Trusