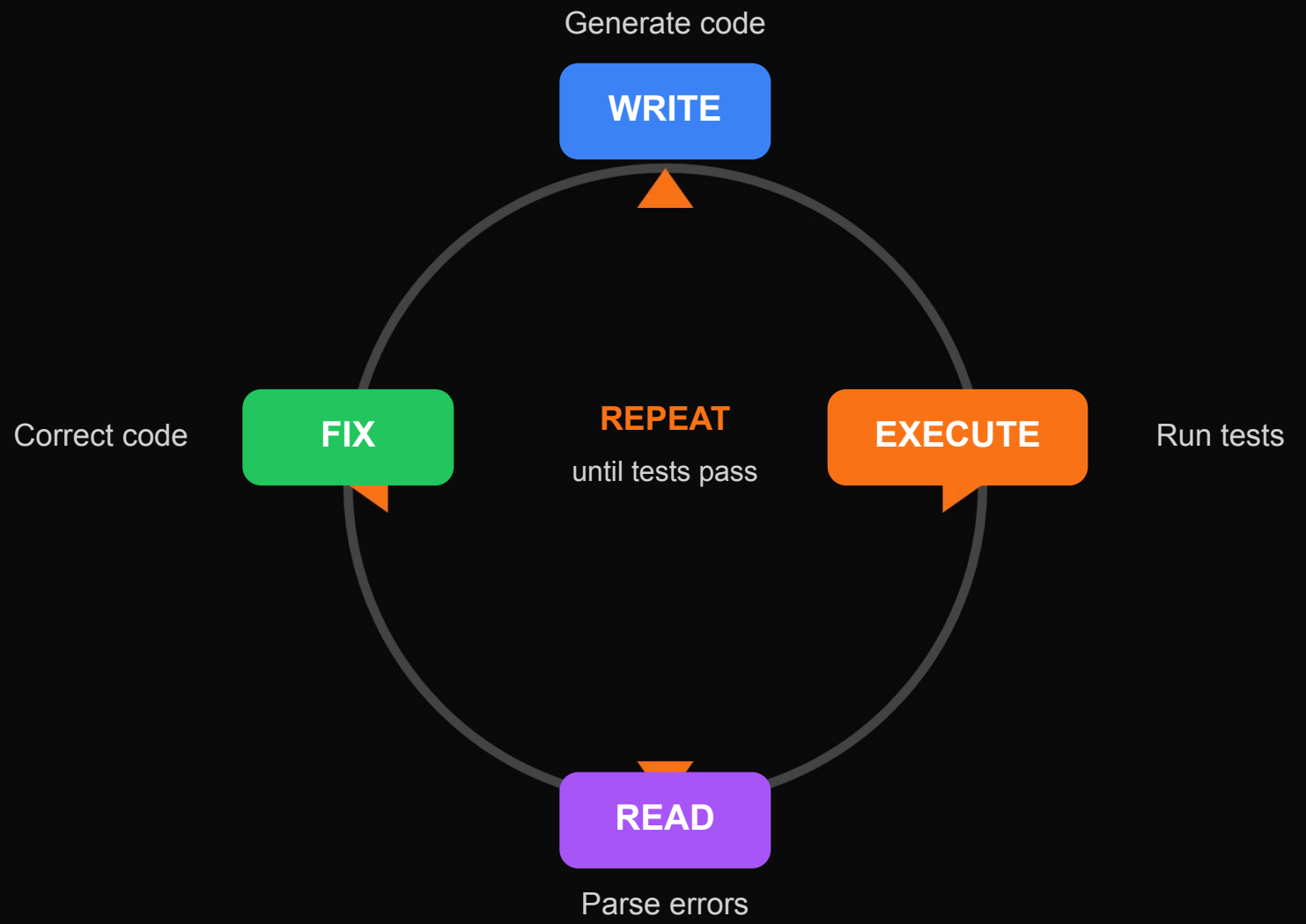
**It's not about LLM accuracy. It's about the feedback loop.**

**THE MYTH**

"LLMs are getting accurate enough to generate correct code on the first try."

**THE REALITY**

Agentic AI iterates until correct. The compiler is the oracle.




| ~65% | 95%+ |
|---|---|
| First try | After loop |

The compiler provides ground truth. The loop provides convergence.

# NVIDIA Research: Small Language Models are Sufficient?

## Small Language Models are the Future of Agentic AI

**Peter Belcak**[1] **Greg Heinrich**[1] **Shizhe Diao**[1] **Yonggan Fu**[1] **Xin Dong**[1]
**Saurav Muralidharan**[1] **Yingyan Celine Lin**[1,2] **Pavlo Molchanov**[1]
[1]NVIDIA Research [2]Georgia Institute of Technology
agents-research@nvidia.com

Here we lay out the position that small language models (SLMs) are *sufficiently powerful, inherently more suitable, and necessarily more economical for many invocations in agentic systems, and are therefore the future of agentic AI*. Our argumentation is grounded in the current level of capabilities exhibited by SLMs, the common architectures of agentic systems, and the economy of LM deployment. We further argue that in situations where general-purpose conversational abilities are essential, heterogeneous agentic systems (i.e., agents invoking multiple different models) are the natural choice. We discuss the potential barriers for the adoption of SLMs in agentic systems and outline a general LLM-to-SLM agent conversion algorithm.

# Can this idea be scaled up to a complete language? A complete modeling system?

# Dyad: Modeling and Simulation Designed for Agentic AI

Let AI write models, but let codegen make it fast and accurate

Simpler code for the AI to write and debug

Static checks for accuracy and agentic debugging

```dyad
# Sample component
component MyComponent
  # A component can have one or more extends clauses, like this one...
  extends BaseComponent
  # A normal parameter only results in a parametric change, not a structural change,
  parameter y::Real
  # A structural parameter is a parameter changes the number of equations or variables.
  structural parameter N::Integer = 10
  # Variables and parameters can have a number of different attributes
  variable x::Real(min = 0, max = 10, guess = 5, units = "m")
  # Subcomponents are components nested inside other components forming hierarchical models
  subcomponent = SomeComponent()
  # Connectors represent points of interaction between the components
  p = Pin()
relations
  # One type of relation is an equation, like this one
  der(x) = y
  # Another type of relation is a connection
  connect(subcomponent.p, p)
end
```

Future: Allow for variables to depend on attributes that specify properties of the variable (e.g. min, max, units)

Supports composition of physical models into components

Acausal modeling framework for capturing algebraic-differential relationships

JuliaHub

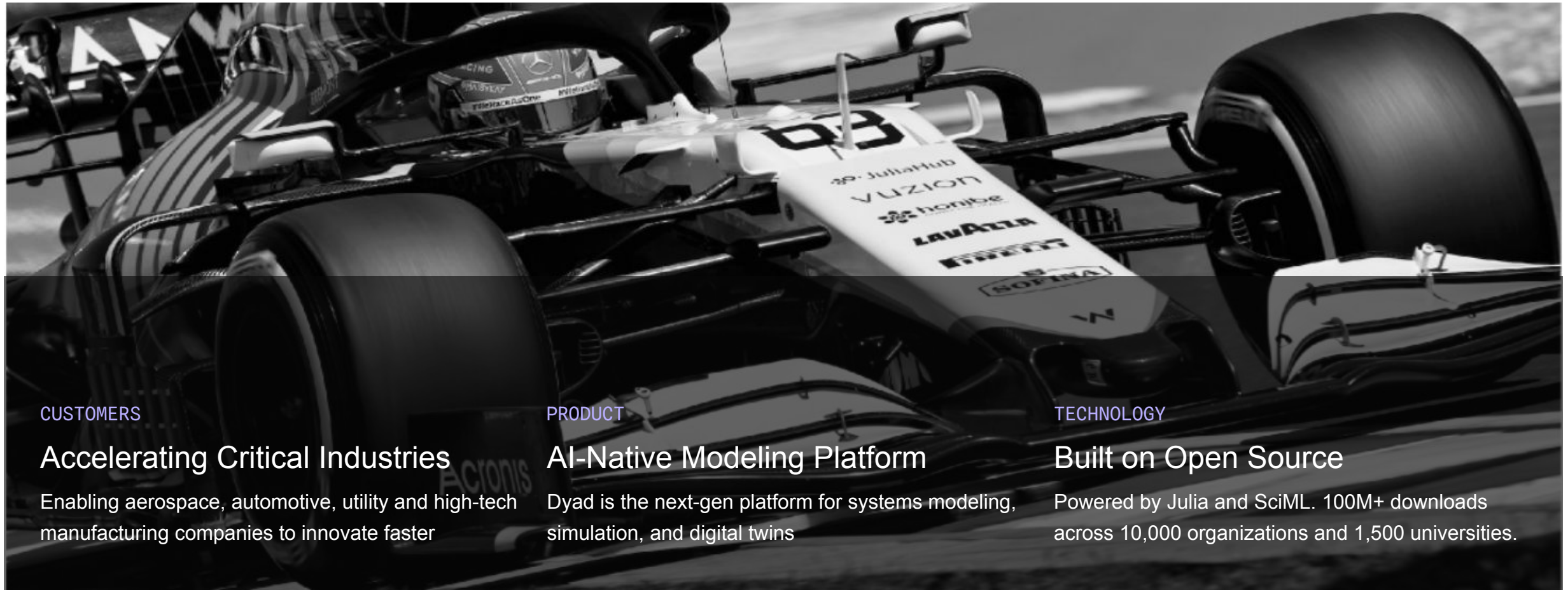
# Dyad

## Advancing Julia into Industrial Modeling and Simulation

1:1 GUI and Code
Physics-based modeling, controls and embedded software in one tool
Generative AI and SciML
Access to the full Julia ecosystem

MISSION

# Dyad accelerates breakthrough engineering with Software-Defined Machines




CUSTOMERS

## Accelerating Critical Industries

Enabling aerospace, automotive, utility and high-tech manufacturing companies to innovate faster

PRODUCT

## AI-Native Modeling Platform

Dyad is the next-gen platform for systems modeling, simulation, and digital twins

TECHNOLOGY

## Built on Open Source

Powered by Julia and SciML. 100M+ downloads across 10,000 organizations and 1,500 universities.

# Physical Modeling with Dyad

**What is Dyad?**

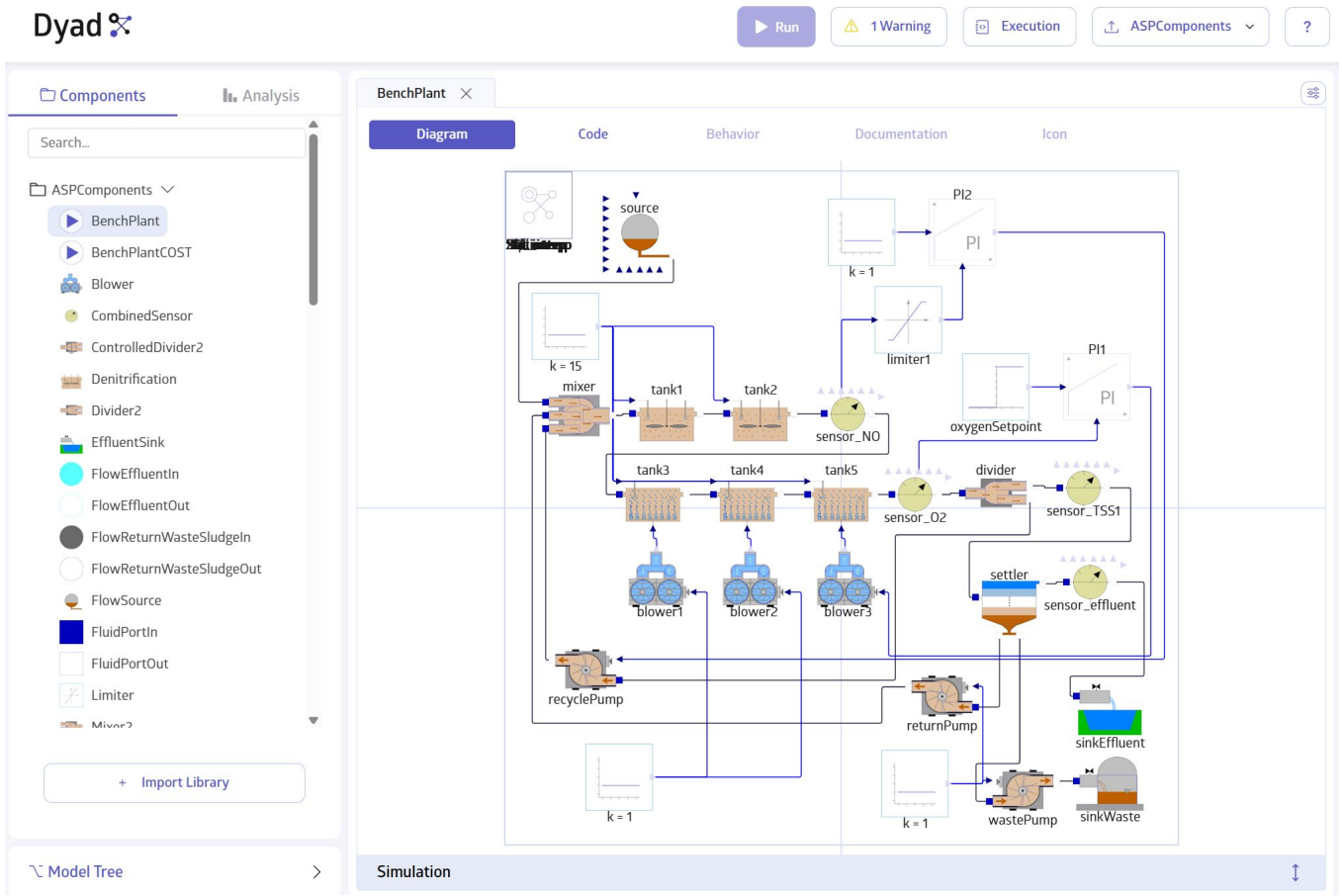
- Acausal plant modeling (0D, 1D, 2D)
- Causal synchronous modeling with state machines
- Agentic AI for engineer productivity
- Scientific AI for analysis models
- Control systems synthesis and code generation
- Modern development workflows: git, CI/CD

**Dyad AI and SciML**

- Surrogate modeling
- Agentic workflows
- Design space optimization through ensembles, UQ, Sensitivity analysis

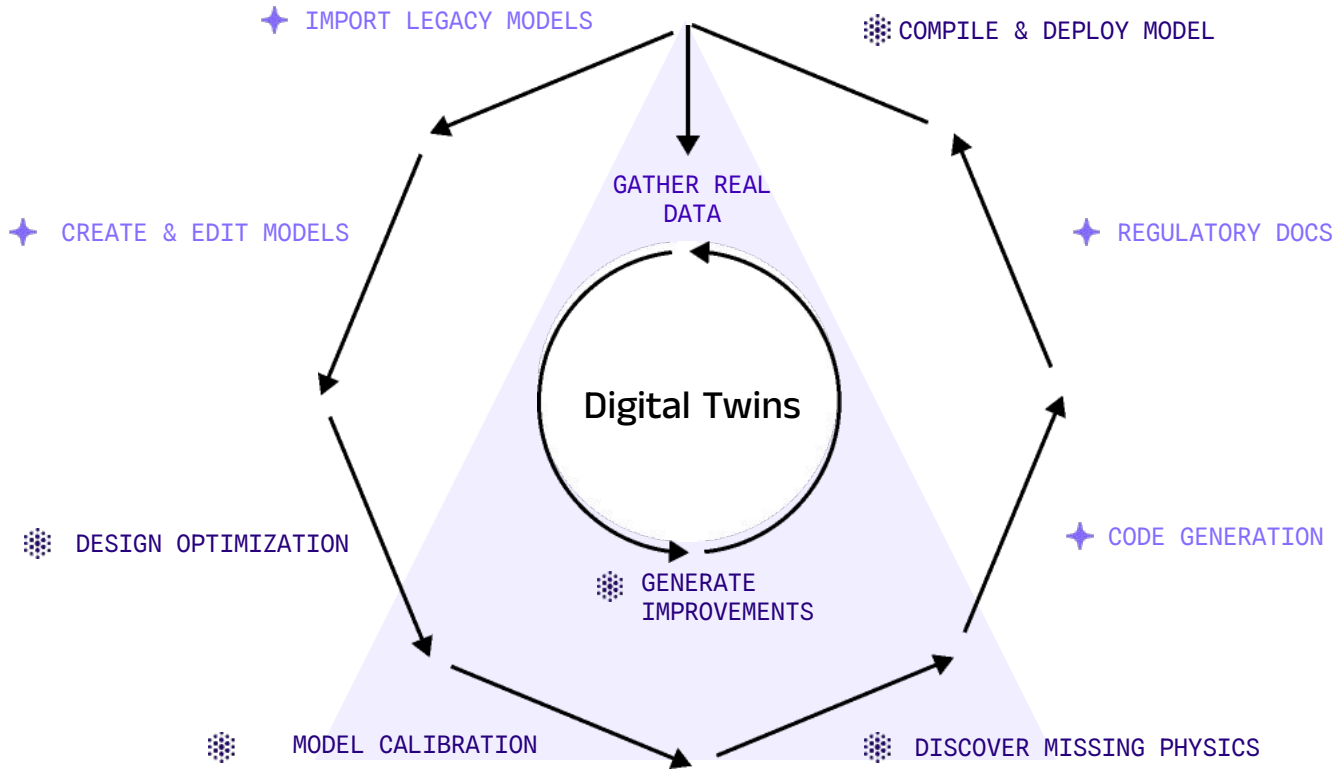
**Go To Market**

- Partnership with Ansys to include Dyad in TwinAI
- Already demonstrated customer success



AI

# Dyad delivers a powerful combination of Scientific and Generative AI



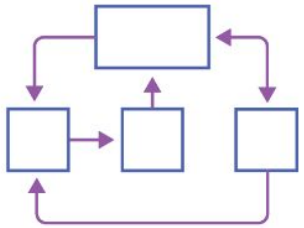
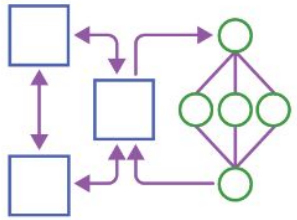
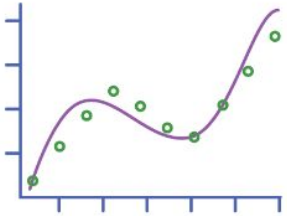
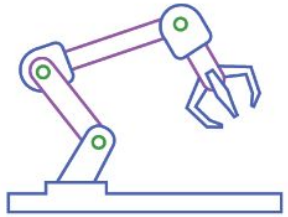
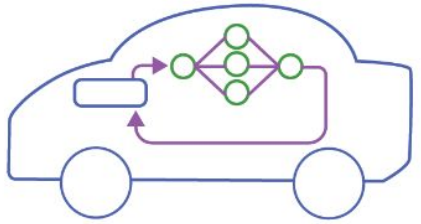

**SCIENTIFIC AI**

Through physics-constrained and data-driven modeling, Scientific AI drives increased productivity for engineers, and faster time to market

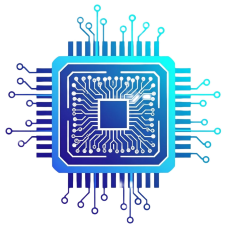
**GENERATIVE AI**

Generative AI makes it possible to break the walled gardens of legacy software which opens the door for the best compiler to win the market

# Dyad at a Glance

| Design | Discover | Calibrate | Control | Surrogatize |
|---|---|---|---|---|
| • Build realistic physical models with minimal code<br>• Run simulations 100x faster | • Use Machine Learning to autocomplete models<br>• Discover missing physics | • Turn models into Digital Twins<br>• Robust nonlinear fitting with automatic differentiation | • Build robust nonlinear controls<br>• Deploy Model-Predictive Controllers (MPC) | • Train neural networks to match models<br>• Accelerate fast simulations by another 100x |
|  |  |  |  |  |

# Dyad Standard Libraries



**ElectricalComponents**

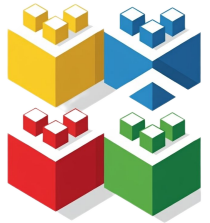
Analog/digital components as well as electrical machines

**RotationalComponents**

Rotational mechanical components used to make clutches, gears

**ThermalComponents**

Conductive and radiative components, heat transfer

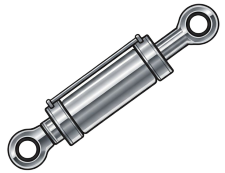


**PrimitiveComponents**

Mathematical operations for embedded programming

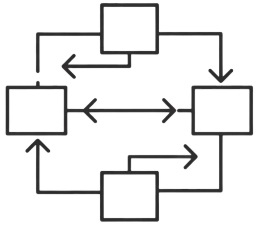
**TranslationalComponents**

Flanges, masses, springs, dampers, forces, translational mechanics components used to construct drive shaft and simple vehicle models



**HydraulicComponents**

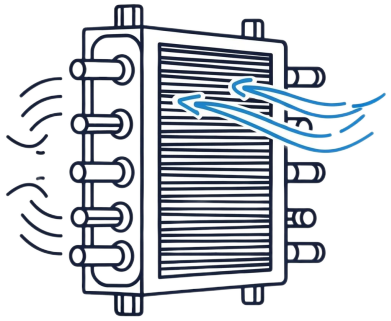
Valves, pipes, pistons



**BlockComponents**

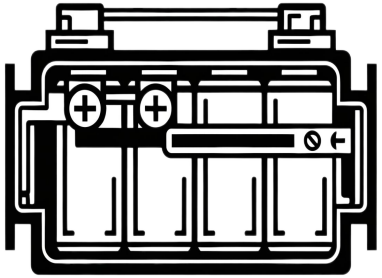
Numeric and logical components, and source blocks

# Domain-Specific In-House Libraries



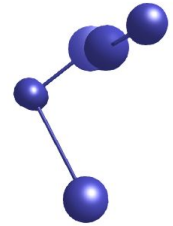
## HVAC

- Single phase and two-phase flow
- Standard library of pipes, compressors, heat exchangers, etc.
- Proprietary refrigerant and media libraries with specialized splines for high accuracy and performance in two-phase scenarios



## Batteries

- Equivalent Circuit Models (ECMs) (low complexity approximations)
- Single particle Models (SPM) and enhancements (medium complexity approximations)
- Doyle-Fuller-Newman Model (DFN) (high-complexity and high-accuracy)
- Battery degradation models
- Validated chemistry libraries (NiCd, Li-ion, etc.)



## Multibody

- Automated generation of animations and visualizations
- Kinematic loops
- Robotics
- Vehicle Dynamics
- Trajectory planning
- Aerial, Ground and Sea Vehicles such as drones, submarines

# Dyad HVAC

**Pre-built components:**

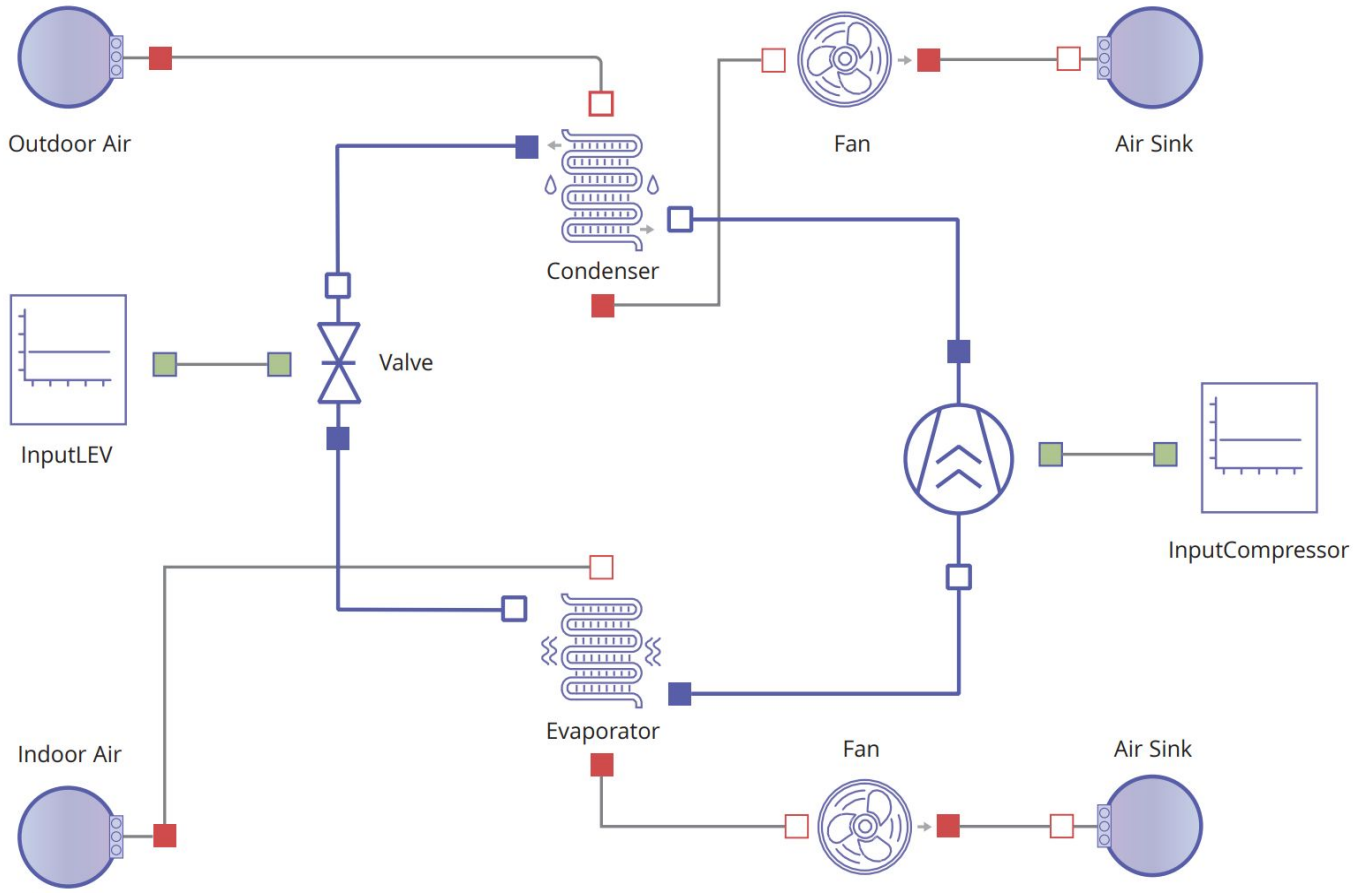
- Tube Fin, Plate heat exchangers
- Compressors, Valves, Fans, Conditioned Spaces, Pipes

**Thermodynamic Property Models:**

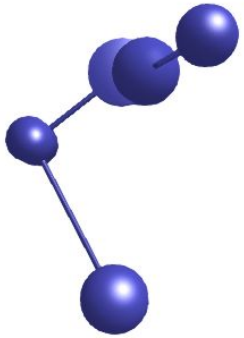
- Spline-based thermodynamic property models (R32, R1234YF, R290, R152a, R134a, R410A, R717)
- Dry and Moist air, Media models

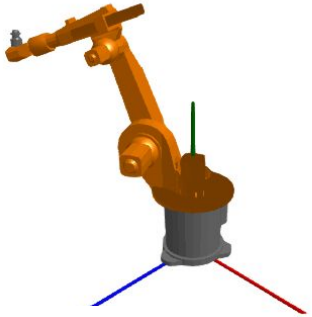
**Ecosystem:**

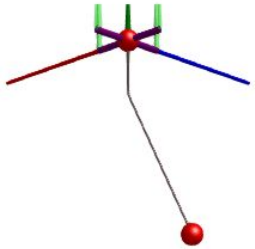
- Numerical Solvers
- Calibration, Machine Learning, Control

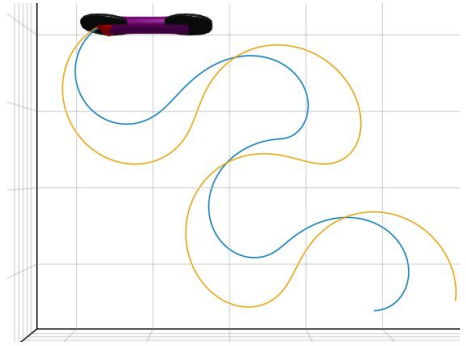


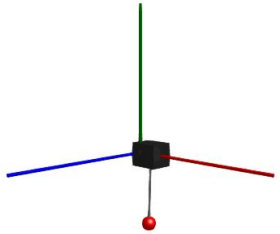
# Dyad Multibody

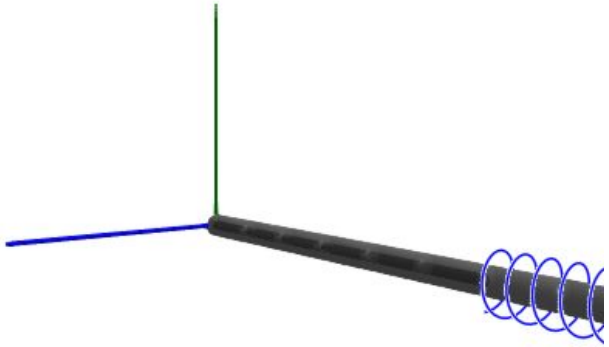


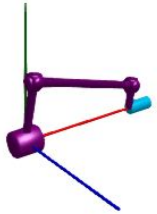


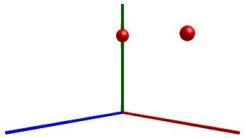


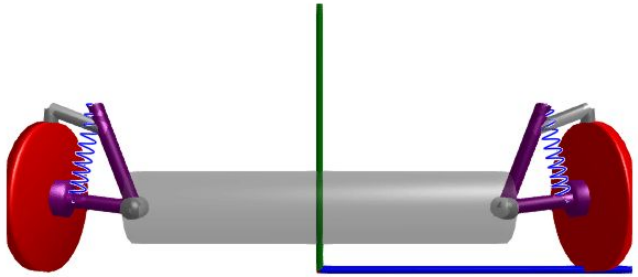


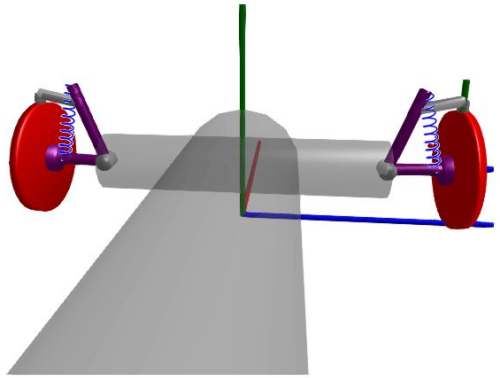










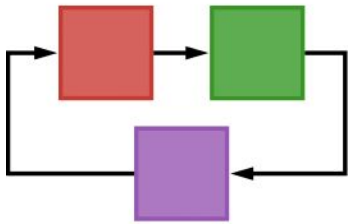


# Dyad Analysis Libraries

SciML

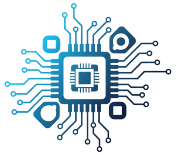
### Scientific Machine Learning (SciML) Analysis Libraries

- Model Optimizer
  - Model calibration
  - Design optimization
  - Global Optimization
- Model Discovery
  - Neural Differential Equations and Universal Differential Equations
  - Symbolic Regression model generation from data
- Neural Surrogate Generation



### Control Analysis Libraries

- System Identification
- Model Predictive Control
- Optimal Control and Trajectory Optimization
- Linear Controls
  - Generation and curation of analysis points
  - Model Order Reduction
  - Structured controller tuning
- Robust Controls
  - HInf
  - Robust MPC



fmi

### Code Generation and Deployment Analysis Libraries

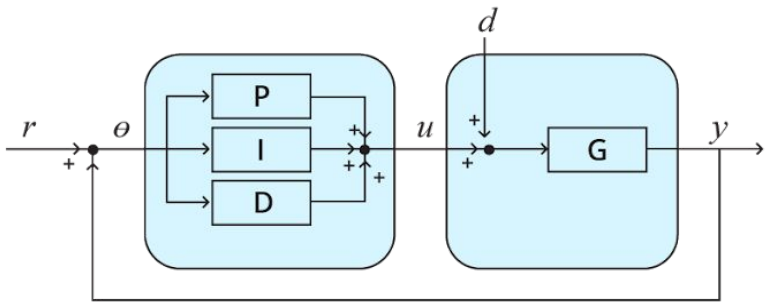
- FMU Import and Export
- Target Binary Generation
  - Generation of C code for embedded devices
  - Regulatory support (DO178, ARP26262, etc.)

# Dyad Control Systems

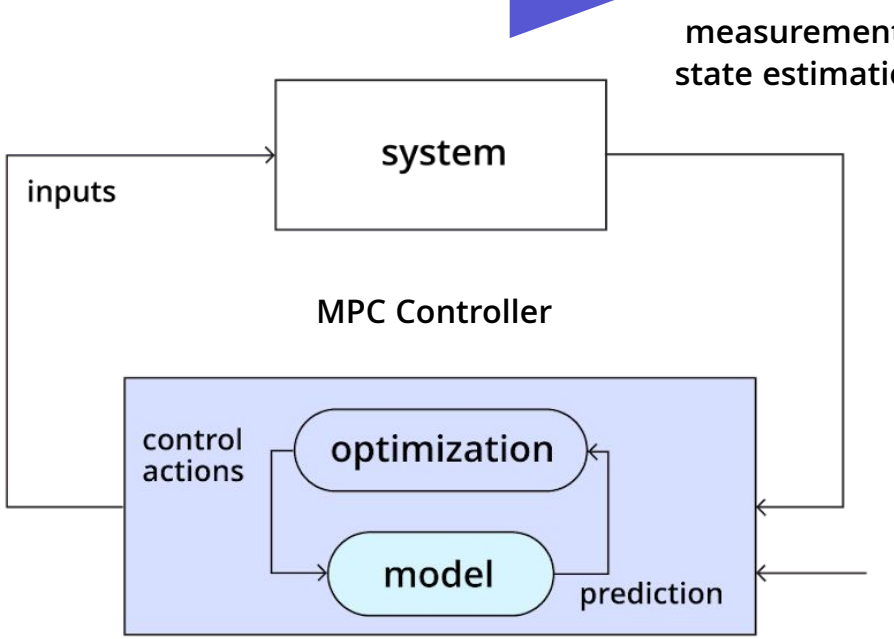
Take control of your hardware, explore your nonlinearities

Simple — **Whole spectrum** — Advanced

PID Controller




MPC Controller




**Linear control analysis and synthesis**

**Construct and tune robust Model-predictive controllers**

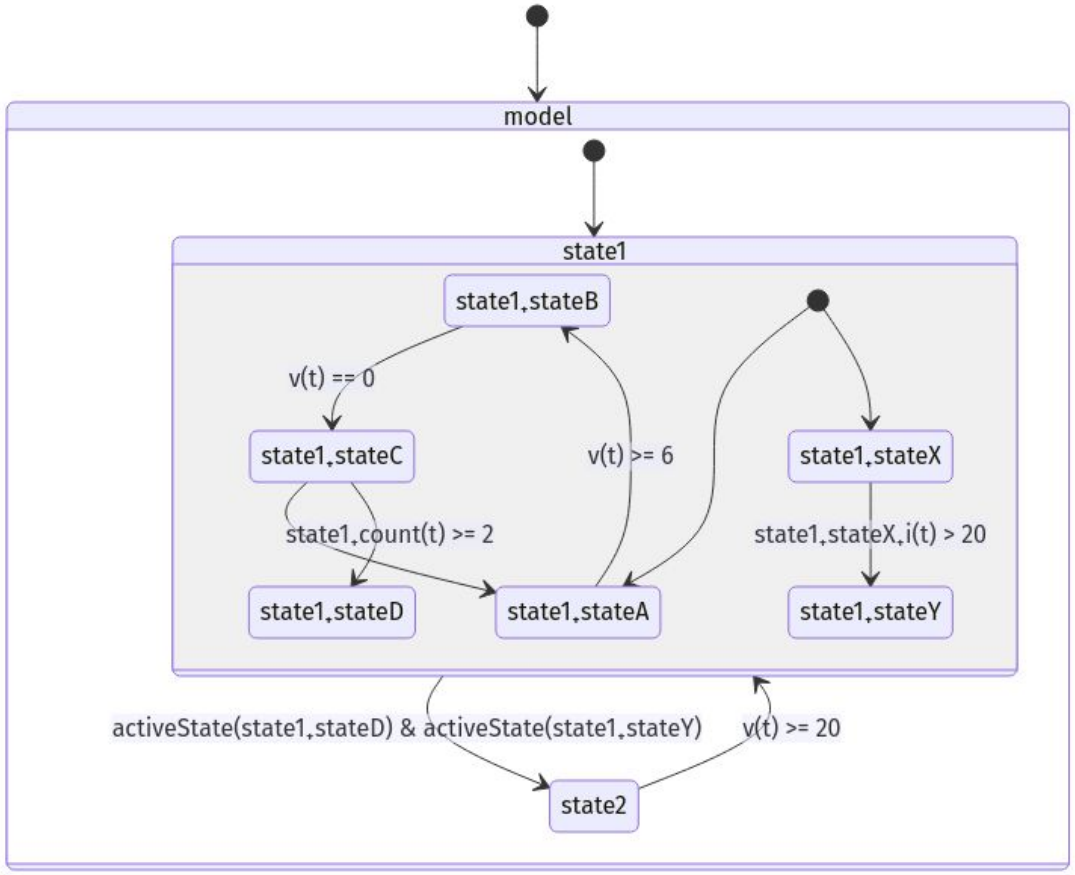
# Dyad State Machines and Synchronous Programs

Features in development:

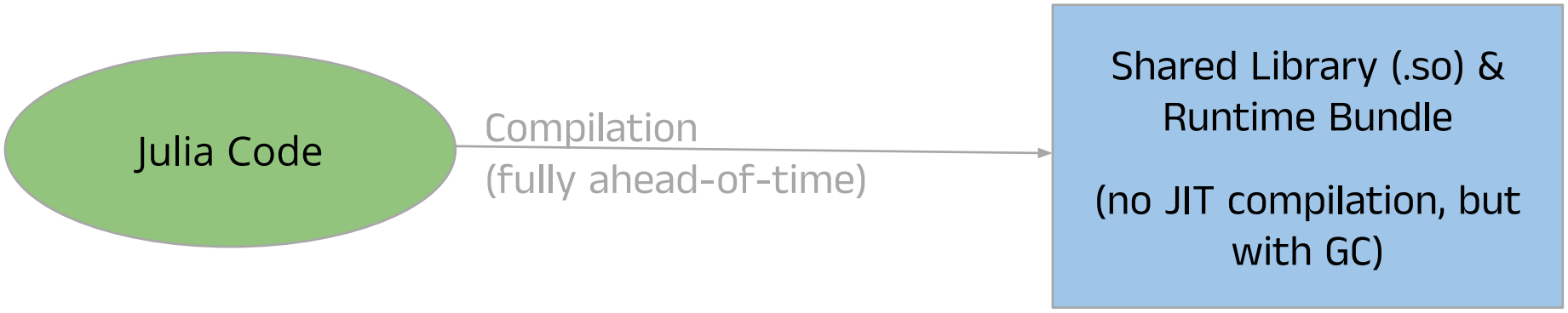
- Fundamental operations such as entry, exit, transitions, repeated operations
- Clock-based discrete time models
- Hybrid discrete-continuous systems

General Availability timeline:

- Q1 2026 for hybrid systems
- Q2 2026 for state machines



# Compiling Julia down to Hardware Targets




Supported targets: ARM v8, x86-64, x86
Supported OS: Linux, macOS, Windows

---


Planned OS/Target support including SoCs
Will support Traceability and Safety-critical software

# Agentic Model Based Design From Scratch


Thermo-mechanical model of a braking system built from scratch using Dyad

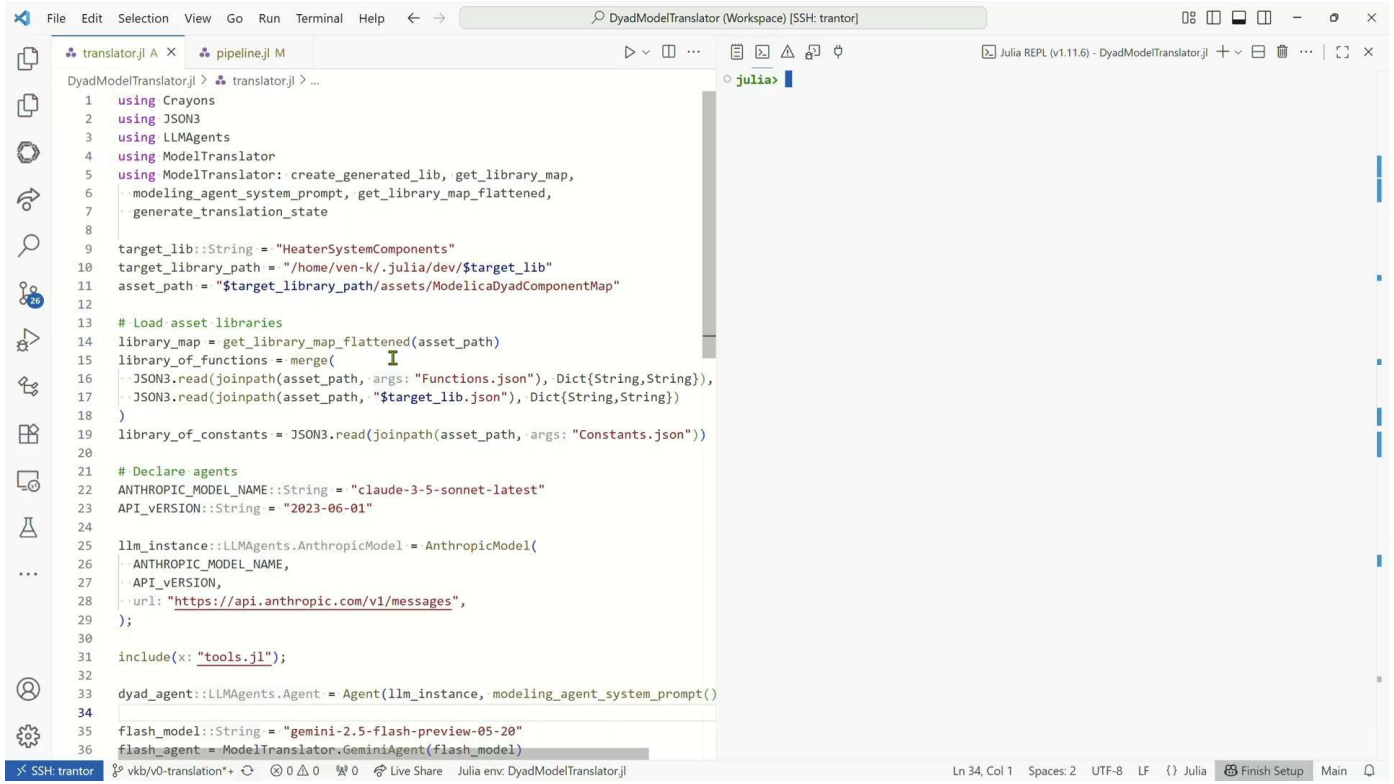
Dyad has developed an AI agent for building complex system models

Through simple queries, the agent derives the physical equations and parametrization for user feedback, after which it implements the model in Dyad.

Update parameterization of model and generate test cases

Blog post:
https://juliahub.com/blog/agentic-ai-dyad

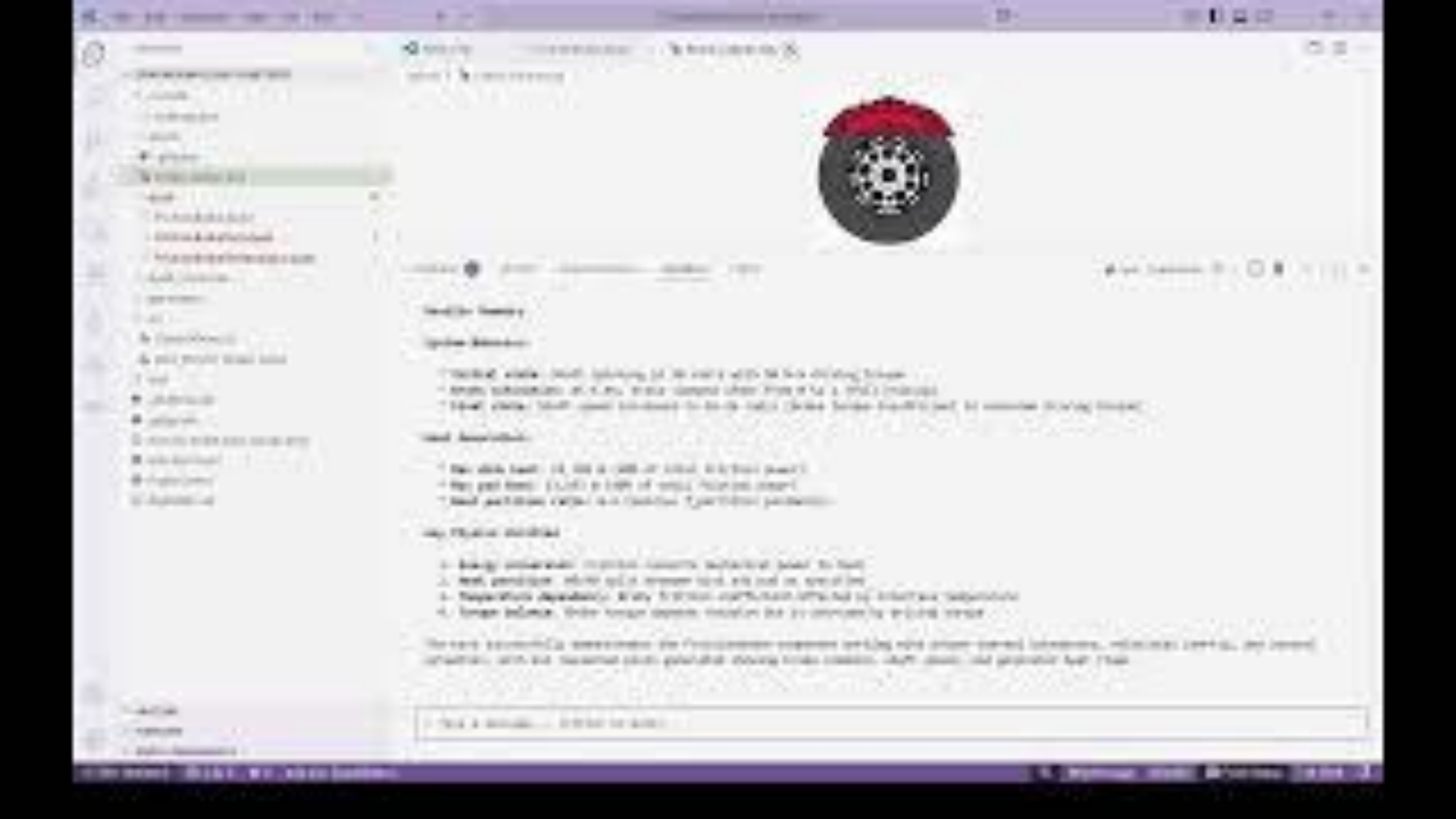
# Agentic Translation from Modelica to Dyad

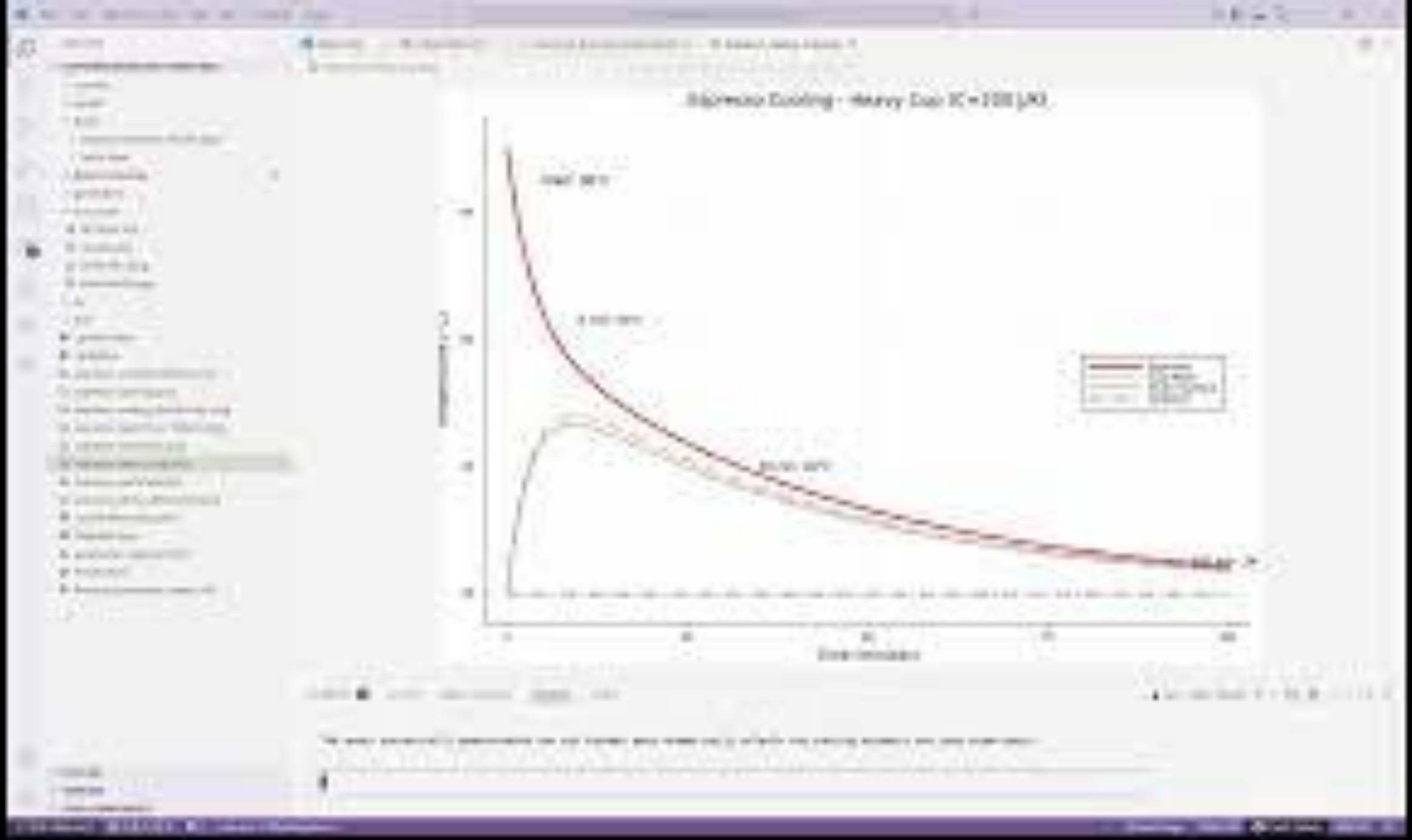



Translator decomposes model into a hierarchical tree and searches for Dyad equivalents to each node

Can easily convert converts Modelica equations to Dyad

Runs translated model through Dyad compiler and revises model based on error messages



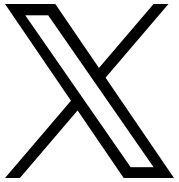

Adiabatic Cooling - Heavy Duo (C = 220 µK)

JuliaHub

# JULIASIM RECREATION OF THE NASA HL20 LIFTING BODY

# Connect with SciML

**Join our community chatrooms**
julialang.org/slack



**Follow on Twitter**
@ChrisRackauckas

**Follow on LinkedIn**
https://www.linkedin.com/in/chrisrackauckas/

**Star the GitHub repositories**
https://github.com/SciML


**Dr. Chris Rackauckas**

**VP of Modeling and Simulation**
JuliaHub

**Research Affiliate**
MIT CSAIL

**Director of Scientific Research**
Pumas-AI

# SciML Open Source Software Organization
# sciml.ai

- DifferentialEquations.jl: 2x-10x Sundials, Hairer, …
- DiffEqFlux.jl: adjoints outperforming Sundials and PETSc-TS
- ModelingToolkit.jl: 15,000x Simulink
- Catalyst.jl: >100x SimBiology, gillespy, Copasi
- DataDrivenDiffEq.jl: >10x pySindy
- NeuralPDE.jl: ~2x DeepXDE* (more optimizations to be done)
- NeuralOperators.jl: ~3x original papers (more optimizations required)
- ReservoirComputing.jl: 2x-10x pytorch-esn, ReservoirPy, PyRCN
- SimpleChains.jl: 5x PyTorch GPU with CPU, 10x Jax (small only!)
- DiffEqGPU.jl: Some wild GPU ODE solve speedups coming soon

**And 100 more libraries to mention…**

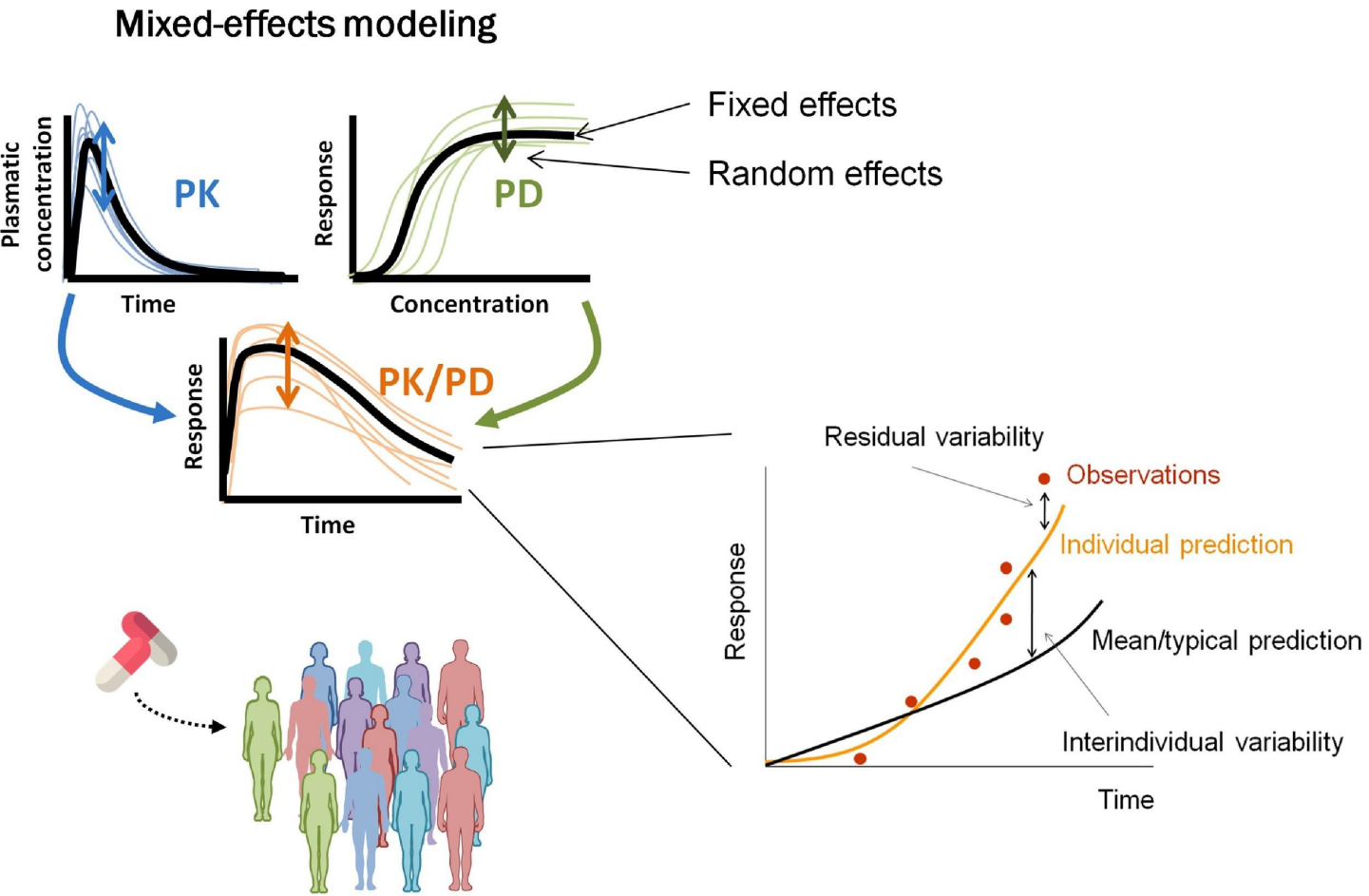
**If you work in SciML and think optimized and maintained implementations of your method would be valuable, please let us know and we can add it to the queue.**

---

**Democratizing SciML via pedantic code optimization**
**Because we believe full-scale open benchmarks matter**

# DeepNLME: Integrate neural networks into traditional NLME modeling
# DeepNLME is SciML-enhanced modeling for clinical trials

## DeepNLME is SciML-enhanced modeling for clinical trials




Trends in Pharmacological Sciences

- Automate the discovery of predictive covariates and their relationship to dynamics
- Automatically discover dynamical models and assess the fit
- Incorporate big data sources, such as genomics and images, as predictive covariates

# From Dynamics to Nonlinear Mixed Effects (NLME) Modeling

**Goal: Learn to predict patient behavior (dynamics) from simple data (covariates)**

$$Z_i = \begin{bmatrix} wt_i, \\ sex_i, \end{bmatrix}$$

**Covariates**

Math: Find $(\theta, \eta)$ such that $\mathrm{E}[\eta] = 0$

Requires special fitting procedures (Pumas)

$$g_i = \begin{bmatrix} Ka \\ CL \\ V \end{bmatrix} = \begin{bmatrix} \theta_1 e^{\eta_{i,1} \kappa_{i,k,1}}, \\ \theta_2 (\frac{wt_i}{70})^{0.75} \theta_4^{sex_i} e^{\eta_{i,2}}, \\ \theta_3 e^{\eta_{i,3}}, \end{bmatrix}$$

**Structural Model (pre)**

$$\frac{d[\text{Depot}]}{dt} = -Ka[\text{Depot}],$$
$$\frac{d[\text{Central}]}{dt} = Ka[\text{Depot}] - \frac{CL}{V}[\text{Central}].$$

**Dynamics**

Intution: $\eta$ (the random effects) are a fudge factor

Find $\theta$ (the fixed effect, or average effect) such that you can predict new patient dynamics as good as possible

# The Impact of Pumas (PharmacUtical Modeling And Simulation)

## We have been using Pumas software for our pharmacometric needs to support our development decisions and regulatory submissions.

Pumas software has surpassed our expectations on its accuracy and ease of use. We are encouraged by its capability of supporting different types of pharmacometric analyses within one software. **Pumas has emerged as our "go-to" tool for most of our analyses in recent months.** We also work with Pumas-AI on drug development consulting. We are impressed by the quality and breadth of the experience of Pumas-AI scientists in collaborating with us on modeling and simulation projects across our pipeline spanning investigational therapeutics and vaccines at various stages of clinical development

**Husain A. PhD (2020)**
Director, Head of Clinical Pharmacology and Pharmacometrics,
*Moderna Therapeutics, Inc*

moderna™
messenger therapeutics

# From Dynamics to Nonlinear Mixed Effects (NLME) Modeling

**Goal: Learn to predict patient behavior (dynamics) from simple data (covariates)**

$$Z_i = \begin{bmatrix} wt_i, \\ sex_i, \end{bmatrix}$$

**Covariates**

Math: Find $(\theta, \eta)$ such that $\mathrm{E}[\eta] = 0$

$$g_i = \begin{bmatrix} Ka \\ CL \\ V \end{bmatrix} = \begin{bmatrix} \theta_1 e^{\eta_{i,1} \kappa_{i,k,1}}, \\ \theta_2 (\frac{wt_i}{70})^{0.75} \theta_4^{sex_i} e^{\eta_{i,2}}, \\ \theta_3 e^{\eta_{i,3}}, \end{bmatrix}$$

**Structural Model (pre)**

**How can we find these models?**

$$\frac{d[\text{Depot}]}{dt} = -Ka[\text{Depot}],$$
$$\frac{d[\text{Central}]}{dt} = Ka[\text{Depot}] - \frac{CL}{V}[\text{Central}].$$

**Dynamics**

Intution: $\eta$ (the random effects) are a fudge factor

Find $\theta$ (the fixed effect, or average effect) such that you can predict new patient dynamics as good as possible

# From Dynamics to Nonlinear Mixed Effects (NLME) Modeling

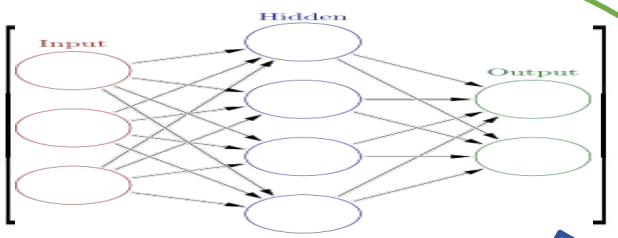
**Goal: Learn to predict patient behavior (dynamics) from simple data (covariates)**

$$Z_i = \begin{bmatrix} wt_i, \\ sex_i, \end{bmatrix}$$

**Covariates**

Math: Find $(\theta, \eta)$ such that $\mathrm{E}[\eta] = 0$

$$g_i = \begin{bmatrix} Ka \\ CL \\ V \end{bmatrix} =$$

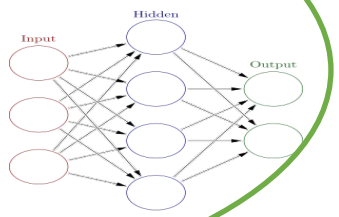



**Structural Model (pre)**

**How can we find these models?**

$$\frac{d[\text{Depot}]}{dt} = -Ka[\text{Depot}],$$

$$\frac{d[\text{Central}]}{dt} = Ka[\text{Depot}] -$$




**Dynamics**

**Idea: Parameterize the model such that the models can be neural networks, where the weights of the neural networks are fixed effects!**

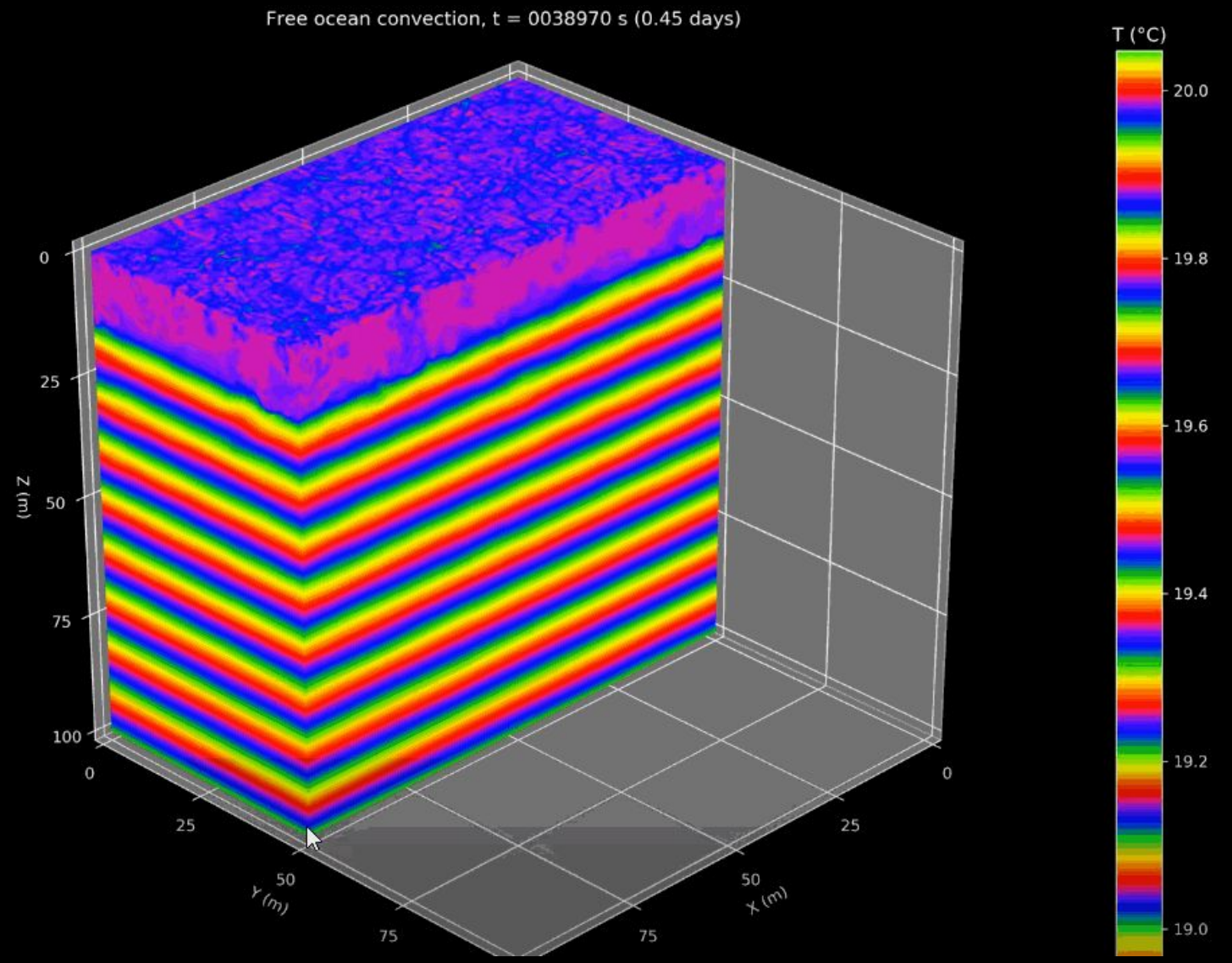
**Indirect learning of unknown functions!**

# How are the networks trained?

# High fidelity surrogates of ocean columns for climate models

3D simulations are high resolution but too expensive.

**Can we learn faster models?**





Ramadhan, Ali, John Marshall, Andre Souza, Gregory LeClaire Wagner, Manvitha Ponnapati, and Christopher Rackauckas. "Capturing missing physics in climate model parameterizations using neural differential equations." *arXiv preprint arXiv:2010.12559* (2020).


(Update going online in the next month)

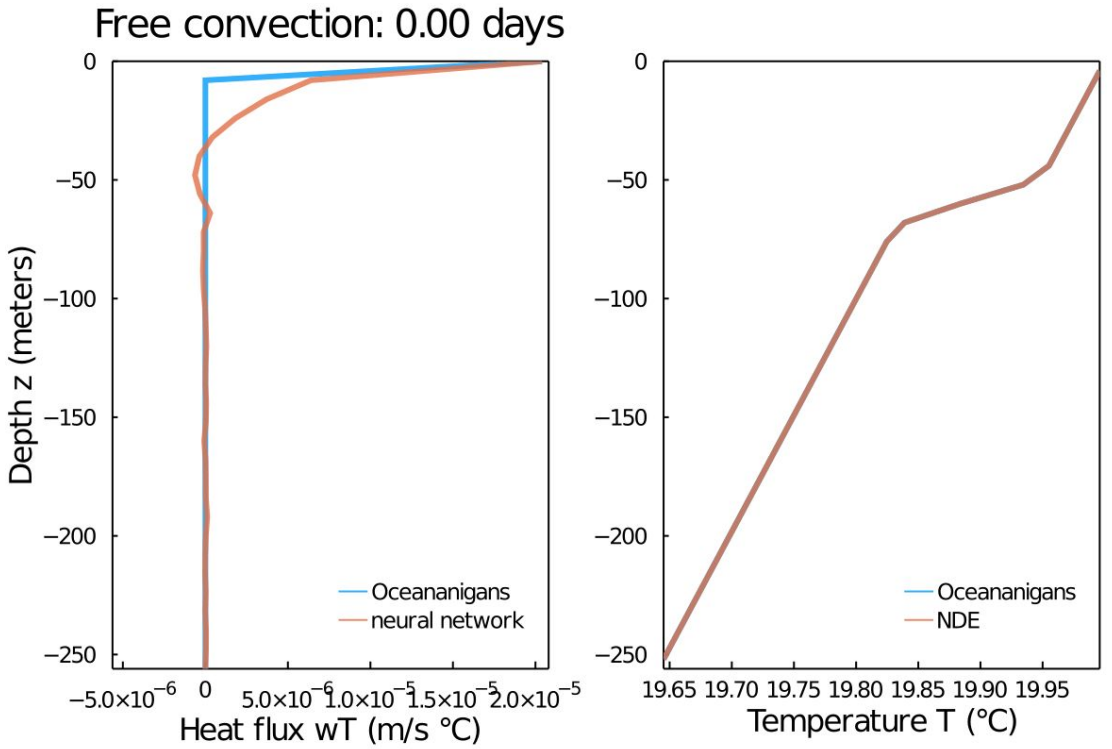
# Neural Networks Infused into Known Partial Differential Equations

## Derive a 1D approximation to the 3D model

$$\frac{\partial T}{\partial t} = -\frac{\partial}{\partial z}\underbrace{\left(\text{NN} - K\frac{\partial T}{\partial z}\right)}_{\overline{w'T'}}$$

## Incorporate the "convective adjustment"

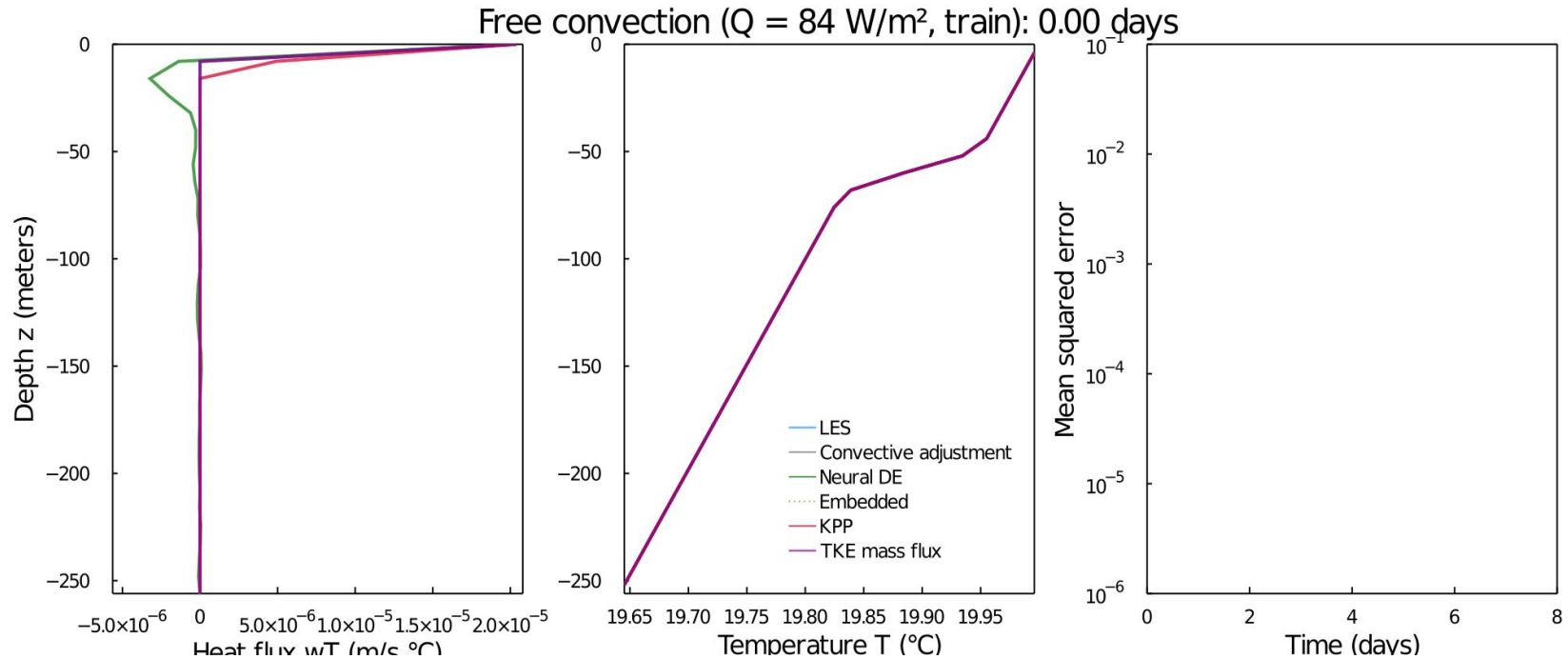
$$K = \begin{cases} 0 & \text{if } \partial_z T > 0 \\ 100\ \text{m}^2/\text{s} & \text{if } \partial_z T < 0 \end{cases}$$




$$loss(T, wT) = |NN(T) - wT|^2$$

Only okay, **but why?**

# Good Engineering Principles: Integral Control!

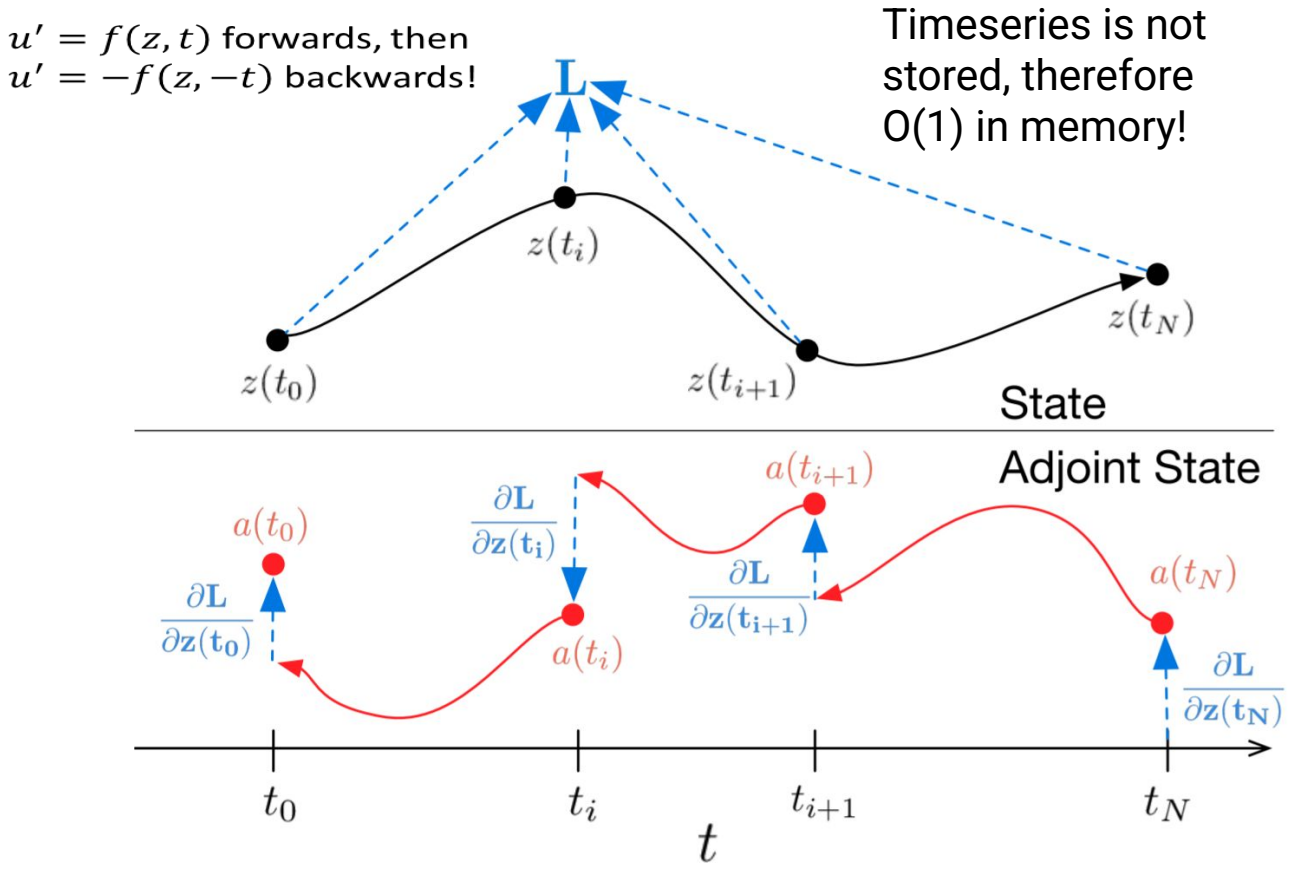



$$\frac{\partial T}{\partial t} = -\frac{\partial}{\partial z}\underbrace{\left(\text{NN}(T) - K\frac{\partial T}{\partial z}\right)}_{\overline{w'T'}}$$

$$loss(T_{NN}, T) = |T_{NN}(z,t) - T(z,t)|^2$$

# But differentiation is just automatic differentiation… right?

# Machine Learning Neural Ordinary Differential Equations




Timeseries is not stored, therefore O(1) in memory!

The adjoint equation is an ODE!

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

How do you get z(t)? One suggestion: Reverse the ODE

$$\frac{d\mathbf{a}_{aug}(t)}{dt} = -\begin{bmatrix}\mathbf{a}(t) & \mathbf{a}_\theta(t) & \mathbf{a}_t(t)\end{bmatrix} \frac{\partial f_{aug}}{\partial[\mathbf{z}, \theta, t]}(t)$$

Chen, Ricky TQ, et al. "Neural ordinary differential equations." Advances in neural information processing systems. 2018.

# “Adjoints by reversing” also is unconditionally unstable on some problems!

Advection Equation:

$$\frac{\partial u}{\partial t} + \frac{a(\partial u)}{\partial x} = 0$$

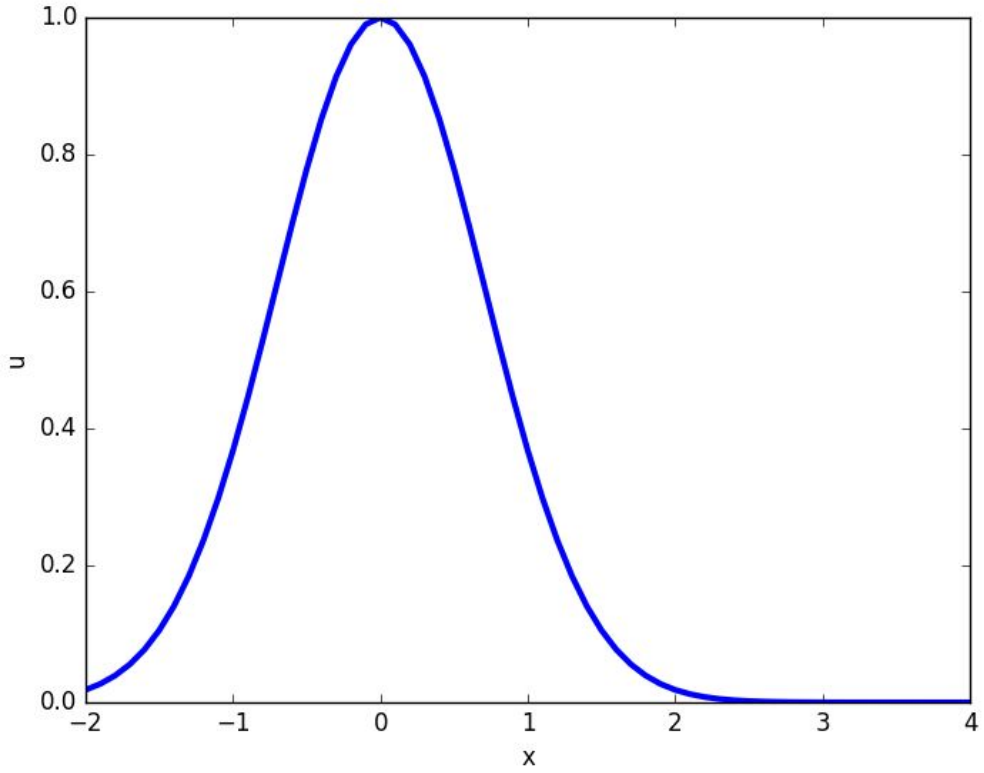
Approximating the derivative in $x$ has two choices: forwards or backwards

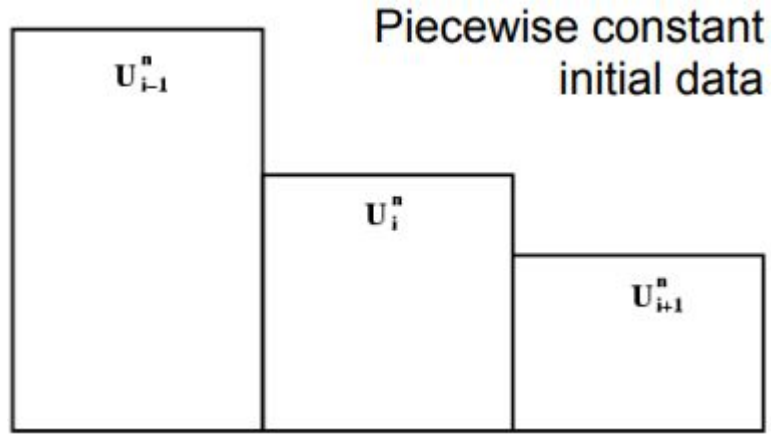
$$u_i' = -\frac{a(u_i - u_{i-1})}{\Delta x} \text{ or } u_i' = -\frac{a(u_{i+1} - u_i)}{\Delta x}?$$

If you discretize in the wrong direction you get **unconditional instability**

**You need to understand the engineering principles and the numerical simulation properties of domain to make ML stable on it.**

Result: the method in the PyTorch library is unconditionally unstable!!!

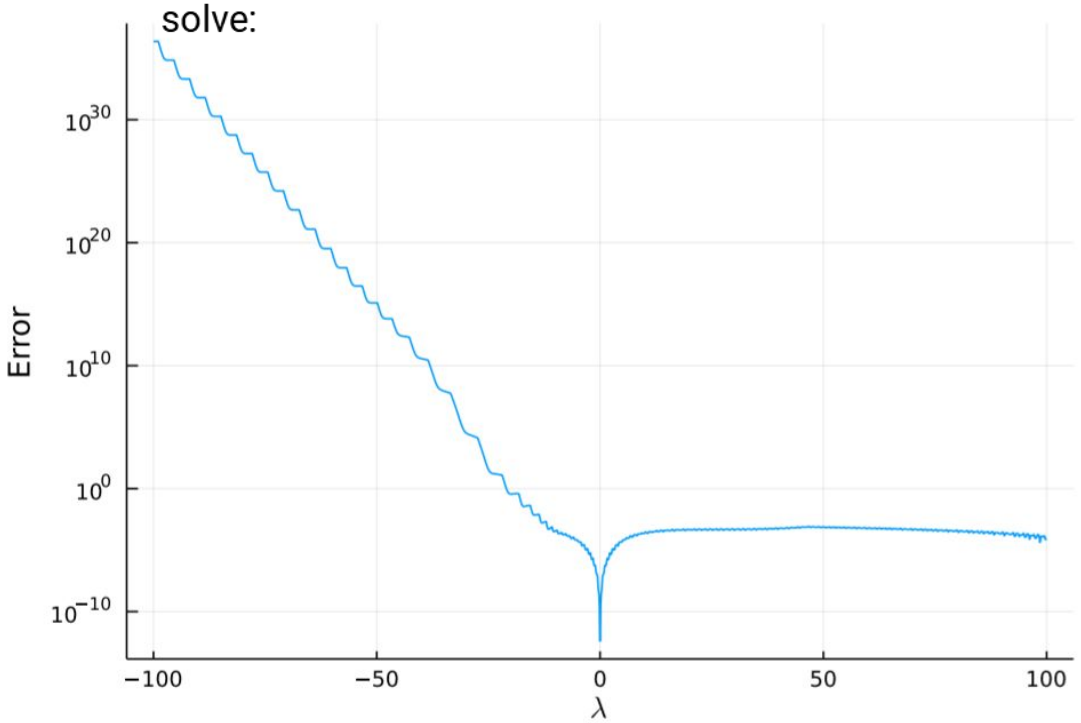


# Problems With Naïve Adjoint Approaches On Stiff Equations

**Error grows exponentially…**

$u'(t) = \lambda u(t)$, plot the error in the reverse solve:

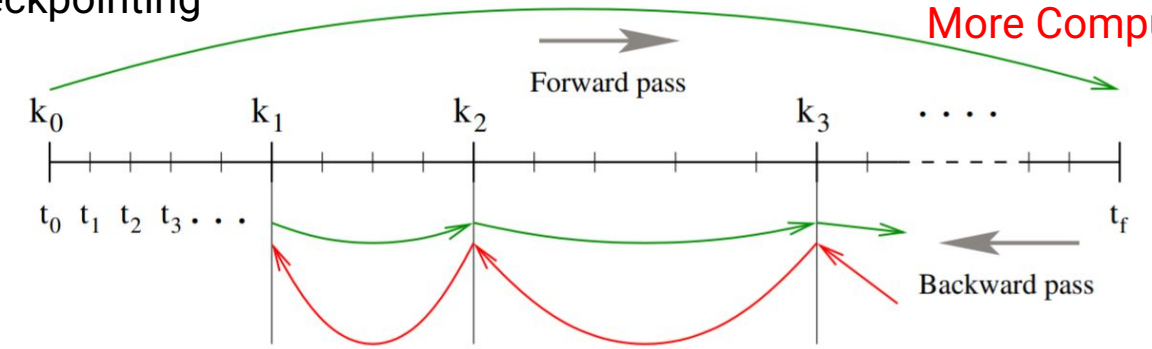



How do you get u(t) while solving backwards?
3 options!

1. $u' = f(z,t)$ forwards, then $u' = -f(z,-t)$ backwards! Unstable

2. Store u(t) while solving forwards (dense output) High memory

3. Checkpointing More Compute



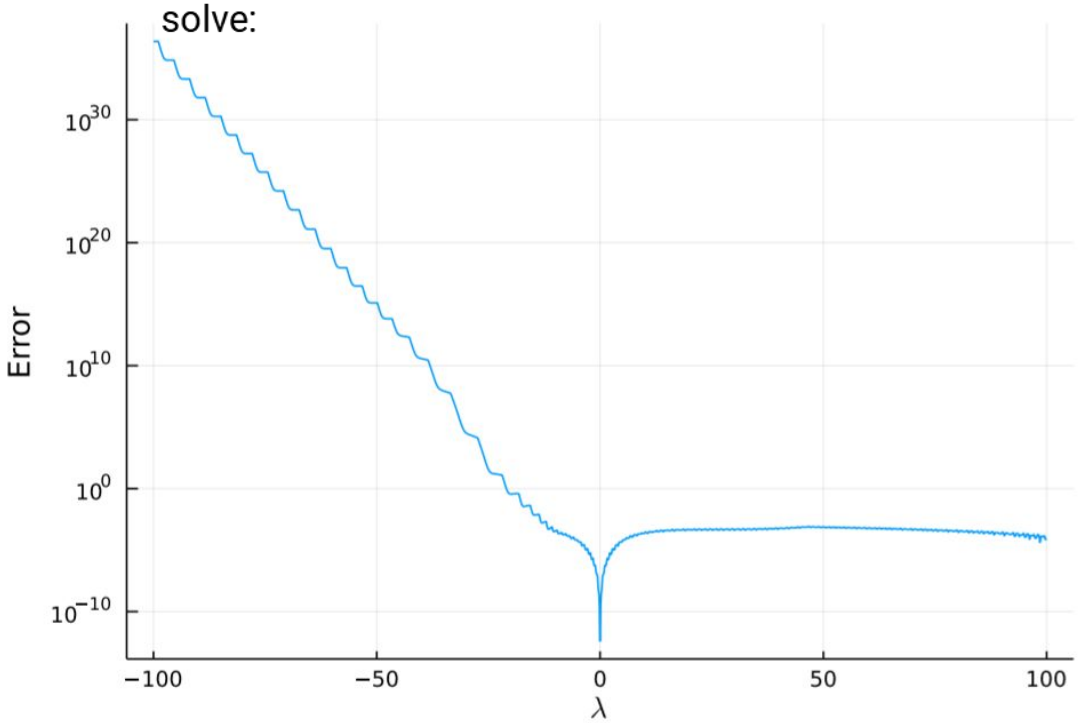

Each choices has an engineering trade-off!

Kim, Suyong, Weiqi Ji, Sili Deng, and Christopher Rackauckas. "Stiff neural ordinary differential equations." *Chaos* (2021).

# Problems With Naïve Adjoint Approaches On Stiff Equations

**Error grows exponentially…**

$u'(t) = \lambda u(t)$, plot the error in the reverse solve:




**Compute cost is cubic with parameter size when stiff**

Size of reverse ODE system is:

$$2states + parameters$$

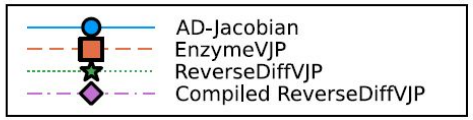
Linear solves inside of stiff ODE solvers, ~cubic

Thus, adjoint cost:

$$O\left((states + parameters)^3\right)$$

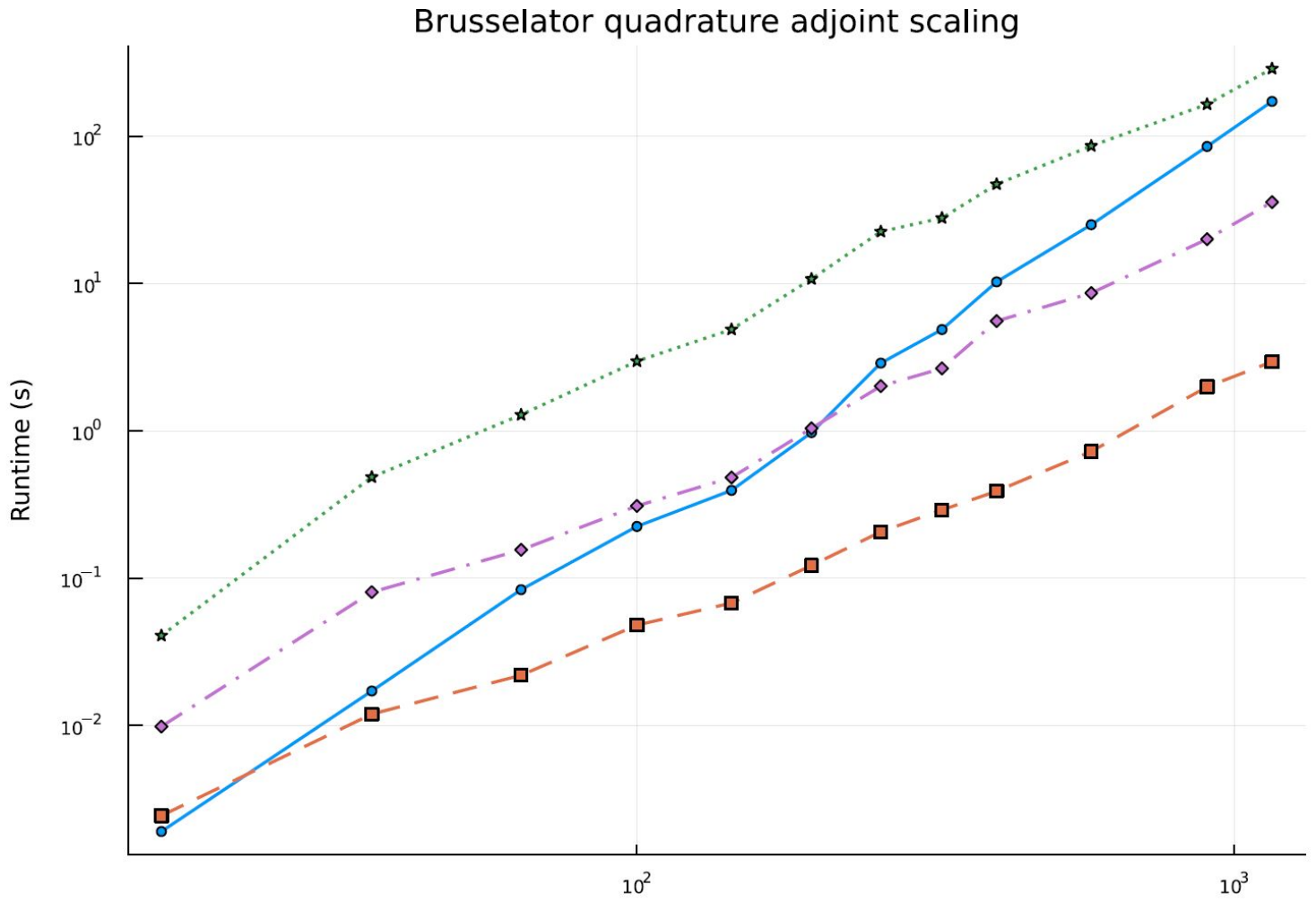
Kim, Suyong, Weiqi Ji, Sili Deng, and Christopher Rackauckas. "Stiff neural ordinary differential equations." *Chaos* (2021).

# Gives about a 10x performance improvement for adjoint calculations on PDEs




Ma, Yingbo, Vaibhav Dixit, Michael J. Innes, Xingjian Guo, and Chris Rackauckas. "A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions." In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1-9. IEEE, 2021.

**"Adjoint" can mean many many different things…**



# The Test Problem

```
using OrdinaryDiffEq, ForwardDiff, Zygote, SciMLSensitivity

function f!(du, u, p, t)
    du[1] = p[1] * (u[1] - u[2])
    du[2] = p[2] * (u[2] - u[1])
end
p = [1.0, 1.0];
u0 = [1.0; 1.0];
prob = ODEProblem(f!, u0, (0.0, 5.0), p)
sol = solve(prob, Tsit5(), saveat = 0.1)

function cost(u0)
    _prob = remake(prob, u0 = u0)
    solve(_prob, Tsit5(), reltol = 1e-12, abstol = 1e-12, saveat = 0.1)[1, :]
end
```

# Julia Derivative Results

```julia
function cost(u0)
    _prob = remake(prob, u0 = u0)
    solve(_prob, Tsit5(), reltol = 1e-12,
          abstol = 1e-12, saveat = 0.1)[1,
end
dx = ForwardDiff.jacobian(cost, u0)[end, :]
#=
2-element Vector{Float64}:
  11013.732897407262
 -11012.732897407262
=#
dx2 = Zygote.jacobian(cost, u0)[1][end, :]
#=
2-element Vector{Float64}:
  11013.732897407262
 -11012.732897407262
=#
```

All methods compute the same derivative to many digits

```julia
function cost(u0)
    _prob = remake(prob, u0 = u0)
    solve(_prob, Tsit5(), reltol = 1e-6,
          abstol = 1e-6, saveat = 0.1)[1, :]
end
dx = ForwardDiff.jacobian(cost, u0)[end, :]
#=
2-element Vector{Float64}:
  11013.73051284873
 -11012.73051284873
=#

dx2 = Zygote.jacobian(cost, u0)[1][end, :]
#=
2-element Vector{Float64}:
  11013.73051284873
 -11012.73051284873
=#
```

# Jax Implementation of the Test Problem

```python
import jax
import jax.numpy as jnp
from diffrax import diffeqsolve, ODETerm, Tsit5,
                    PIDController, BacksolveAdjoint,
                    RecursiveCheckpointAdjoint, DirectAdjoint

from jax import config
config.update("jax_enable_x64", True)

def vector_field(t, u, args):
    x, y = u
    a, b = args
    dx = a * (x - y)
    dy = b * (y - x)
    return dx, dy
```

```python
def run(y0, adjoint = RecursiveCheckpointAdjoint(), tol = 1e-12):
    term = ODETerm(vector_field)
    solver = Tsit5(scan_kind="bounded")
    stepsize_controller = PIDController(rtol=tol, atol=tol)
    t0 = 0
    t1 = 5.0
    dt0 = 0.1
    p0 = (1.0, 1.0)
    sol = diffeqsolve(term, solver, t0, t1, dt0, y0,
                      adjoint = adjoint,
                      stepsize_controller = stepsize_controller,
                      args=p0)
    ((x,), _) = sol.ys
    return x
```

**Jax requires a lot more code**

# Jax Derivative Results: BacksolveAdjoint

```
J = jax.jacrev(lambda y0: run(y0, BacksolveAdjoint()))(y0)
# (Array(11013.73289742, dtype=float64, weak_type=True),
# Array(-11012.73289742, dtype=float64, weak_type=True))

J = jax.jacrev(lambda y0: run(y0, BacksolveAdjoint(), tol = 1e-3))(y0)
# (Array(10869.87401012, dtype=float64, weak_type=True),
# Array(-10868.87401012, dtype=float64, weak_type=True))

y1 = (jnp.array(1.000001), jnp.array(1.0))
y0 = (jnp.array(1.0), jnp.array(1.0))
(run(y1) - run(y0)) / .000001
# Array(11013.73156938, dtype=float64)
```

**BacksolveAdjoint works here, but it's the unstable one!**

# Jax Derivative Results: RecursiveCheckpointingAdjoint and DirectAdjoint

```
y0 = (jnp.array(1.0), jnp.array(1.0))
J = jax.jacrev(run)(y0)
# (Array(3755.79674193, dtype=float64, weak_type=True),
# Array(-3754.79674193, dtype=float64, weak_type=True))

J = jax.jacrev(lambda y0: run(y0, RecursiveCheckpointAdjoint(), tol = 1e-3))(y0)
# (Array(3755.79674193, dtype=float64, weak_type=True)
# Array(-3754.79674193, dtype=float64, weak_type=True))

J = jax.jacfwd(lambda y0: run(y0, DirectAdjoint()))(y0)
# (Array(3755.79674193, dtype=float64), Array(-3754.79674193, dtype=float64))

J = jax.jacfwd(lambda y0: run(y0, DirectAdjoint(), tol = 1e-3))(y0)
# (Array(3755.79674193, dtype=float64), Array(-3754.79674193, dtype=float64))
```

**Jax's other methods don't converge, >60% error!**

**Conclusion:**

**The PyTorch library uses a numerically unstable method**

**The Jax library uses a non-convergent method.**

**The Julia library is numerically stable and convergent.**

**Calculating derivatives is hard.**

# But… any solver you can differentiate can do UDE things

# Improving Coverage of Automatic Differentiation over Solvers

LinearSolve.jl: Unified Linear Solver Interface

$$A(p)x = b$$

NonlinearSolve.jl: Unified Nonlinear Solver Interface

$$f(u,p) = 0$$

DifferentialEquations.jl: Unified Interface for all Differential Equations

$$u' = f(u,p,t)$$

$$du = f(u,p,t)dt + g(u,p,t)dW$$

$$\vdots$$

Optimization.jl: Unified Optimization Interface

$$\begin{aligned} &\text{minimize } f(u,p) \\ &\text{subject to } g(u,p) \leq 0, h(u,p) \end{aligned}$$

Integrals.jl: Unified Quadrature Interface

$$\int_{lb}^{ub} f(t,p)dt$$

Unified Partial Differential Equation Interface

$$u_t = u_{xx} + f(t)$$

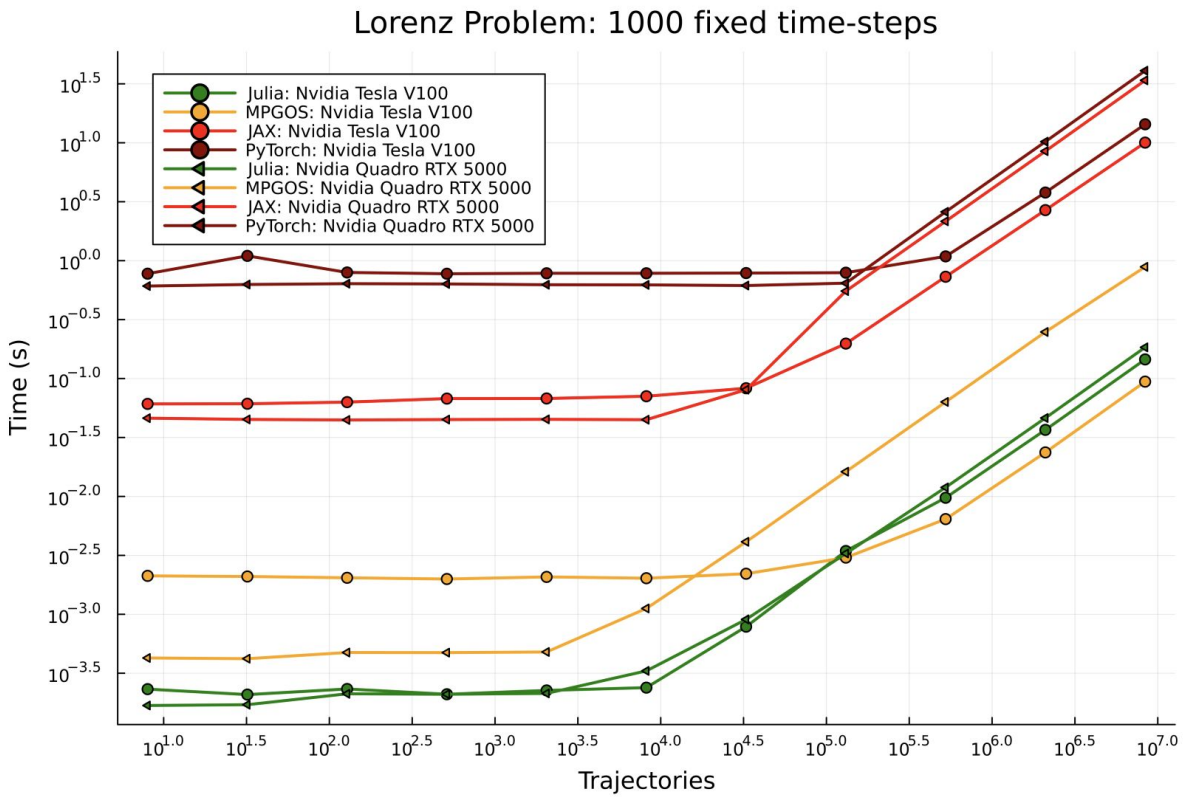
$$u_{tt} = u_{xx} + f(t)$$

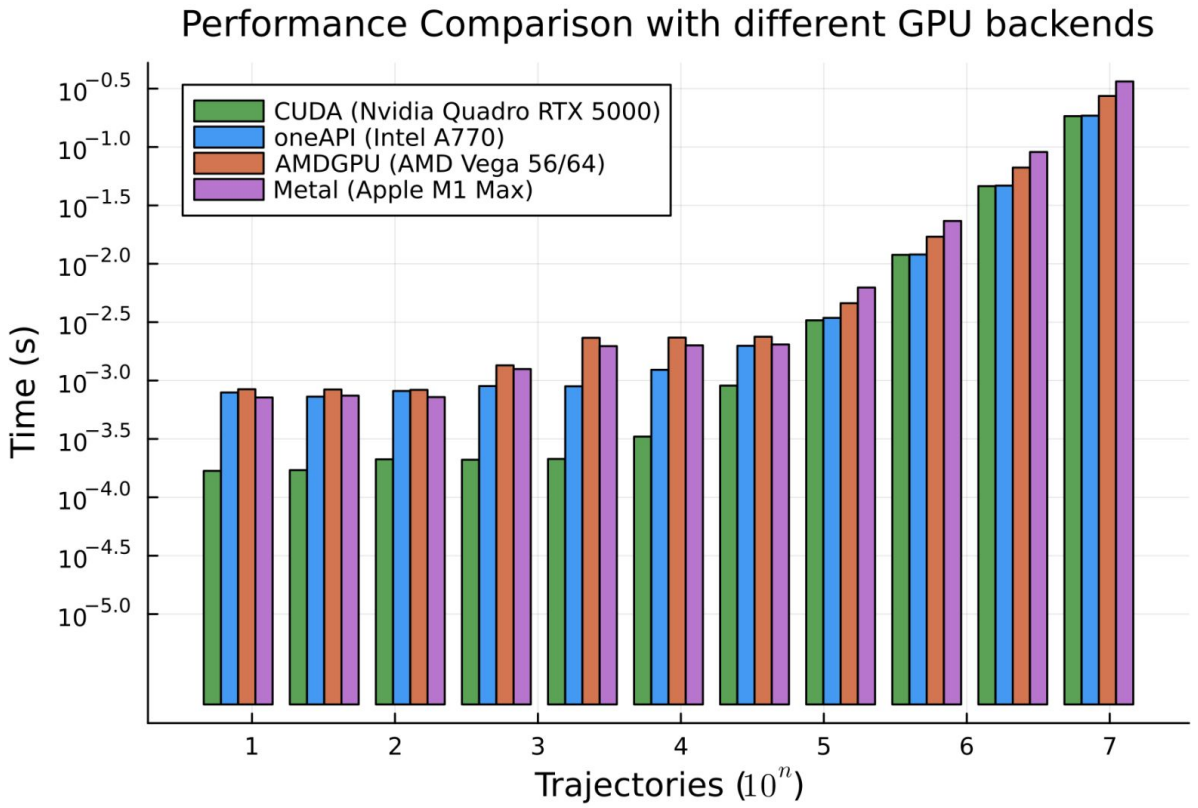
$$\vdots$$

**The SciML Common Interface for Julia Equation Solvers**

https://scimlbase.sciml.ai/dev/

# New Parallelized GPU ODE Parallelism: 20x-100x Faster than Jax and PyTorch




**Matches State of the Art on CUDA, but also works with AMD, Intel, and Apple GPUs**


Utkarsh, U., Churavy, V., Ma, Y., Besard, T., Srisuma, P., Gymnich, T., ... & Rackauckas, C. (2024). Automated translation and accelerated solving of differential equations on multiple GPU platforms. *Computer Methods in Applied Mechanics and Engineering*, *419*, 116591.

# Foundation: Fast Differential Equation Solvers

1. Speed
2. Stability
3. Stochasticity
4. Adjoints and Inference
5. Parallelism


**DifferentialEquations.jl is generally:**

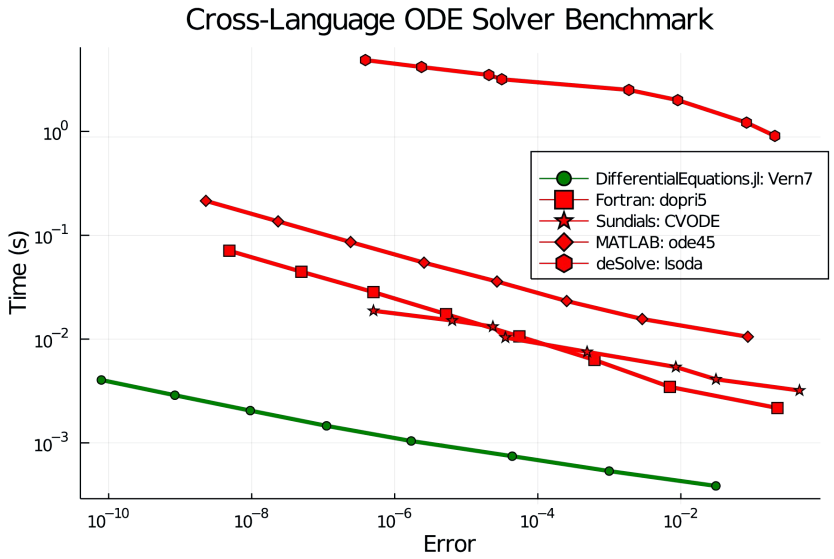
- 50x faster than SciPy
- 50x faster than MATLAB
- 100x faster than R's deSolve

https://github.com/SciML/SciMLBenchmarks.jl

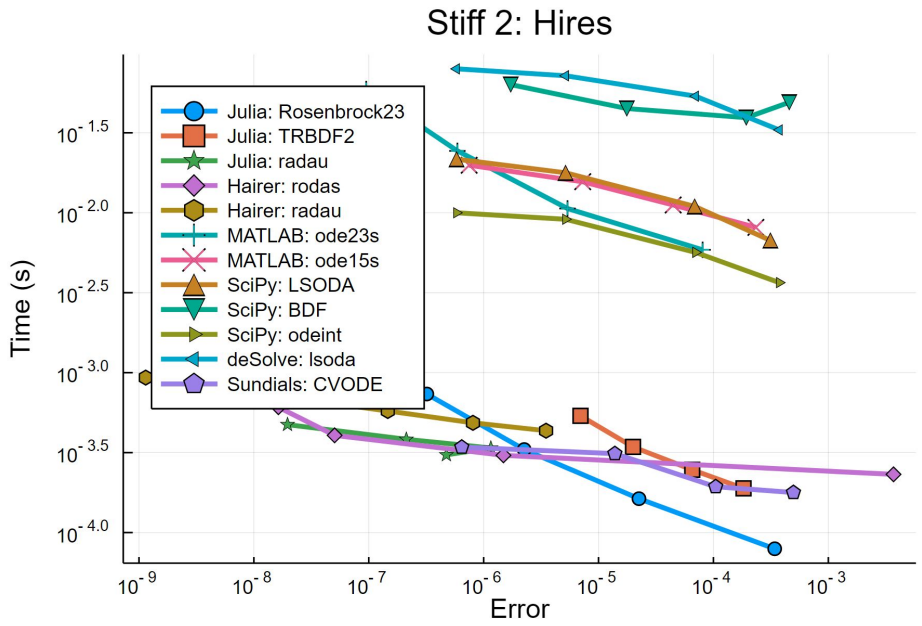
Rackauckas, Christopher, and Qing Nie. "Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia." Journal of Open Research Software 5.1 (2017).

Rackauckas, Christopher, and Qing Nie. "Confederated modular differential equation APIs for accelerated algorithm development and benchmarking." Advances in Engineering Software 132 (2019): 1-6.

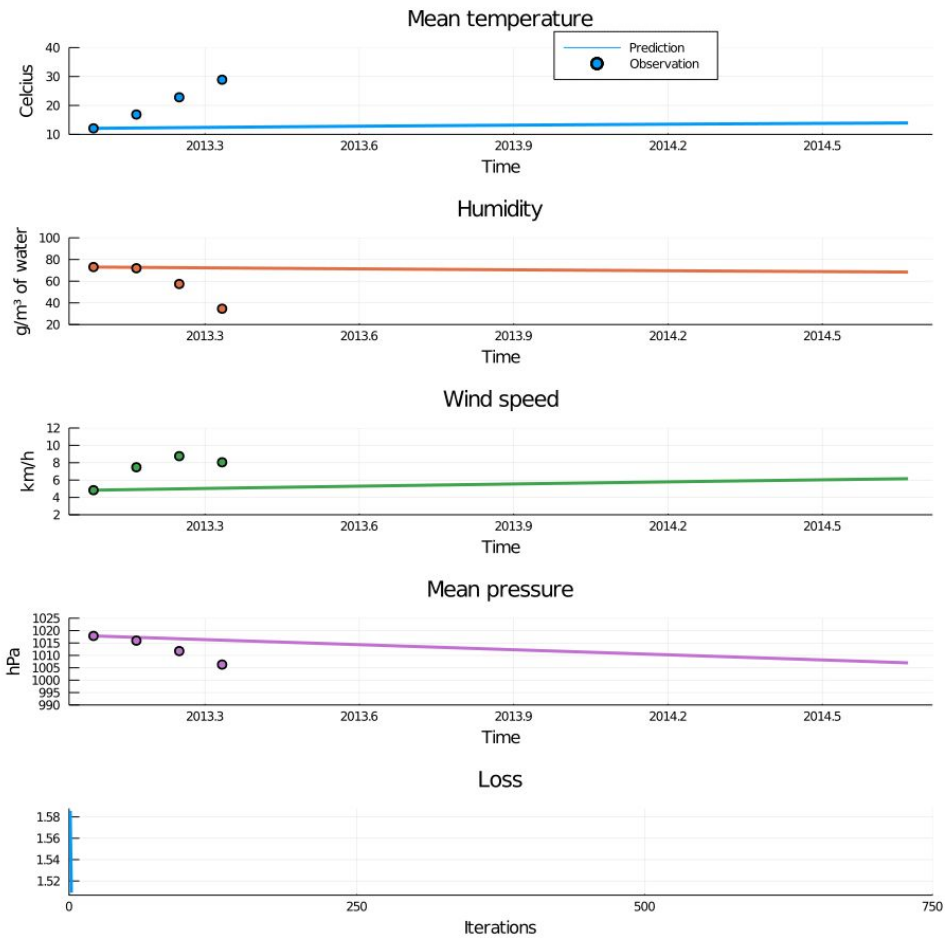
## Non-Stiff ODE: Rigid Body System

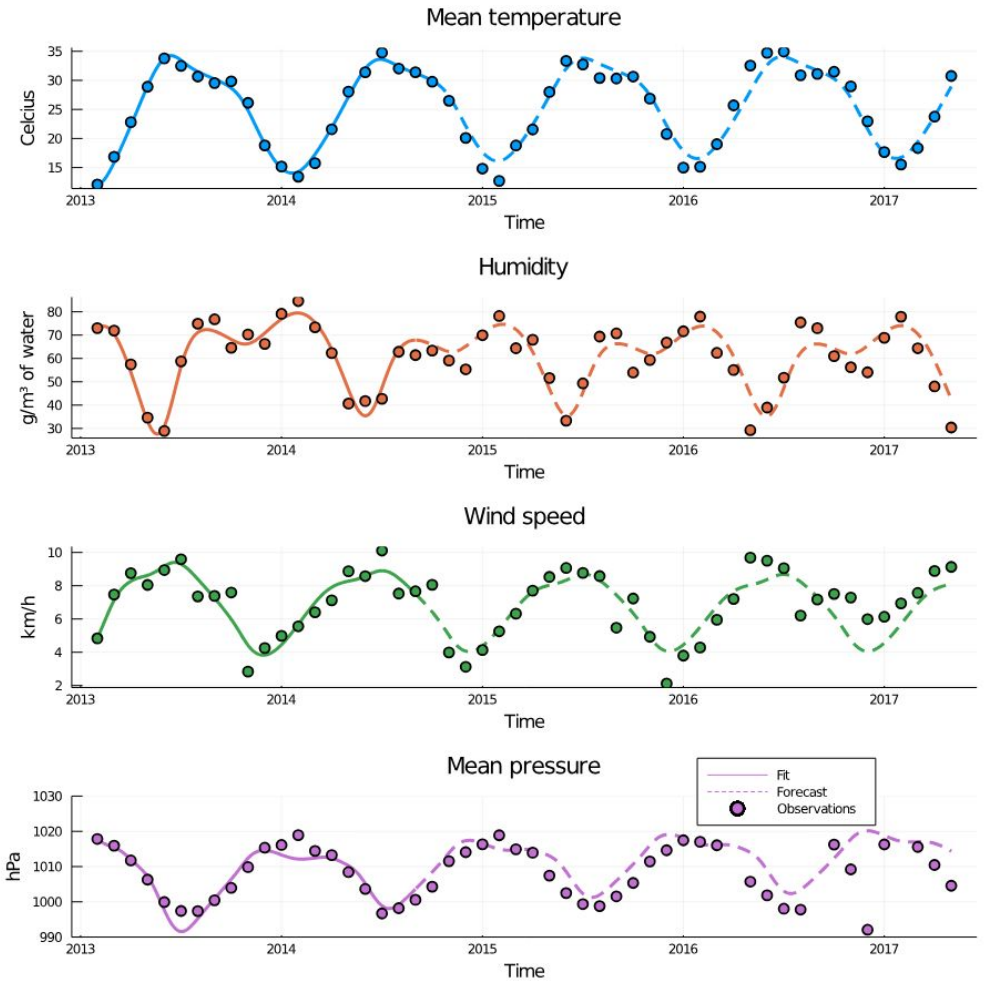



## 8 Stiff ODEs: HIRES Chemical Reaction Network



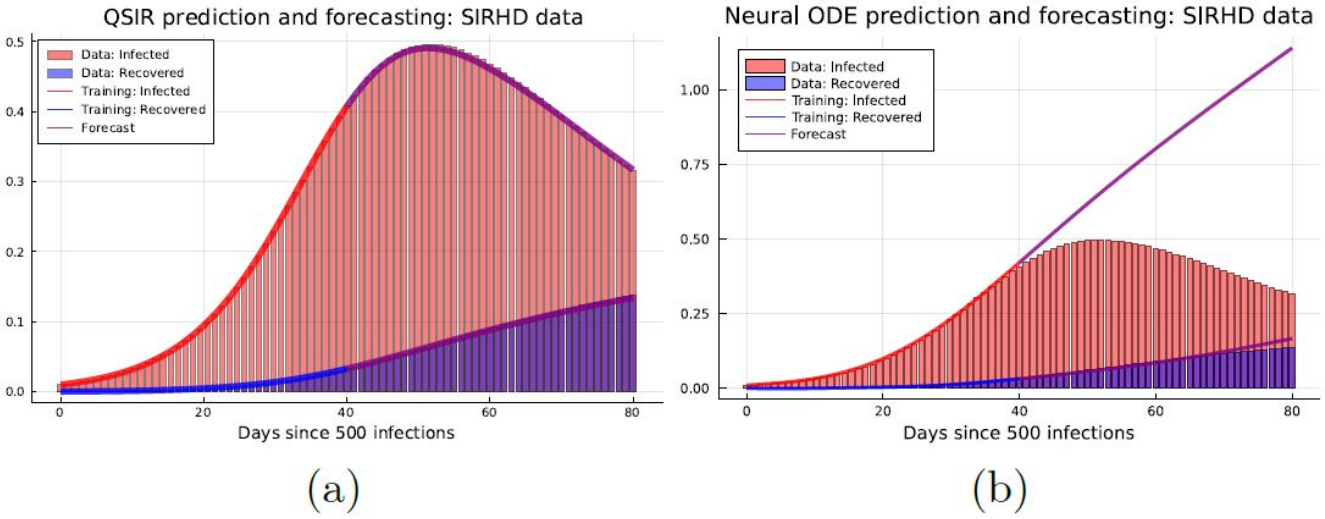
# Implicit NNs are Good at Extrapolation

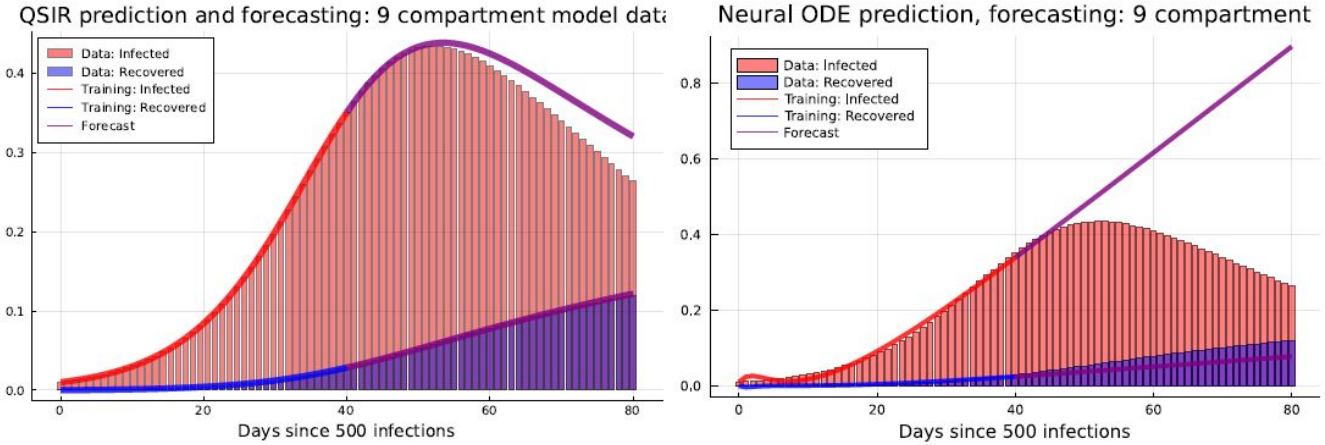


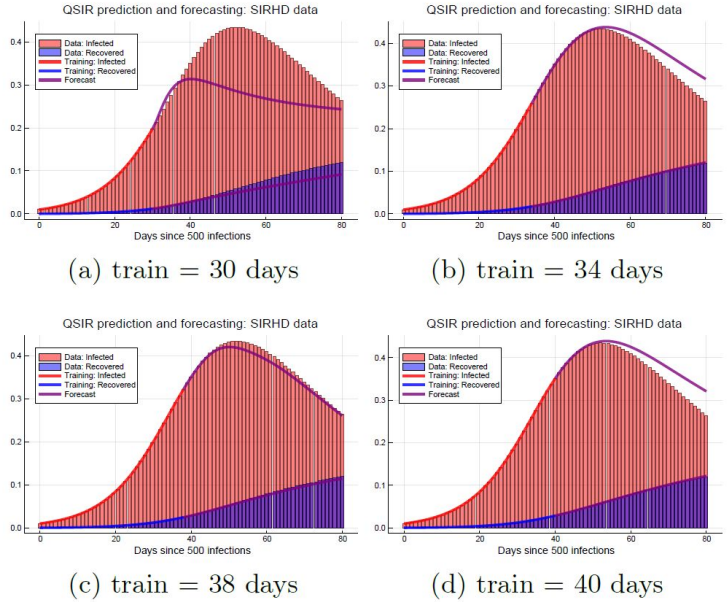




Graphics taken from https://sebastiancallh.github.io/post/neural-ode-weather-forecast/ that was written using our open source softwares Flux.jl, DiffEqFlux.jl, & SciMLSensitivity.jl

# Scientific Machine Learning: Improving Predictions with Less Data





Dandekar, R., Rackauckas, C., & Barbastathis, G. (2020). A machine learning-aided global diagnostic and comparative tool to assess effect of quarantine control in COVID-19 spread. Patterns, 1(9), 100145.

Acquesta, E., Portone, T., Dandekar, R., Rackauckas, C., Bandy, R., & Huerta, J. (2022). *Model-Form Epistemic Uncertainty Quantification for Modeling with Differential Equations: Application to Epidemiology* (No. SAND2022-12823). Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

# The UDE formulation fairly generally allows for imposing prior known structure

# UDEs Effectively Recover Nonlinearities of Epidemic Models

The baseline case:

$$\frac{\mathrm{d}S(t)}{\mathrm{d}t} = -\frac{\tau_{SI}\, S(t)\, I(t)}{N}$$
$$\frac{\mathrm{d}I(t)}{\mathrm{d}t} = \frac{\tau_{SI}\, S(t)\, I(t)}{N} - \tau_{IR} I(t) - \tau_{ID} I(t)$$
$$\frac{\mathrm{d}R(t)}{\mathrm{d}t} = \tau_{IR} I(t)$$
$$\frac{\mathrm{d}D(t)}{\mathrm{d}t} = \tau_{ID} I(t).$$

Replacement of all terms with neural networks:

$$\frac{\mathrm{d}S(t)}{\mathrm{d}t} = -NN_{SI}$$
$$\frac{\mathrm{d}I(t)}{\mathrm{d}t} = NN_{SI} - NN_{IR} - NN_{ID}$$
$$\frac{\mathrm{d}R(t)}{\mathrm{d}t} = NN_{IR}$$
$$\frac{\mathrm{d}D(t)}{\mathrm{d}t} = NN_{ID}$$

**Use SciML knowledge to constrain the interaction graph, but learn the nonlinearities!**

| | Actual Equations | SINDY Active terms | SINDY Equations | Minimum AICC |
|---|---|---|---|---|
| $NN_{SI}$ | 0.85 S I | 1: SI | 0.74 S I | 14 |
| $NN_{IR}$ | 0.1 I | 1: I | 0.097 I | 19 |
| $NN_{ID}$ | 0.05 I | 1: I | 0.049 I | 21 |

Table 4: SIRD: SINDY Recovered terms

# Universal Differential Equations Predict Chemical Processes

$$\frac{\partial c}{\partial t^*} = -\frac{1-\varepsilon}{\varepsilon}\mathrm{ANN}(q, q^*, \theta) - \frac{\partial c}{\partial x^*} + \frac{1}{Pe}\frac{\partial c^2}{\partial x^{*2}},$$

$$\frac{\partial q}{\partial t^*} = \mathrm{ANN}(q, q^*, \theta),$$

$$\frac{\partial c(x^* = 1, \forall t)}{\partial x^*} = 0,$$

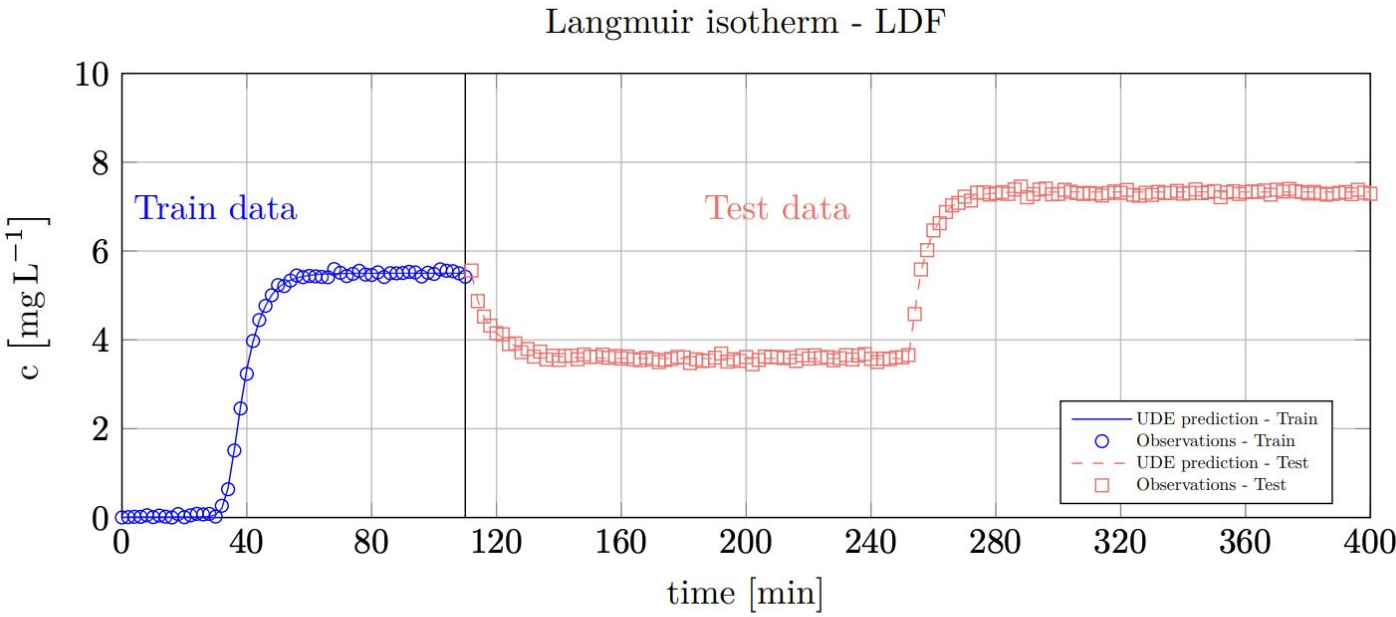
$$\frac{\partial c(x^* = 0, \forall t)}{\partial x^*} = Pe(c - c_{inlet}),$$

$$c(x^* \in (0,1), t^* = 0) = c_0,$$

$$q(x^* \in (0,1), t^* = 0) = q^*(c_0),$$

$$q^* = f(c, p),$$

Santana, V. V., Costa, E., Rebello, C. M., Ribeiro, A. M., Rackauckas, C., & Nogueira, I. B. (2023). Efficient hybrid modeling and sorption model discovery for non-linear advection-diffusion-sorption systems: A systematic scientific machine learning approach. *Chemical Engineering Science*



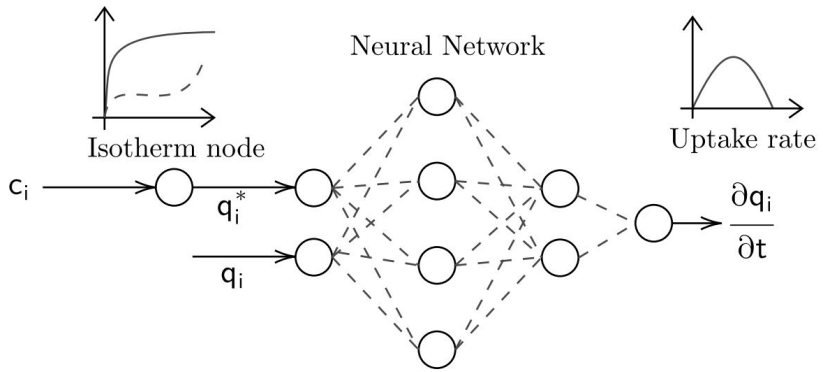




Figure 2: Schematic representation of the proposed hybrid model.

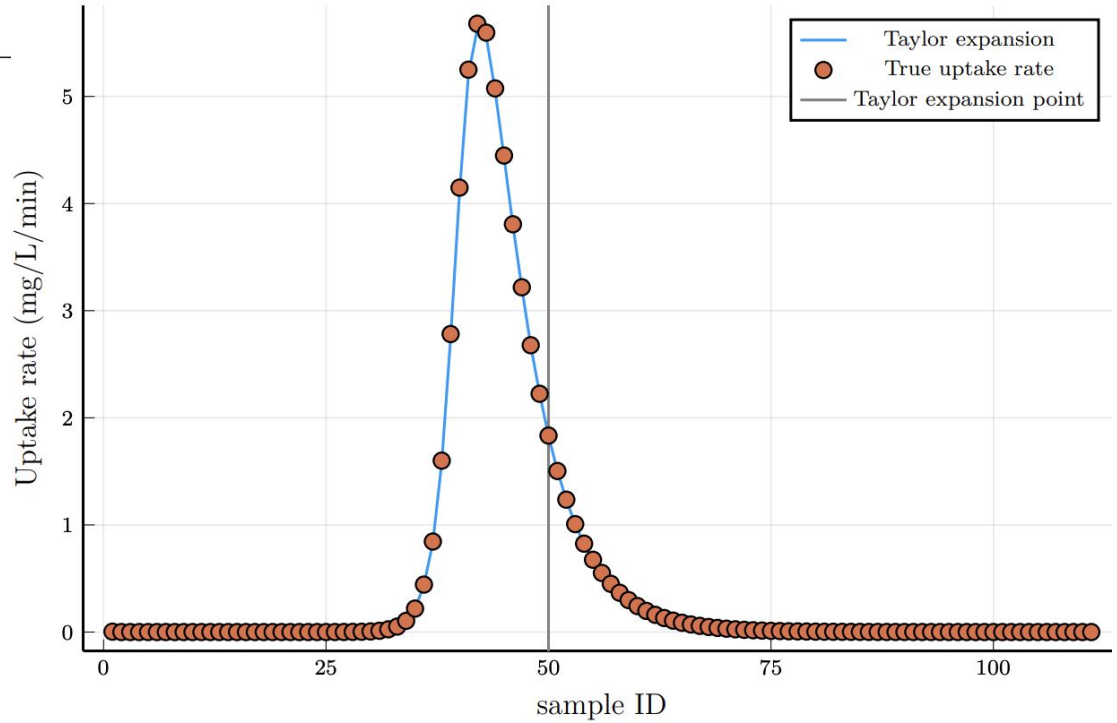
# Universal Differential Equations Predict Chemical Processes

Table 5: Symbolic regression learned polynomials.

| Isotherm | Kinetic | True kinetics | Learned kinetics |
|---|---|---|---|
| Langmuir | LDF | $0.22q^* - 0.22q$ | $-0.535 - 0.225q + 0.234(q^*)$ |
| Langmuir | improved LDF | $0.22(q^* + 0.2789q^* e^{\frac{-q}{2q^*}} - q)$ | $-0.554 - 0.234q + 0.281(q^*)$ |
| Langmuir | Vermeulen's | $0.22\frac{q^{*2}-q^2}{2.0q}$ | $-0.6098 + 0.0122q + 0.263q^*$ $-0.00526qq^*$ |
| Sips | LDF | $0.22q^* - 0.22q$ | $0.198q^* - 0.200q$ |
| Sips | improved LDF | $0.22(q^* + 0.2789q^* e^{\frac{-q}{2q^*}} - q)$ | $0.277q^* - 0.241q$ |
| Sips | Vermeulen's | $0.22\frac{q^{*2}-q^2}{2.0q}$ | $-0.003557q^{*2} - 0.216q + 0.395q^*$ |

$$0.22(q^* + 0.2789q^* e^{\frac{-q}{2q^*}} - q)(49.23, 49.22) \approx 1.834 + 0.275q^* - 0.238q + \mathcal{O}(\|x^2\|)$$

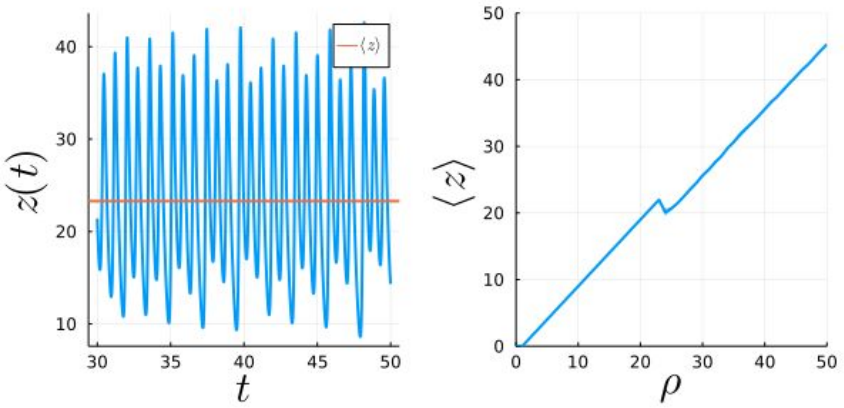
Recovers equations with the same 2nd order Taylor expansion




Santana, V. V., Costa, E., Rebello, C. M., Ribeiro, A. M., Rackauckas, C., & Nogueira, I. B. (2023). Efficient hybrid modeling and sorption model discovery for non-linear advection-diffusion-sorption systems: A systematic scientific machine learning approach. *Chemical Engineering Science*

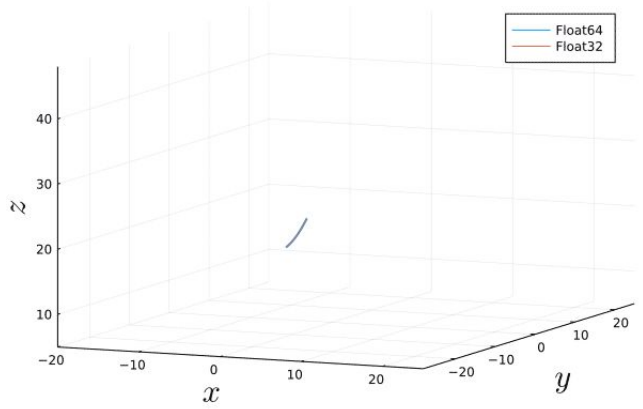
# Can we generalize Differentiable Simulation beyond continuous models?

# Differentiation of Chaotic Systems: Shadow Adjoints




$$\frac{\mathrm{d}}{\mathrm{d}\rho}\langle z\rangle_\infty \neq \lim_{T\to\infty}\frac{\partial}{\partial\rho}\langle z\rangle_T$$

chaotic systems: trajectories diverge to o(1) error … but shadowing lemma guarantees that the solution lies on the attractor




- AD and finite differencing fails!

$$\left.\frac{\mathrm{d}\langle z\rangle_\infty}{\mathrm{d}\rho}\right|_{\rho=28} \approx -49899 \text{ (ForwardDiff)}$$

$$\left.\frac{\mathrm{d}\langle z\rangle_\infty}{\mathrm{d}\rho}\right|_{\rho=28} \approx 472 \text{ (Calculus)}$$

- Shadowing methods in DiffEqSensitivity.jl

$$\left.\frac{\mathrm{d}\langle z\rangle_\infty}{\mathrm{d}\rho}\right|_{\rho=28} \approx 1.028 \text{ (LSS/AdjointLSS)}$$

$$\left.\frac{\mathrm{d}\langle z\rangle_\infty}{\mathrm{d}\rho}\right|_{\rho=28} \approx 0.997 \text{ (NILSS)}$$

https://frankschae.github.io/post/shadowing/

# Differentiation of Chaotic Systems: Shadow Adjoints

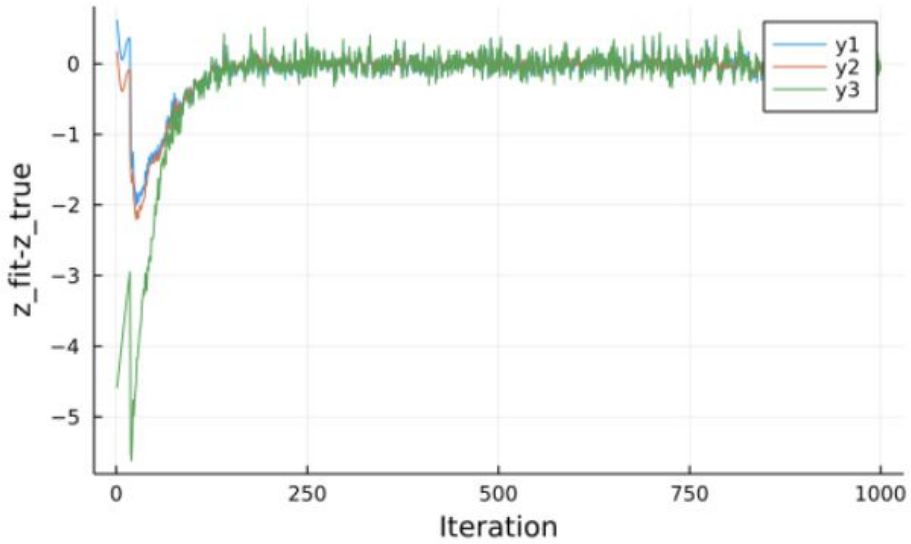



Figure 18: Summary statistic fitting corresponding to Figure 17. Fitting with all 3 Lorenz parameters free using 3 summary statistics ($< |x| >, < |y| >, < |z| >$). ADAM optimizer learning rate is 1e-1.

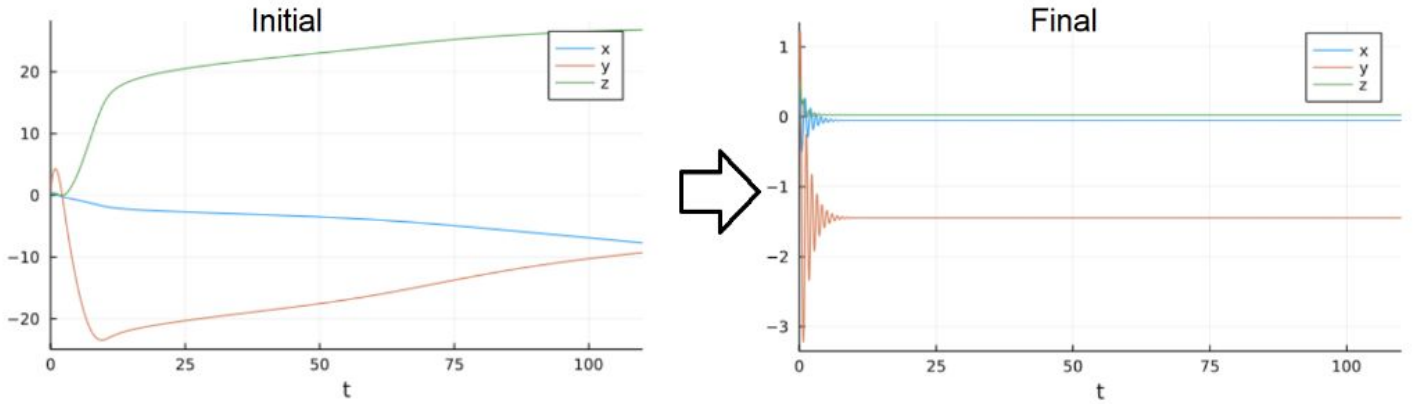
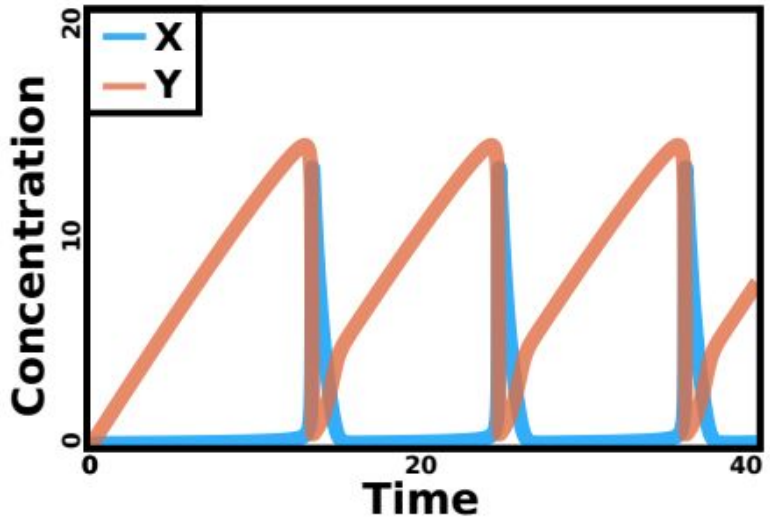



Figure 7: UODE fit with ADAM optimizer 1e-1 and re-initialized ADAM(1e-2) every 50 iterations. Summary statistics are all combinations of 2 summary statistics and standard deviations of each parameter xyz. Left is before and right is after training.

# Poisson Jump Equations: Stochastic Chemical Reactions

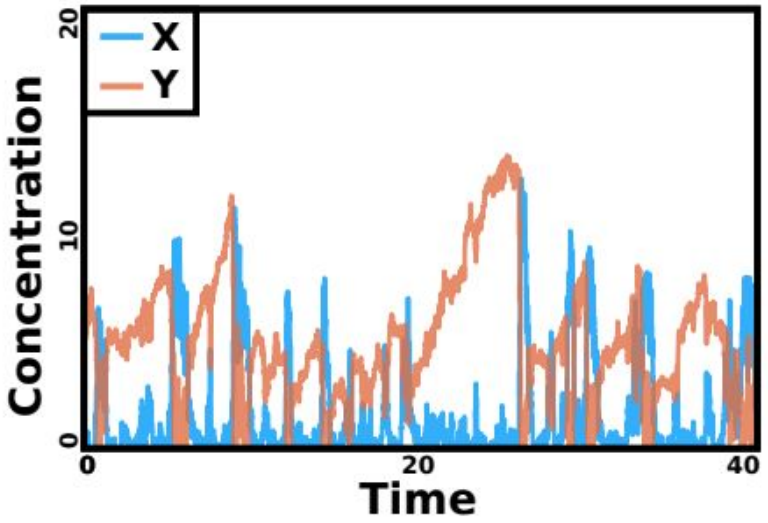
**A**

```
using OrdinaryDiffEq
u0 = [:X => 0., :Y => 0.]
tspan = (0., 60.)
p = [:A => 1.0, :B => 4.0]
oprob = ODEProblem(brusselator,u0,tspan,p)
sol = solve(oprob,Rosenbrock23())
plot(sol)
```



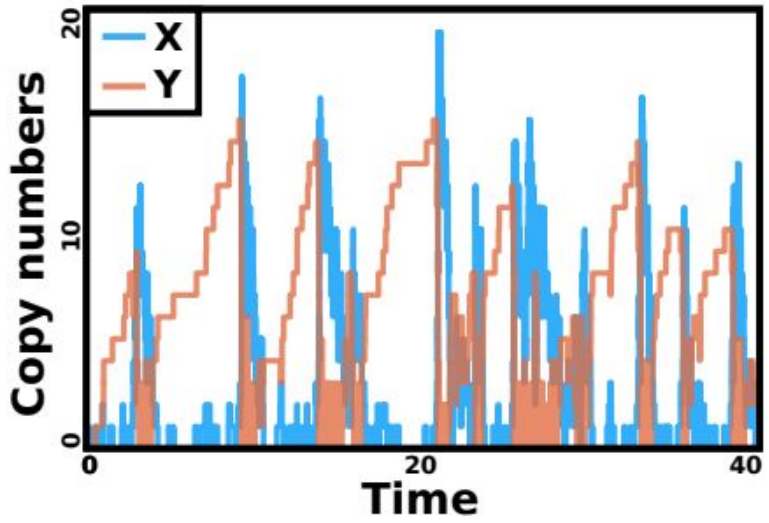

**B**

```
using StochasticDiffEq
u0 = [:X => 0., :Y => 5.]
tspan = (0., 60.)
p = [:A => 1.0, :B => 4.0]
sprob = SDEProblem(brusselator,u0,tspan,p)
sol = solve(sprob,ImplicitEM())
plot(sol)
```



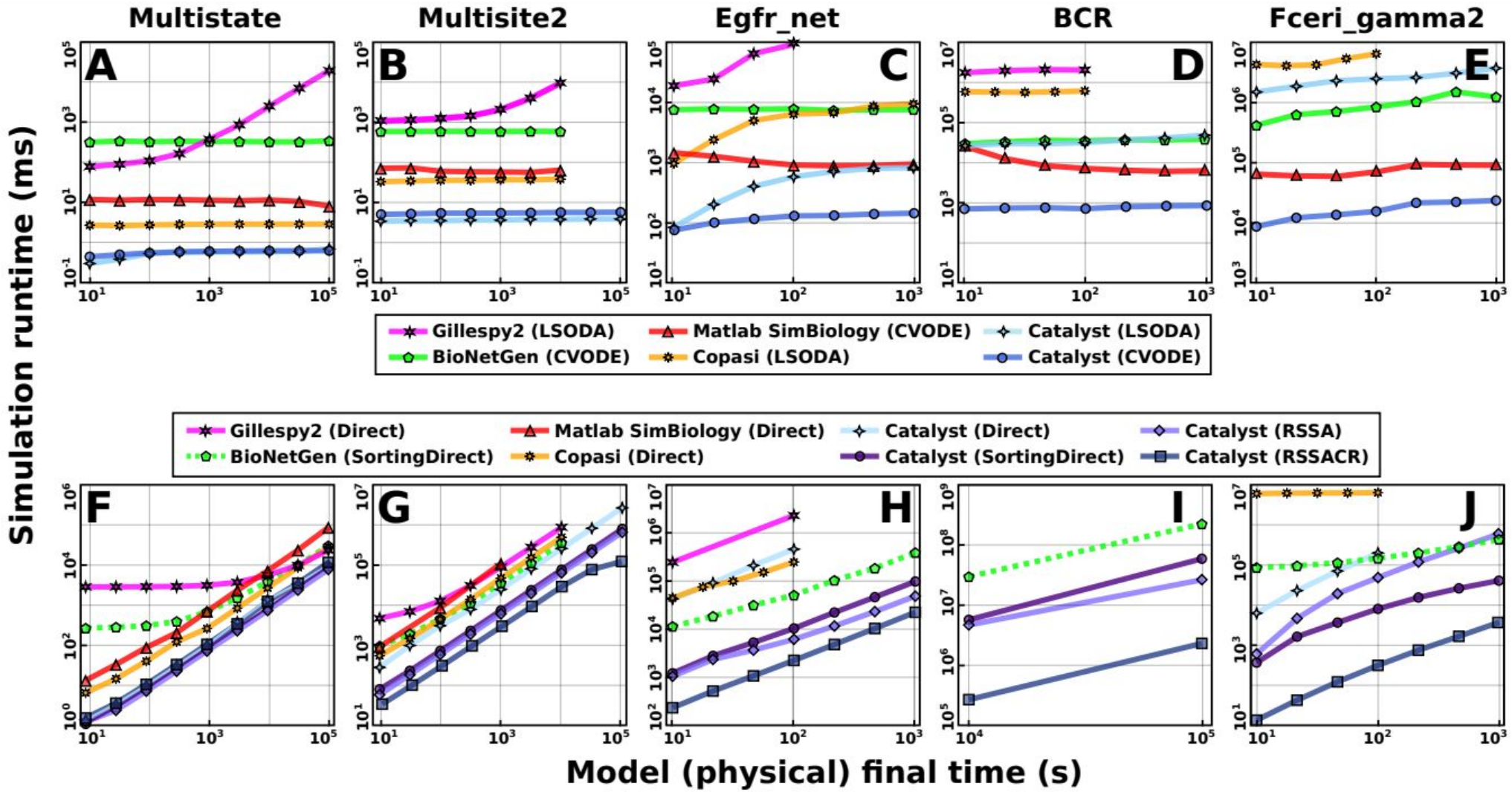

**C**

```
using DiffEqJump
u0 = [:X => 0, :Y => 0]
tspan = (0., 60.)
p = [:A => 3.0, :B => 4.0]
dprob = DiscreteProblem(brusselator,u0,tspan,p)
jprob = JumpProblem(brusselator,dprob,Direct())
sol = solve(jprob,SSAStepper())
plot(sol)
```




Loman, Torkel, Yingbo Ma, Vasily Ilin, Shashi Gowda, Niklas Korsbo, Nikhil Yewale, Christopher Vincent Rackauckas, and Samuel A. Isaacson. "Catalyst: fast biochemical modeling with Julia." *bioRxiv* (2022).

# ~100x performance improvements in Gillespie SSAs with Catalyst.jl



# New Automatic Differentiation of Discrete Stochastic Spaces?

**Traditional AD**

$$f(p)$$

$$\frac{\mathrm{d}f(p)}{\mathrm{d}p}$$

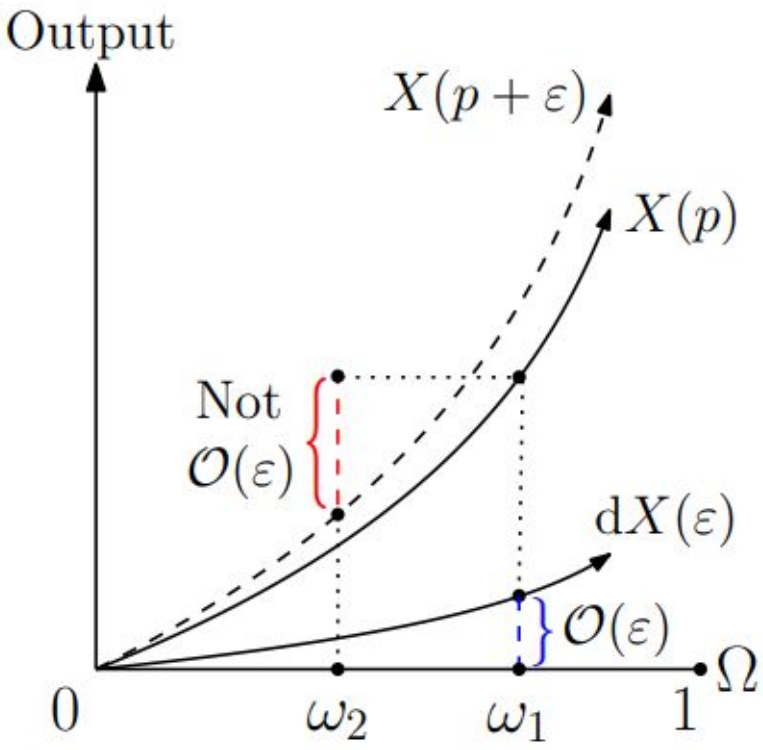
**Discrete Stochastic AD**

$$X(p)$$

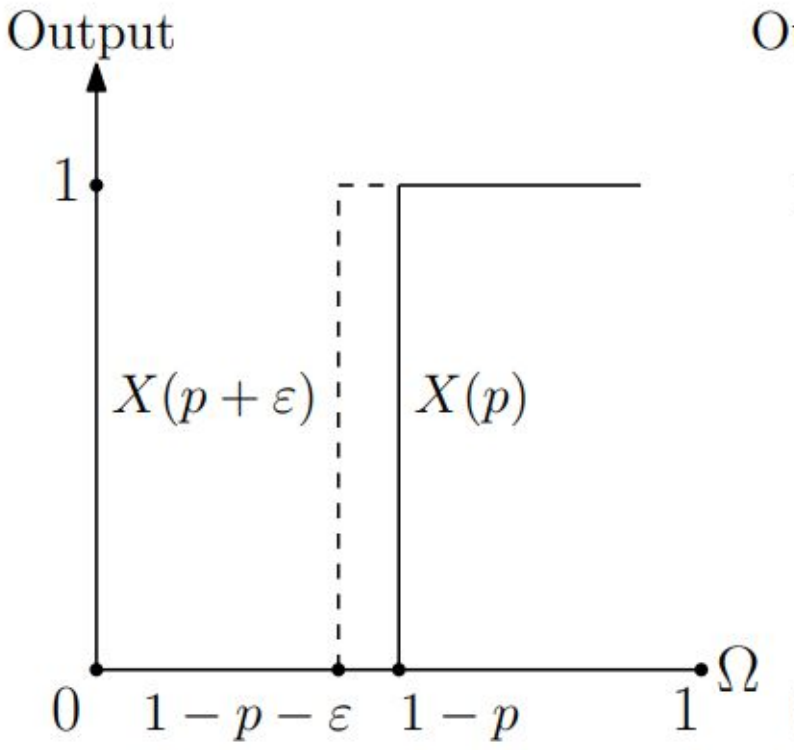
$$\mathbb{E}[\tilde{X}(p)] = \frac{\mathrm{d}\mathbb{E}[X(p)]}{\mathrm{d}p}$$

**Automatic Differentiation of Programs with Discrete Randomness**
**Accepted to NeurIPS 2022!**

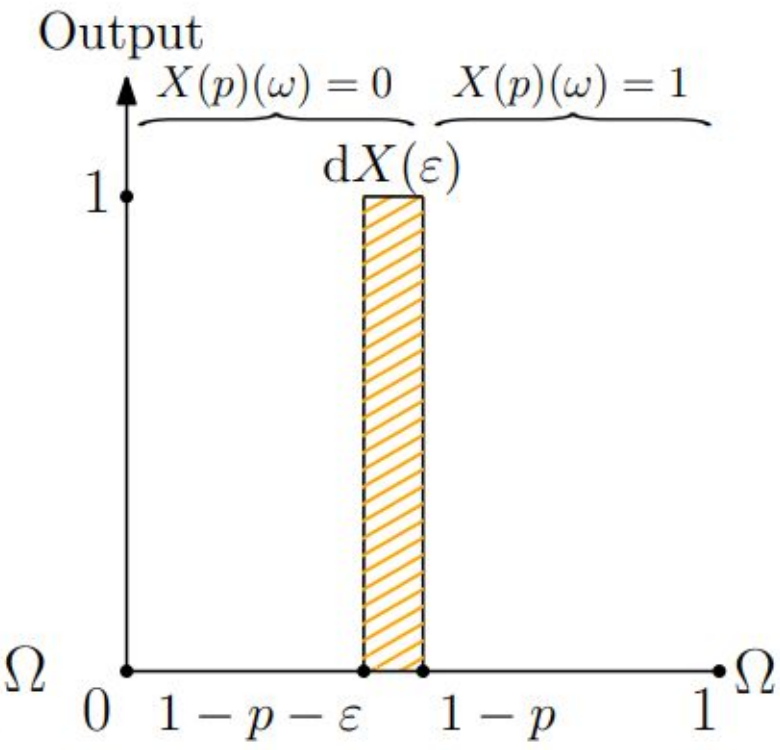
# Infinitesimal Changes -> O(1) Changes




(a) $X(p) \sim \mathrm{Exp}(p)$.

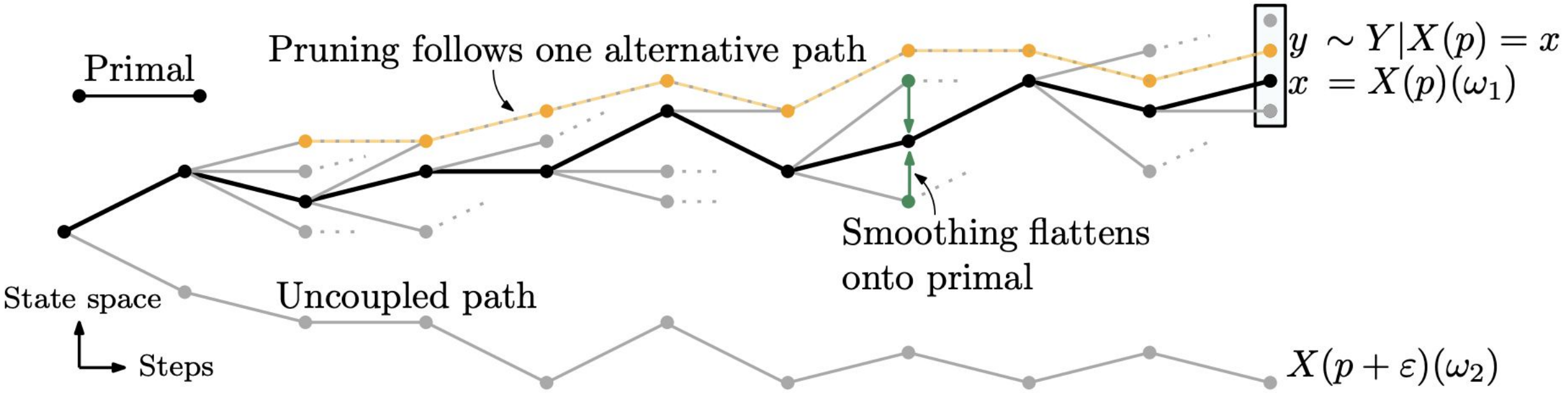



(b) $X(p) \sim \mathrm{Ber}(p)$.




(c) $X(p) \sim \mathrm{Ber}(p)$.

# O(1) changes with infinitesimal probability



# StochasticAD.jl: Unbiased Stochastic Derivatives with Linear Cost

**Perform AD on Agent-Based models, particle filters, chemical reactions, and more!**

```
struct StochasticTriple
    value # primal evaluation
    δ # "infinitesimal" component
    Δs # component of discrete change
       # with "infinitesimal"
       # probability
end
```

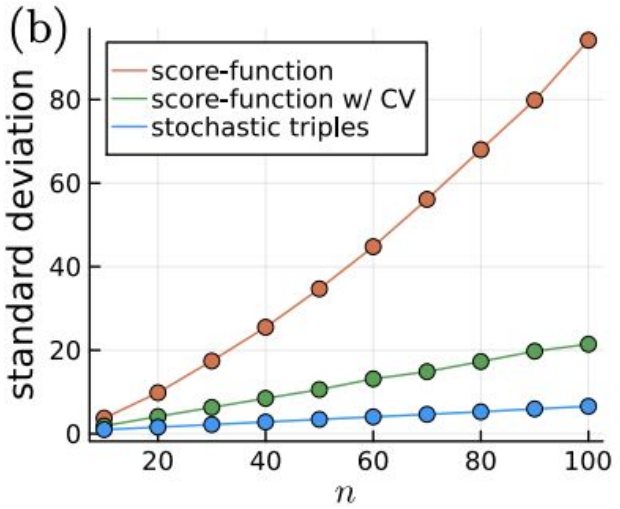
```
using Distributions
function X(p)  p = 0.6 + ε
    a = p^2  0.36 + 1.2ε
    b = rand(Binomial(10, p))
     6 + (1 with probability 10.0ε)
    c = 2 * b + 3 * rand(Bernoulli(p))
     12 + (3 with probability 12.5ε)
    return a * c * rand(Normal(p, p^2))
end
```

```
julia> using StochasticAD
julia> st = stochastic_triple(X, 0.6) # sample a single stochastic triple at p = 0.6
 2.16 + 10.05ε + (0.64 with probability 12.5ε)
julia> derivative_contribution(st) # which produces a single derivative estimate...
 16.79
julia> samples = [derivative_estimate(X, 0.6) for i in 1:1000] # take many estimates!
julia> println("d/dp of E[X(p)]: $(mean(samples)) ± $(std(samples) / sqrt(1000))")
d/dp of E[X(p)]: 19.39 ± 0.48
```

# StochasticAD.jl on Agent-Based Models

(a)

```
...
X = 0
for step in 1:n
  i = rand(Categorical(probs(X)))
  X += steps[i]
end
return f(X)
```



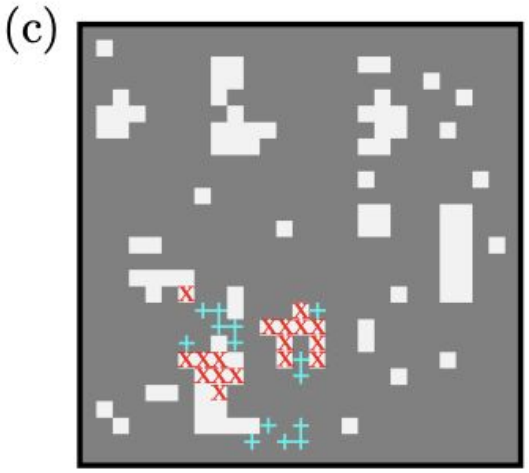



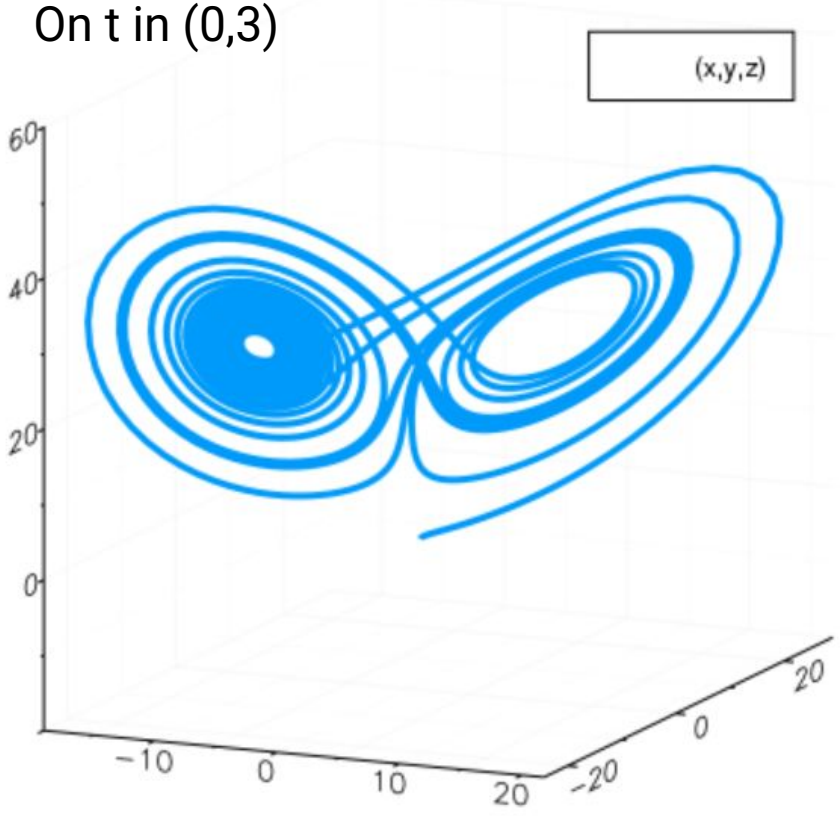

Figure 4: Automatic differentiation of discrete-time Markov processes. **(a)** Code snipped for a 1D random walk, which can be automatically differentiated by `StochasticAD.jl`; `probs(X)` gives the transition probabilities at value `X` and `steps[i]` gives the step size for the `i`th transition. **(b)** The variance of unbiased gradient estimates of the random walk program using stochastic triples and the score function, which is applied both without a control variate (CV) and with a pre-computed batch-average CV. **(c)** The final board for one run of the stochastic Game of Life with $N = 25$ and $T = 10$, where the "+" signs represent additional living cells (white) in the stochastic alternative path, and the "X" signs represent additional dead cells (grey).

**But wait, why not PINNs?**

**How does differentiable simulation compare to PINNs?**

# Keeping Neural Networks Small Keeps Speed For Inverse Problems

Problem: parameter estimation
of Lorenz equation from data
On t in (0,3)




## DeepXDE (TensorFlow Physics-Informed NN)

```
Best model at step 57000:
  train loss: 5.91e-03
  test loss: 5.86e-03
  test metric: []

'train' took 362.351454 s
```

## DiffEqFlux.jl (Julia UDEs)

```
opt = Opt(:LN_BOBYQA, 3)
lower_bounds!(opt,[9.0,20.0,2.0])
upper_bounds!(opt,[11.0,30.0,3.0])
min_objective!(opt, obj_short.cost_function2)
xtol_rel!(opt,1e-12)
maxeval!(opt, 10000)
@time (minf,minx,ret) = NLopt.optimize(opt,LocIniPar) # 0.1 seconds
```

```
0.032699 seconds (148.87 k allocations: 14.175 MiB)
(2.7636309213683456e-18, [10.0, 28.0, 2.66], :XTOL_REACHED)
```

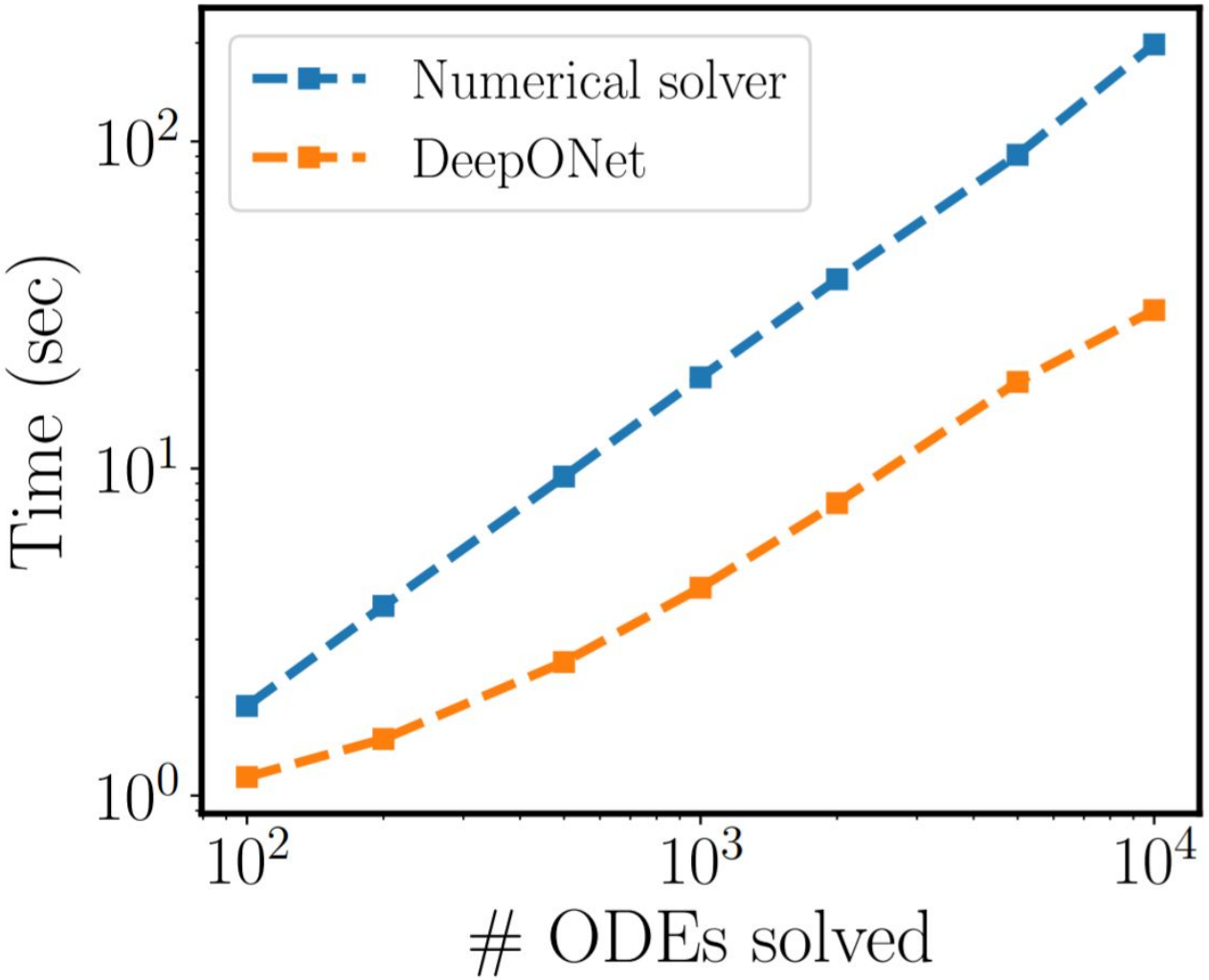
# Note on Neural Networks “Outperforming” Classical Solvers

## Long-time integration of parametric evolution equations with physics-informed DeepONets

Sifan Wang, Paris Perdikaris

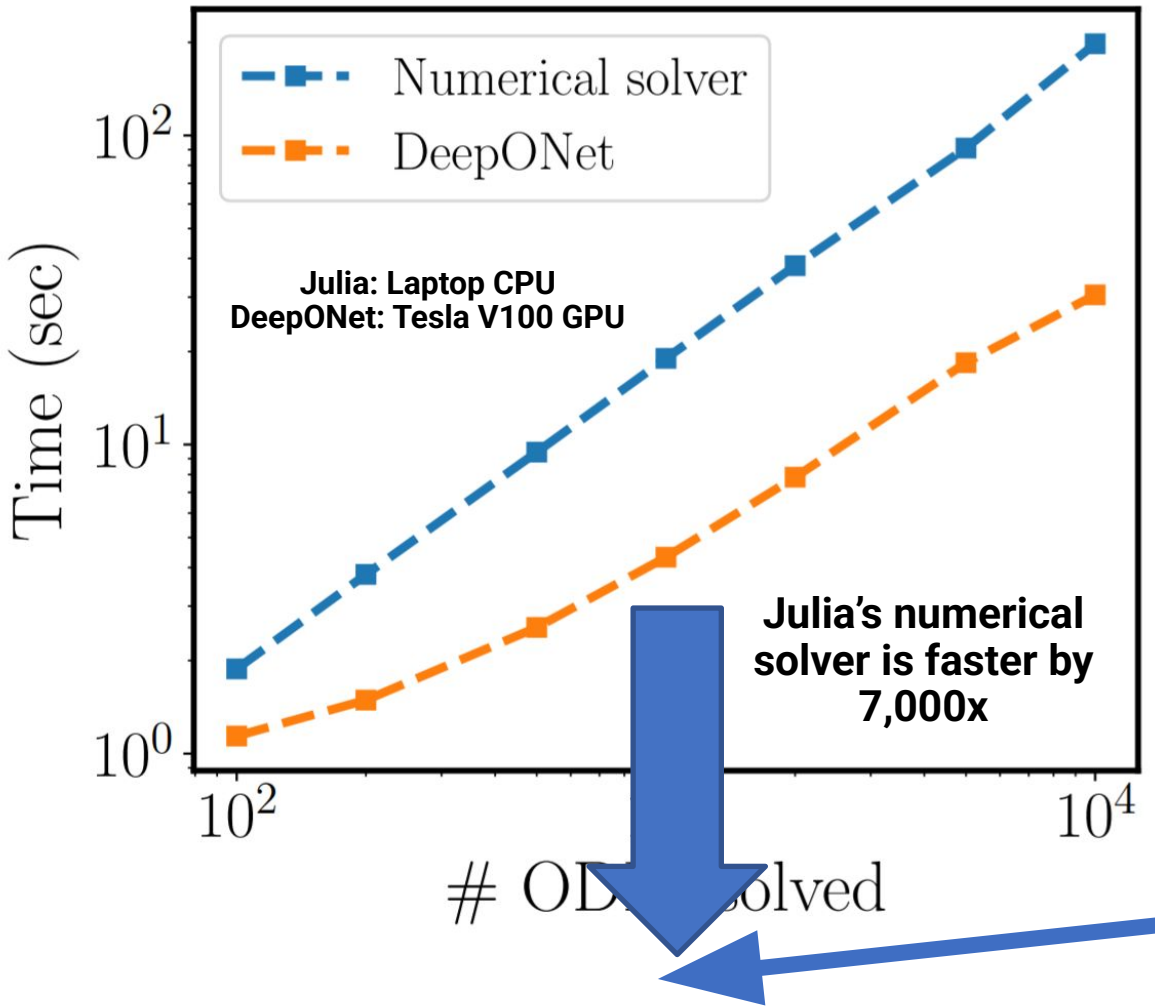
Ordinary and partial differential equations (ODEs/PDEs) play a paramount role in analyzing and simulating complex dynamic processes across all corners of science and engineering. In recent years machine learning tools are aspiring to introduce new effective ways of simulating PDEs, however existing approaches are not able to reliably return stable and accurate predictions across long temporal horizons. We aim to address this challenge by introducing an effective framework for learning infinite-dimensional operators that map random initial conditions to associated PDE solutions within a short time interval. Such latent operators can be parametrized by deep neural networks that are trained in an entirely self-supervised manner without requiring any paired input-output observations. Global long-time predictions across a range of initial conditions can be then obtained by iteratively evaluating the trained model using each prediction as the initial condition for the next evaluation step. This introduces a new approach to temporal domain decomposition that is shown to be effective in performing accurate long-time simulations for a wide range of parametric ODE and PDE systems, from wave propagation, to reaction-diffusion dynamics and stiff chemical kinetics, all at a fraction of the computational cost needed by classical numerical solvers.

# Note on Neural Networks “Outperforming” Classical Solvers



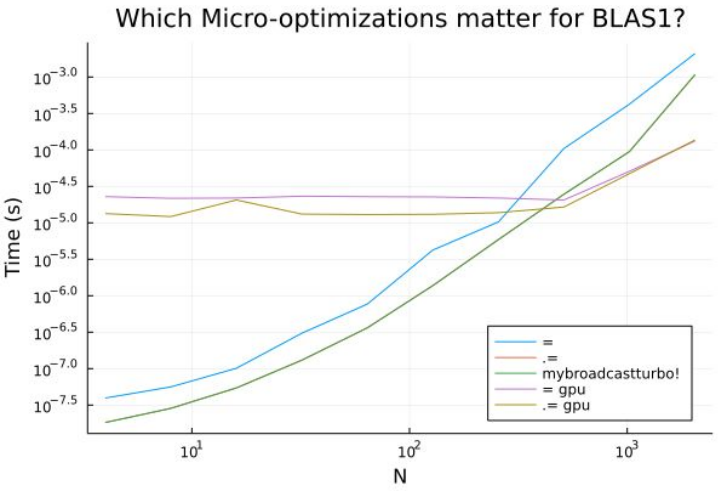

**Oh no, we’re doomed!**

# Stay tuned…




```julia
using ModelingToolkit, OrdinaryDiffEq, StaticArrays

@variables t y₁(t) y₂(t) y₃(t)
@parameters k₁ k₂ k₃
D = Differential(t)
eqs = [D(y₁) ~ -k₁*y₁+k₃*y₂*y₃
       D(y₂) ~  k₁*y₁-k₂*y₂^2-k₃*y₂*y₃
       D(y₃) ~  k₂*y₂^2]

sys = ODESystem(eqs, t)
prob = ODEProblem{false}(sys,SA[y₁=>1f0,y₂=>0f0,y₃=>0f0],(0f0,500f0),
                                SA[k₁=>4f-2,k₂=>3f7,k₃=>1f4],jac=true)

N = 1000
y₁s = rand(Float32,N)
y₂s = 1f-4 .* rand(Float32,N)
y₃s = rand(Float32,N)

function prob_func(prob,i,repeat)
    remake(prob,p=SA[y₁s[i],y₂s[i],y₃s[i]])
end

monteprob = EnsembleProblem(prob, prob_func = prob_func, safetycopy=false)
solve(monteprob,Rodas5(),EnsembleThreads(),trajectories=1000)

@time solve(monteprob,Rodas5(),EnsembleThreads(),trajectories=1000)

#0.006486 seconds (172.26 k allocations: 16.740 MiB)
#0.006024 seconds (172.26 k allocations: 16.740 MiB)
#0.007074 seconds (172.26 k allocations: 16.740 MiB)
```

# Code Optimization in Machine Learning vs Scientific Computing

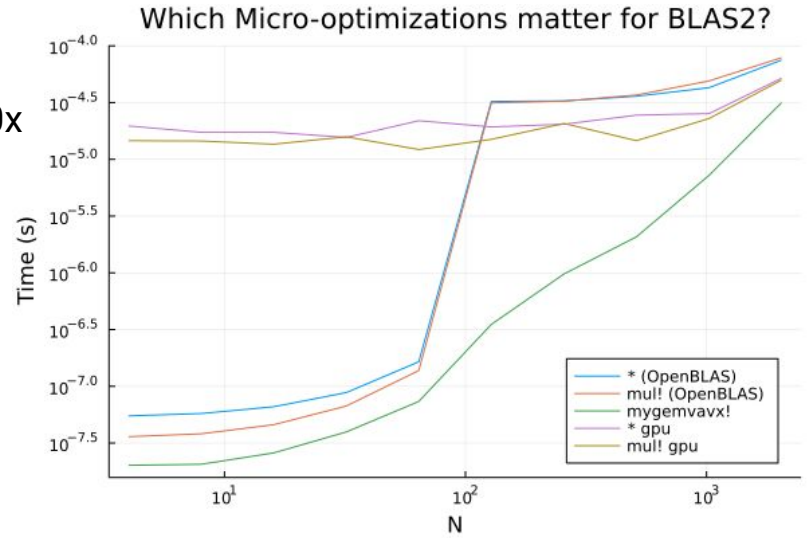


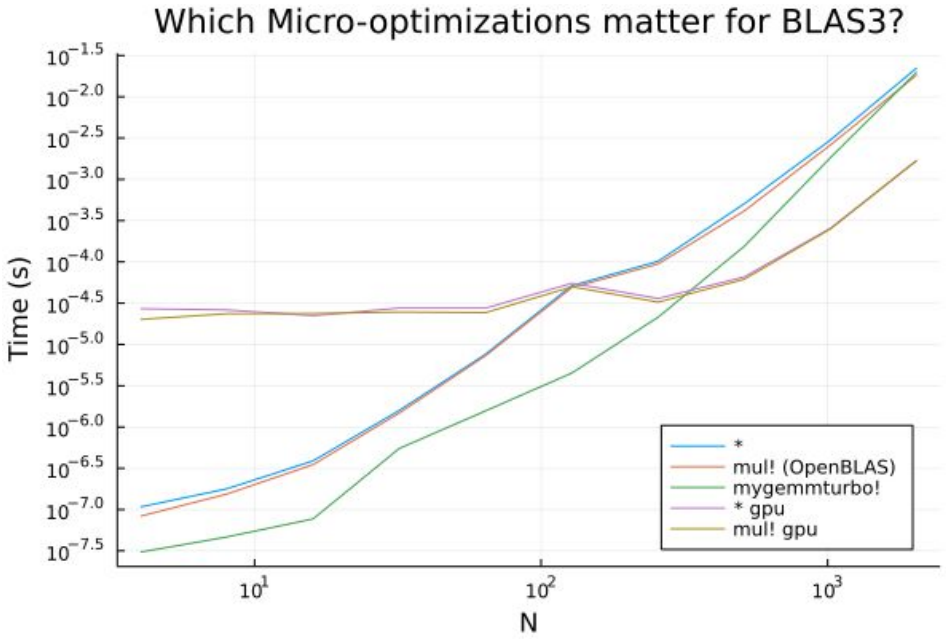

Scientific codes
O(n) and O(n^2) operations

Mutation and
Memory management: 10x

Manual SIMD: 5x

…




Big O(n^3) operations?
Just use a GPU
Don’t worry about overhead
You’re fine!

**Simplest code is ~3x from optimized**

# SimpleChains + StaticArray Neural ODEs

```
sc = SimpleChain(
                static(2),
                Activation(x -> x.^3),
                TurboDense{true}(tanh, static(50)),
                TurboDense{true}(identity, static(2))
            )

p_nn = SimpleChains.init_params(sc)

f(u,p,t) = sc(u,p)
```

his function is plugged into an ODE solver and the L2 loss is calculated from
ıe numerical solution and the NeuralODE output.

```
prob_nn = ODEProblem(f, u0, tspan)

function predict_neuralode(p)
    Array(solve(prob_nn,
Tsit5();p=p,saveat=tsteps,sensealg=QuadratureAdjoint(autojacvec=ZygoteVJP())))
end
```

**About a 5x improvement**

**~1000x in a nonlinear mixed effects context**

**Caveat: Requires sufficiently small ODEs (<20)**

**And simulation processes are still improving rapidly.**

# New Parallelized Extrapolation Methods for Stiff ODEs: 2x-4x improvements

Elrod, Chris, Yingbo Ma, and Christopher Rackauckas. "Parallelizing Explicit and Implicit Extrapolation Methods for Ordinary Differential Equations." *arXiv preprint arXiv:2207.08135* (2022).

Accepted to HPEC 2022!

**Caveat: only good for 5-200 ODEs with no implicit parallelism in f**

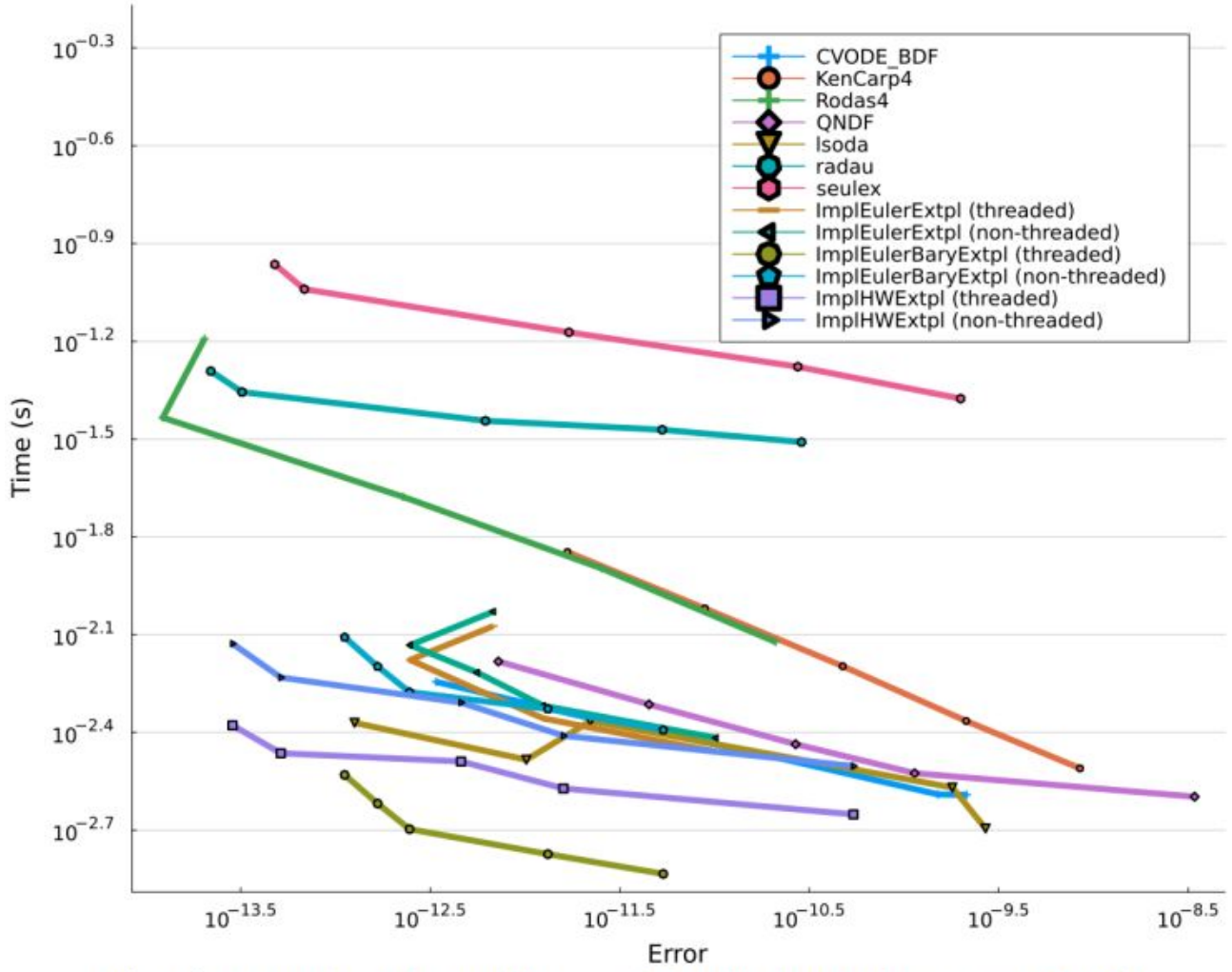


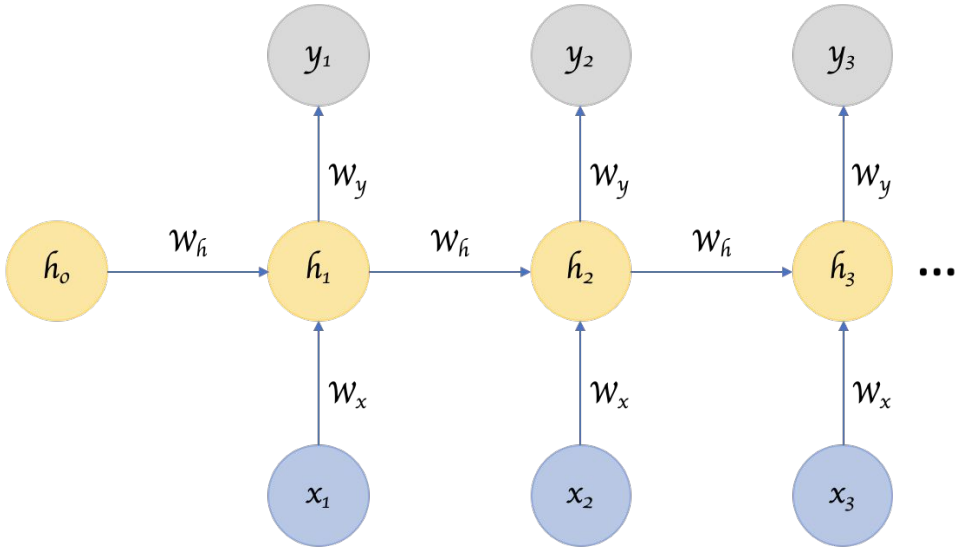

Fig. 6. Benchmark on QSP model with low tolerances [VI-B6].

# Can we improve “standard machine learning” using these new differentiable tools?

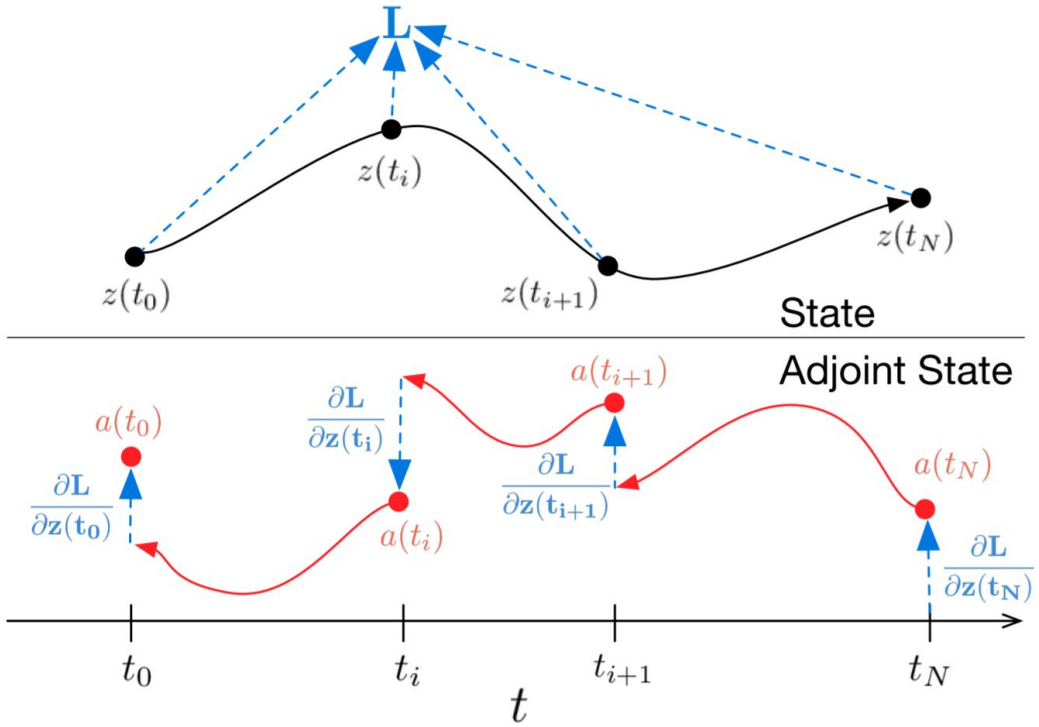
# Using Neural ODEs as automated hyperparameter optimization

# Neural ODEs as Adaptive Layer Methods

**RNN:** Euler's method with fixed number of layers



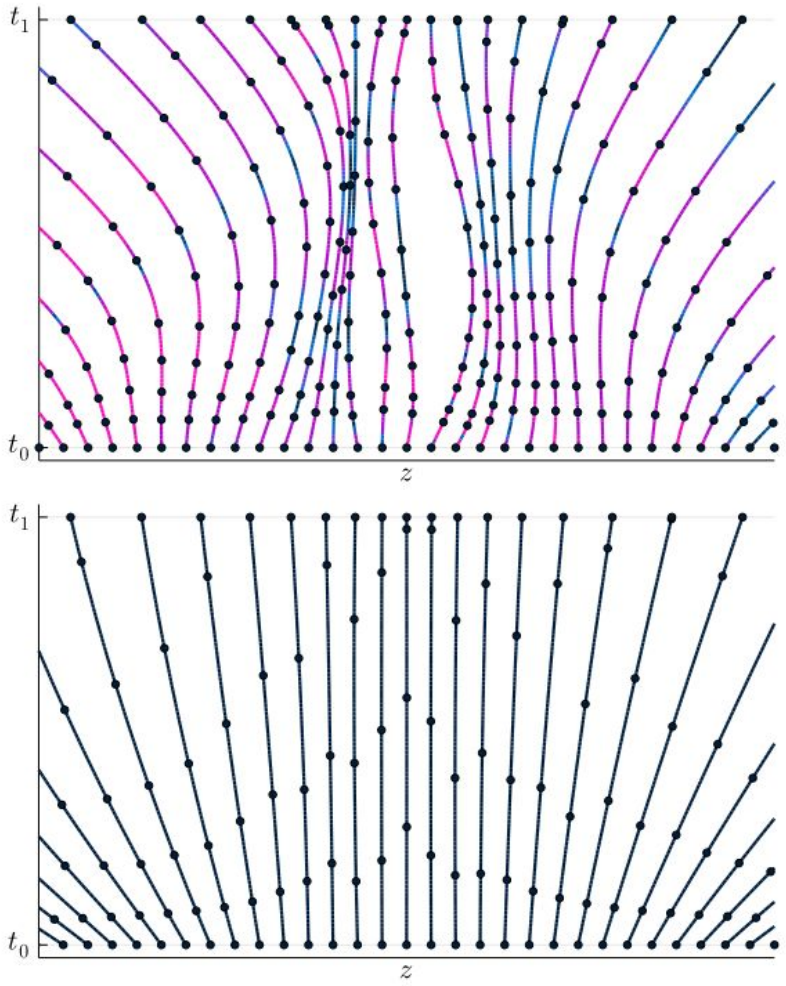

**Neural ODE:** Adaptive ODE solver chooses number of layers




Neural ODEs can be used on classical machine learning problems to automatically learn the required number of layers
ML Layer: Value is initial condition of ODE, output is solution of ODE at final time

Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018, December). Neural ordinary differential equations. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (pp. 6572-6583).

# If you only care about the end, why not learn the easiest dynamics you can?




Adaptive ODE solvers are correct to $K-1$ derivatives and control error on the $K$th.

Use Taylor mode (high order) automatic differentiation to calculate:

$$R_K(\theta) = \int_{t_0}^{t_1} \left\| \frac{d^K z(t)}{dt^K} \right\| dt$$

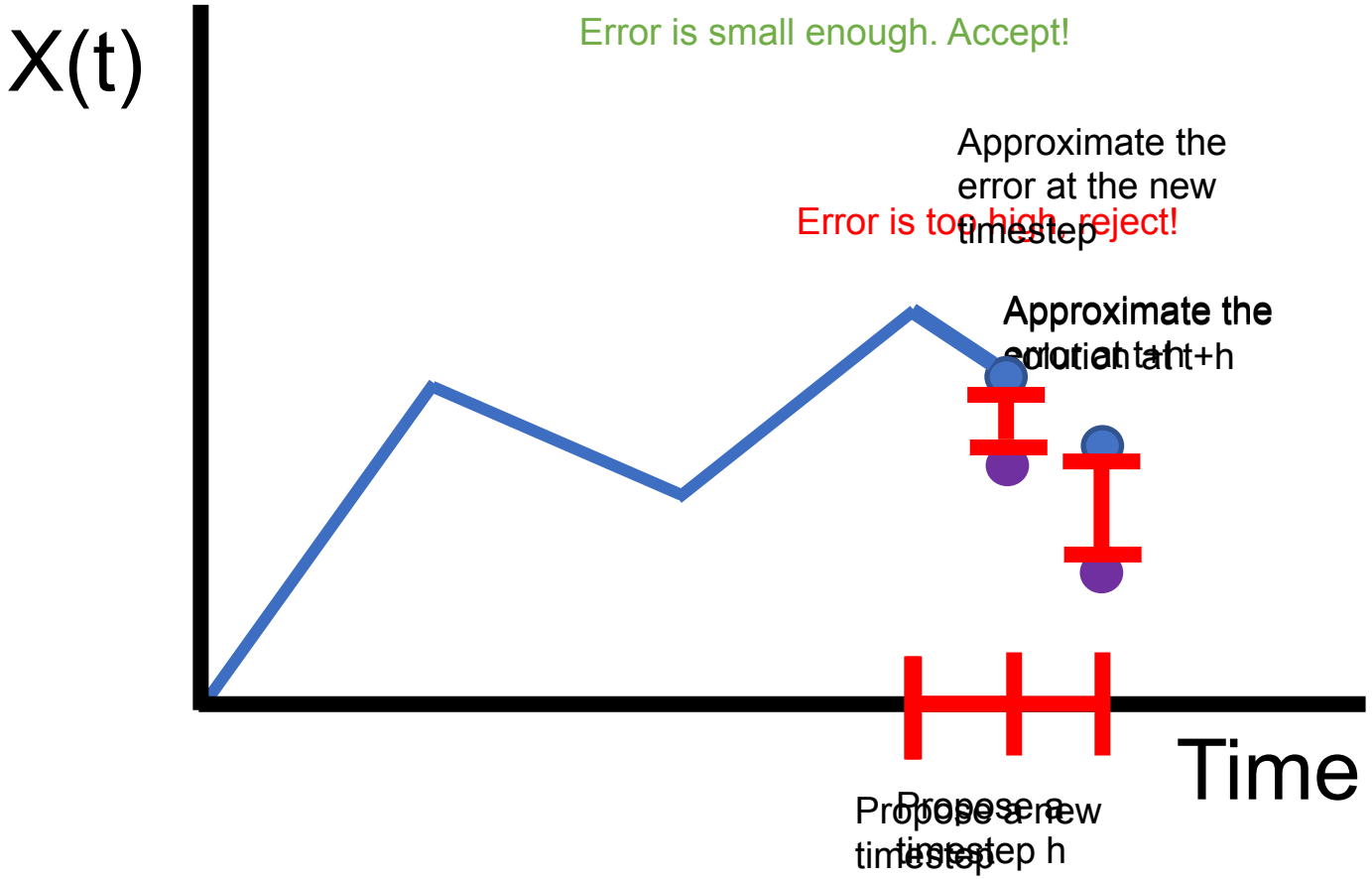
and regularize:

$$L_{reg}(\theta) = L(\theta) + R_K(\theta)$$

**It does accelerate the learned dynamics, but training is expensive (10x slower!) because higher order automatic differentiation is exponentially expensive.**

Kelly, J., Bettencourt, J., Johnson, M.J. and Duvenaud, D., 2020. Learning differential equations that are easy to solve. arXiv preprint arXiv:2007.04504.

# But Solvers “know” a lot about the equation!



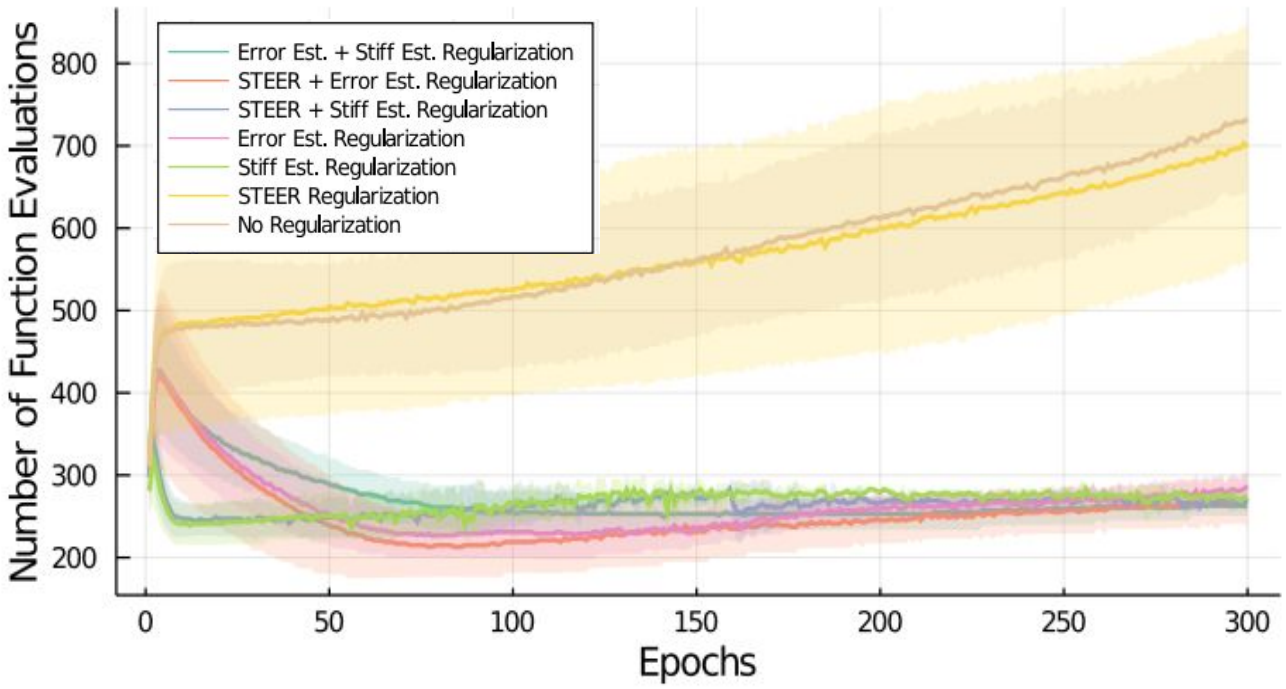

Adaptive ODE solvers already know “free” error estimates and stiffness estimators!

**Idea: use the solver’s internal heuristics to regularize out “hard” dynamics**

# How to improve by an order of magnitude: use knowledge of numerical methods!

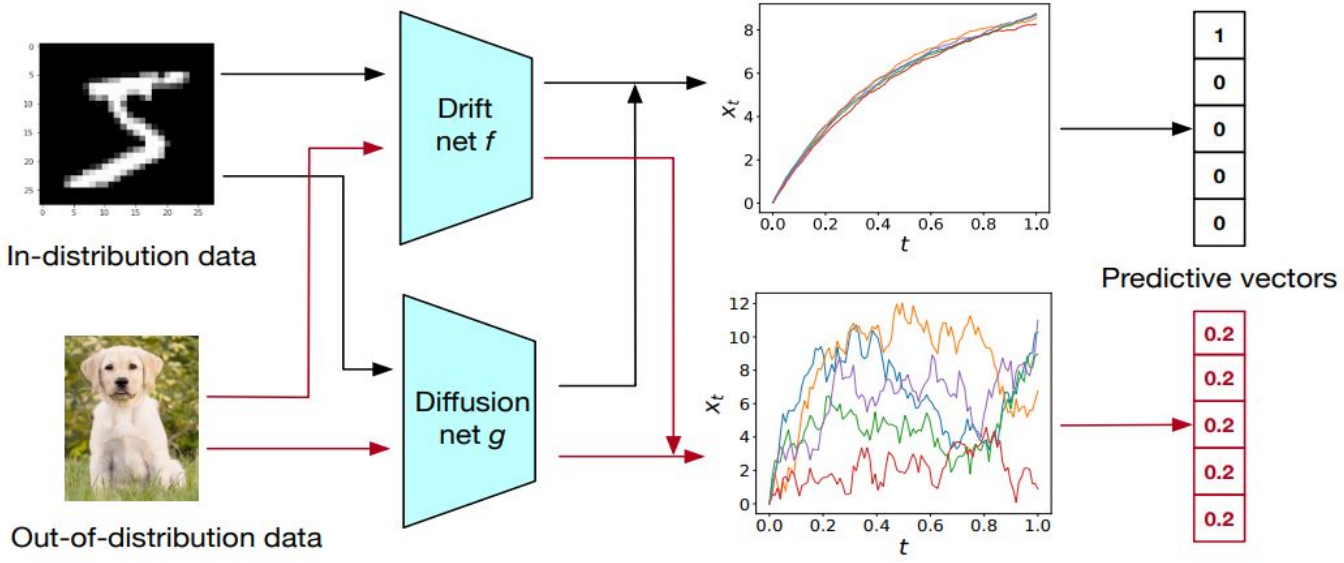
## 10x Neural ODE training vs previous regularization, 2x faster prediction time vs vanilla neural ODE

| Method | Train Loss | Test Loss | Train Time (hr) | Prediction Time (s) |
|---|---|---|---|---|
| Vanilla NODE | 3.48 | 3.55 | 1.75 | 0.53 |
| TayNODE | 4.21 | 4.21 | 12.3 | 0.22 |
| **SRNODE** | **3.52** | **3.58** | **0.87** | **0.20** |

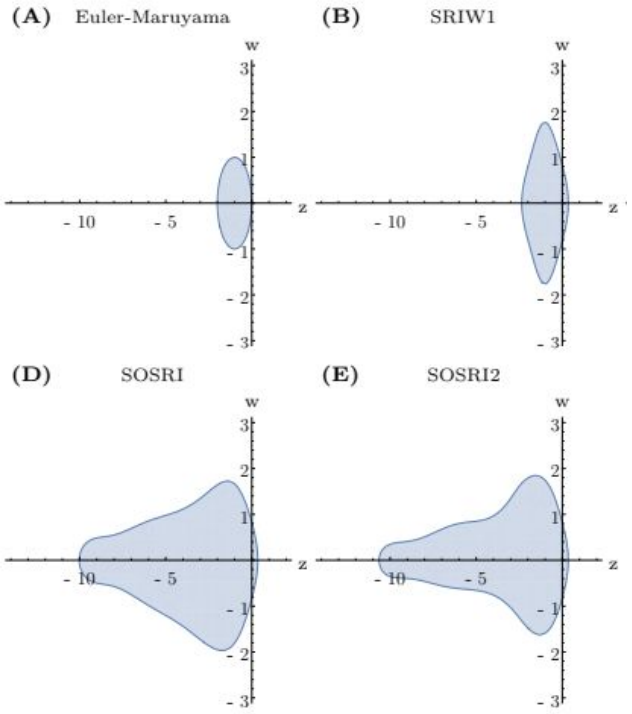


# Neural SDEs improve generalization. Can we improve them?

$$dX = f(X_t, t)dt + g(X_t, t)dW_t$$




**Add noise and uncertainty quantification to continuous layer methods via stochastic differential equations**




**New improved stability SDE solvers with adaptivity and automatic stiffness detection**

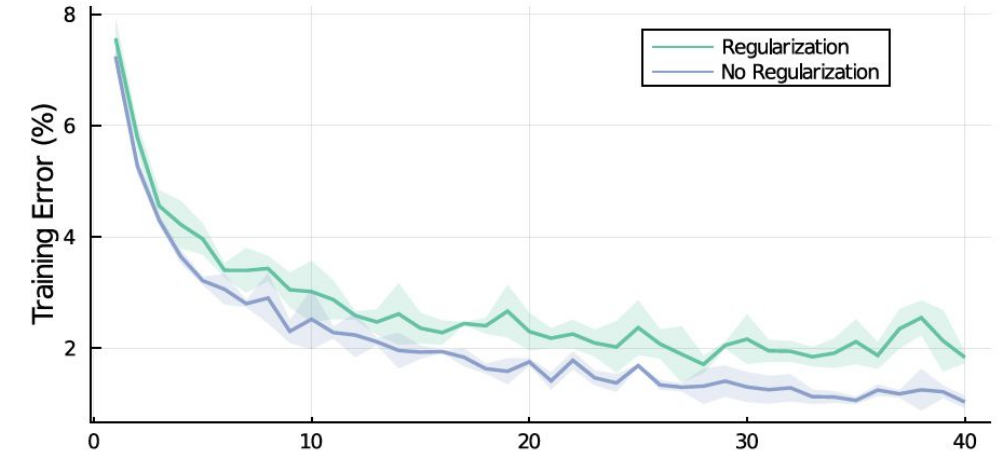

Liu, X., Si, S., Cao, Q., Kumar, S. and Hsieh, C.J., 2019. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*.

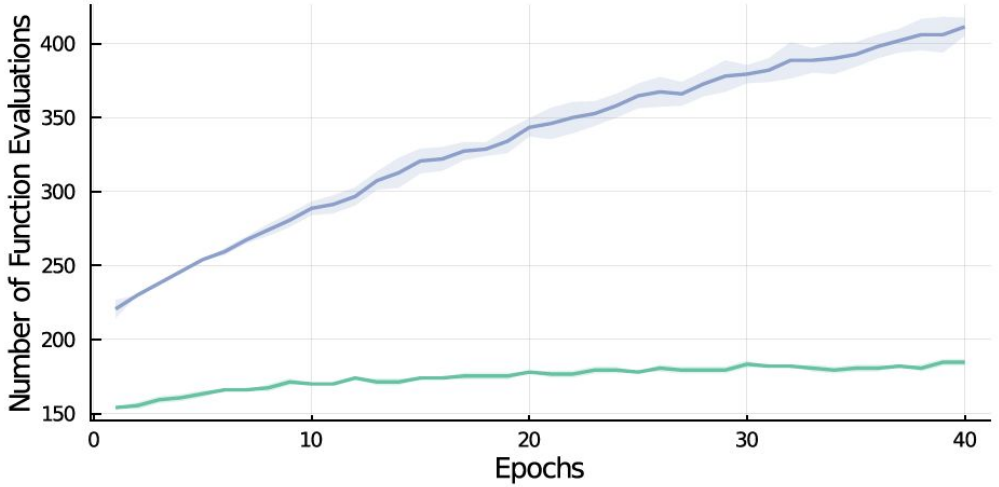
Rackauckas, C. and Nie, Q., 2020, September. Stability-optimized high order methods and stiffness detection for pathwise stiff stochastic differential equations. In 2020 IEEE High Performance Extreme Computing Conference (HPEC) (pp. 1-8). IEEE. **(Quality Submission Award)**

# Major improvements to Neural SDEs on MNIST

| Method | Train Accuracy (%) | Test Accuracy (%) | Train Time (hr) | Prediction Time (s) |
|---|---|---|---|---|
| Vanilla NSDE | 98.97 | 96.95 | 6.32 | 15.07 |
| **RegNSDE** | **98.16** | **96.27** | **4.19** | **7.23** |

**Double Neural SDE prediction speed!**



# Neural ODEs to Infinity are an “Infinite Layer” Network which can train fast!

# DEQs are Steady State Problems

Discrete Formulation

$$\text{Find } z \text{ s.t. } \overbrace{z^\star = \underbrace{f_\theta(z^\star, x)}_{\text{Neural Network}}}^{\text{Fixed Point Iterations}}$$

Continuous Formulation

$$\text{Find } z(\infty) \text{ s.t. } \overbrace{\frac{\mathrm{d}z}{\mathrm{d}t} = \underbrace{f_\theta(z, t)}_{\text{Neural Network}} - z}^{\text{Steady State Problem}}$$

[1] Avik Pal, Alan Edelman, Chris Rackauckas, IEEE HPEC 2023. Continuous Deep Equilibrium Models: Training Neural ODEs Faster by Integrating Them to Infinity

# DEQs are Steady State Problems

$$\text{Find } z(\infty) \text{ s.t. } \overbrace{\frac{\mathrm{d}z}{\mathrm{d}t} = \underbrace{f_\theta(z, t)}_{\text{Neural Network}} - z}^{\text{Steady State Problem}}$$

Adaptive ODE Solver with Terminating Callback

$$z(\infty) = \begin{aligned} & z(T) = \int_{t_0}^{T} (f_\theta(z(t), t) - z(t))\mathrm{d}t \\ & \text{s.t.} \quad f_\theta(z(T), T) = z(T) \end{aligned}$$

[1] Avik Pal, Alan Edelman, Chris Rackauckas, IEEE HPEC 2023. Continuous Deep Equilibrium Models: Training Neural ODEs Faster by Integrating Them to Infinity

# Infinite-Time Neural ODE

$$z(\infty) = z(T) = \int_{t_0}^{T} (f_\theta(z(t), t) - z(t))\mathrm{d}t$$
$$\text{s.t. } f_\theta(z(T), T) = z(T)$$

Continuous DEQ

$$z(T) = \int_{t_0}^{T} f_\theta(z(t), t)\,\mathrm{d}t$$

Neural ODE

[1] Avik Pal, Alan Edelman, Chris Rackauckas, IEEE HPEC 2023. Continuous Deep Equilibrium Models: Training Neural ODEs Faster by Integrating Them to Infinity

# Won't Back Propagation be Expensive?

$$f_\theta(z(T), T) = z(T)$$

Implicit Function

$$\frac{dg_\theta(z^*)}{d\theta} = \frac{\partial g_\theta(z^*)}{\partial \theta} + \frac{\partial g_\theta(z^*)}{\partial z^*} \overbrace{\left(I - \frac{\partial f_\theta(z^*)}{\partial z^*}\right)^{-1}}^{\text{Linear System}} \underbrace{\frac{\partial f_\theta(z^*)}{\partial \theta}}_{\frac{\partial z^*}{\partial \theta}}$$

Only need to backpropogate at the final point!

[1] Avik Pal, Alan Edelman, Chris Rackauckas, IEEE HPEC 2023. Continuous Deep Equilibrium Models: Training Neural ODEs Faster by Integrating Them to Infinity

# Neural ODE Adjoints are Really Slow!

| Model | Continuous | # of Params | Test Accuracy (%) | Training Time (s / batch) | Backward Pass (s / batch) | Prediction Time (s / batch) |
|---|---|---|---|---|---|---|
| Vanilla DEQ | ✗ | 10.63M | $88.913 \pm 0.287$ | $0.625 \pm 0.165$ | $0.111 \pm 0.021$ | $0.414 \pm 0.222$ |
| | ✓ | | $89.367 \pm 0.832$ | $1.284 \pm 0.011$ | $0.739 \pm 0.003$ | $0.606 \pm 0.010$ |
| Skip DEQ | ✗ | 11.19M | $88.783 \pm 0.178$ | $0.588 \pm 0.042$ | $0.112 \pm 0.006$ | $0.314 \pm 0.017$ |
| | ✓ | | $89.600 \pm 0.947$ | $0.697 \pm 0.012$ | $0.150 \pm 0.013$ | $0.625 \pm 0.004$ |
| Skip Reg. DEQ | ✗ | 10.63M | $88.773 \pm 0.115$ | $0.613 \pm 0.048$ | $0.109 \pm 0.008$ | $0.268 \pm 0.031$ |
| | ✓ | | $90.107 \pm 0.837$ | $0.660 \pm 0.019$ | $0.125 \pm 0.003$ | $0.634 \pm 0.019$ |
| Neural ODE | ✓ | 10.63M | $89.047 \pm 0.116$ | $5.267 \pm 0.078$ | $4.569 \pm 0.077$ | $0.573 \pm 0.010$ |

Table 3: **CIFAR10 Classification with Large Neural Network**: Skip Reg. Continuous DEQ achieves the *highest test accuracy*. Continuous DEQs are faster than Neural ODEs during training by a factor of $4.1\times - 7.98\times$, with a speedup of $6.18\times - 36.552\times$ in the backward pass. However, we observe a prediction slowdown for Continuous DEQs of $1.4\times - 2.36\times$ against Discrete DEQs and $0.90\times - 0.95\times$ against Neural ODE.

[1] Avik Pal, Alan Edelman, Chris Rackauckas, IEEE HPEC 2023. Continuous Deep Equilibrium Models: Training Neural ODEs Faster by Integrating Them to Infinity

# Neural ODE Adjoints are Really Slow!

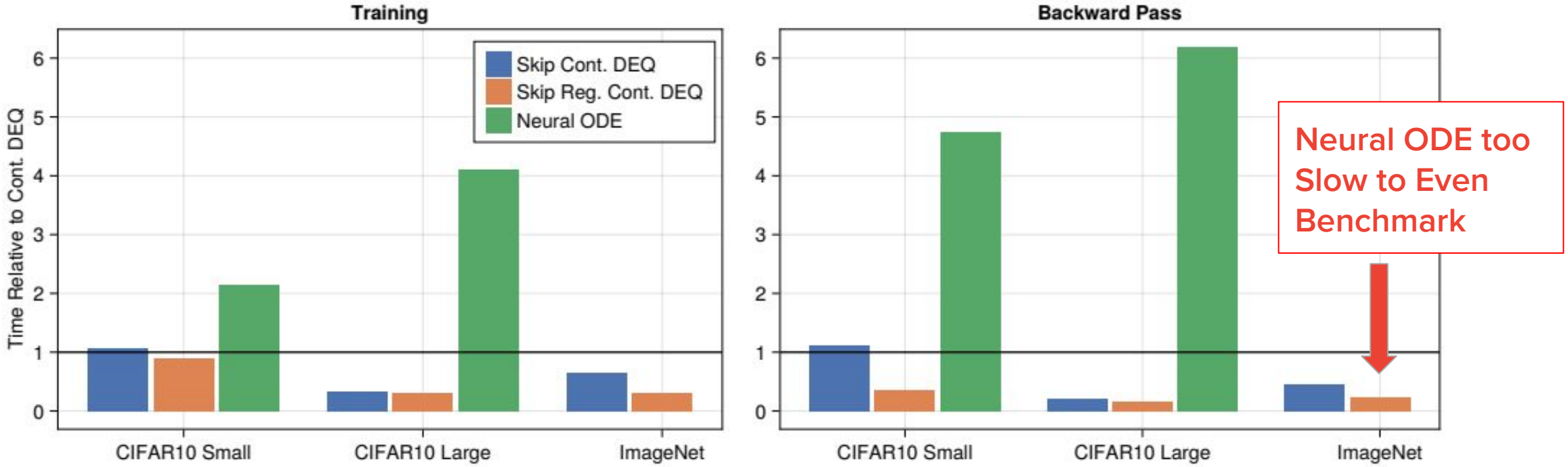



Figure 1: **Relative Training and Backward Pass Timings against Continuous DEQs** *(lower is better)*: In all scenarios, Neural ODEs take $4.7 - 6.182\times$ more time in the backward pass compared to Vanilla Continuous DEQs. Whereas combining Skip (Reg.) with Continuous DEQs accelerates the backward pass by $2.8 - 5.9\times$.

[1] Avik Pal, Alan Edelman, Chris Rackauckas, IEEE HPEC 2023. Continuous Deep Equilibrium Models: Training Neural ODEs Faster by Integrating Them to Infinity

# Many other SciML Methods can be rephrased as UDEs

# Solving Semilinear Parabolic PDEs (Nonlinear Black-Scholes) by Learning Structured Stochastic Differential Equations

- Semilinear Parabolic Form (Diffusion-Advection Equations, Hamilton-Jacobi-Bellman, Black-Scholes)

$$\frac{\partial u}{\partial t}(t,x)+\frac{1}{2}\mathrm{Tr}\left(\sigma\sigma^{\mathrm{T}}(t,x)(\mathrm{Hess}_x u)(t,x)\right)+\nabla u(t,x)\cdot\mu(t,x) \\ +f\left(t,x,u(t,x),\sigma^{\mathrm{T}}(t,x)\nabla u(t,x)\right)=0 \qquad [1]$$

Then the solution of Eq. 1 satisfies the following BSDE (cf., e.g., refs. 8 and 9):

$$u(t,X_t)-u(0,X_0) \\ =-\int_0^t f\left(s,X_s,u(s,X_s),\sigma^{\mathrm{T}}(s,X_s)\,\nabla u(s,X_s)\right)ds \qquad [3] \\ +\int_0^t [\nabla u(s,X_s)]^{\mathrm{T}}\,\sigma(s,X_s)\,dW_s.$$

- Make $(\sigma^T \nabla \mathrm{u})(t,X)$ a neural network.
- Solve the resulting SDEs and learn $\sigma^T \nabla \mathrm{u}$ via:

$$l(\theta)=\mathbb{E}\left[\left|g(X_{t_N})-\hat{u}\left(\{X_{t_n}\}_{0\le n\le N},\{W_{t_n}\}_{0\le n\le N}\right)\right|^2\right].$$

**Simplified:**

- Transform the PDE into a Backwards SDE: a stochastic boundary value problem. The unknown is a function!
- Learn the unknown function via neural network.
- Once learned, the PDE solution is known.

*Solving high-dimensional partial differential equations using deep learning, 2018, PNAS, Han, Jentzen, E*

# Solving Semilinear Parabolic PDEs (Nonlinear Black-Scholes) by Learning Structured Stochastic Differential Equations

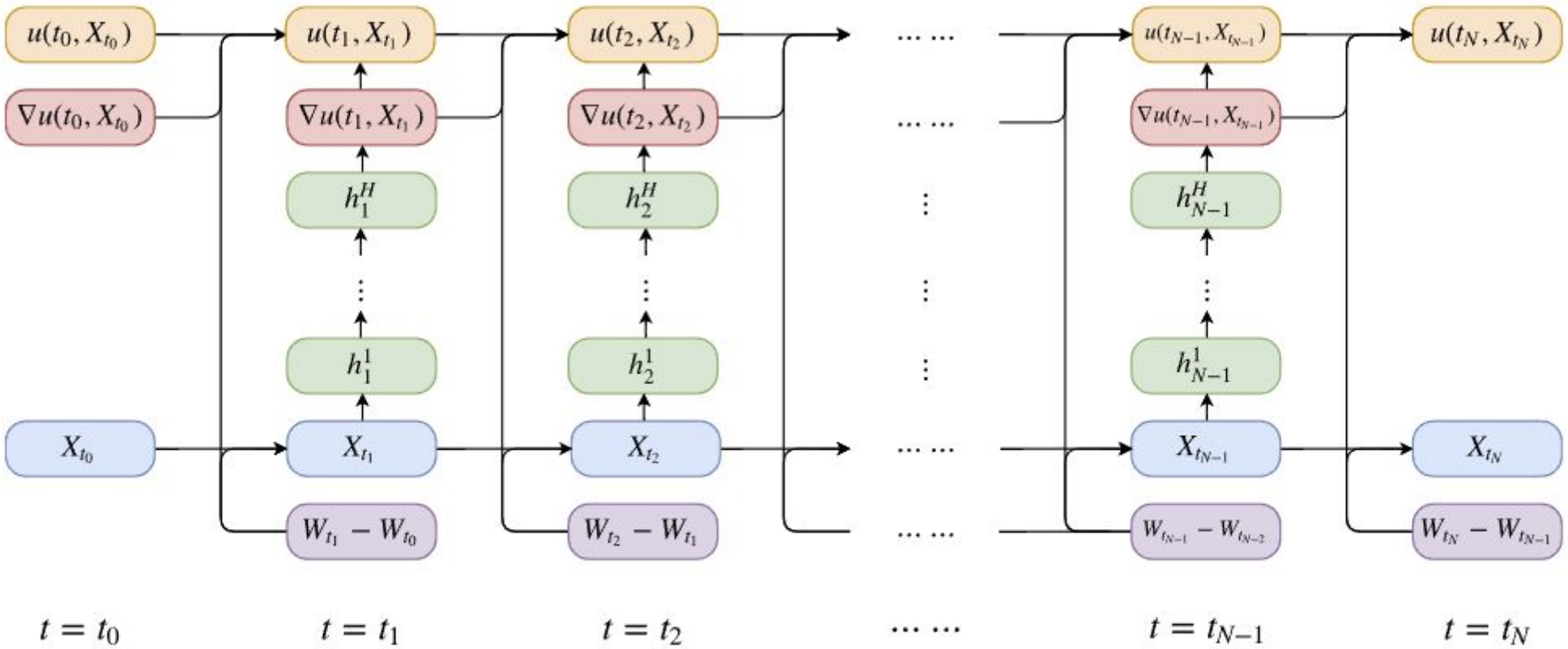


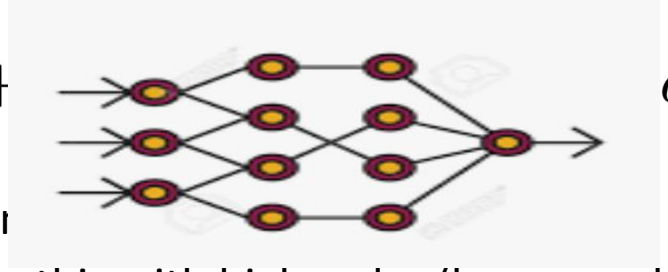

**Fig. 4.** Illustration of the network architecture for solving semilinear parabolic PDEs with $H$ hidden layers for each subnetwork and $N$ time intervals. The whole network has $(H+1)(N-1)$ layers in total that involve free parameters to be optimized simultaneously. Each column for $t = t_1, t_2, \ldots, t_{N-1}$ corresponds to a subnetwork at time $t$. $h_n^1, \ldots, h_n^H$ are the intermediate neurons in the subnetwork at time $t = t_n$ for $n = 1, 2, \ldots, N-1$.

*Solving high-dimensional partial differential equations using deep learning, 2018, PNAS, Han, Jentzen, E*

# Solving Semilinear Parabolic PDEs (Nonlinear Black-Scholes) by Learning Structured Stochastic Differential Equations

- This is equivalent to training the universal SDE:

$$
\begin{aligned}
dX_t &= \mu(t, X_t)dt + \sigma(t, X_t)dW_t, \\
dU_t &= f(t, X_t, U_t, \sigma^T(t, X_t)\nabla u(t, X_t))dt \\
&+ \qquad\qquad\qquad\qquad\qquad\qquad dW_t,
\end{aligned}
$$



- Solves a 100 dimensional PDE i

$$l(\theta) = \mathbb{E}\left[\left|g(X_{t_N}) - \hat{u}\left(\{X_{t_n}\}_{0\le n\le N}, \{W_{t_n}\}_{0\le n\le N}\right)\right|^2\right].$$

- As a universal SDE, we can solve this with high order (less neural network evaluations), adaptivity, etc. methods using DiffEqFlux.jl. Thus we automatically extend the deep BSDE method to better discretizations by using this interpretation!

# HighDimPDE.jl

**HighDimPDE.jl** is a Julia package to **solve Highly Dimensional non-linear, non-local PDEs** of the form

$$(\partial_t u)(t,x) = \int_\Omega f\big(t,x,\mathbf{x},u(t,x),u(t,\mathbf{x}),(\nabla_x u)(t,x),(\nabla_x u)(t,\mathbf{x})\big)\,d\mathbf{x} + \big\langle \mu(t,x),(\nabla_x u)(t,x)\big\rangle + \tfrac{1}{2}\mathrm{Trace}\big(\sigma(t,x)[\sigma(t,x)]^*(\mathrm{Hess}_x u)(t,x)\big).$$

where $u\colon [0,T]\times\Omega\to\mathbb{R}, \Omega\subseteq\mathbb{R}^d$ is subject to initial and boundary conditions, and where $d$ is large.

**HighDimPDE.jl** implements solver algorithms that break down the curse of dimensionality, including

- the Deep Splitting scheme
- the Multi-Level Picard iterations scheme.

To make the most out of **HighDimPDE.jl**, we advise to first have a look at the

- documentation on the Feynman Kac formula,

as all solver algorithms heavily rely on it.

## Algorithm overview

| Features | `DeepSplitting` | `MLP` |
|---|---|---|
| Time discretization free | ✗ | ✓ |
| Mesh-free | ✓ | ✓ |
| Single point $x\in\mathbb{R}^d$ approximation | ✓ | ✓ |
| $d$-dimensional cube $[a,b]^d$ approximation | ✓ | ✗ |
| GPU | ✓ | ✗ |
| Gradient non-linearities | ✓ | ✗ |

✓ : will be supported in the future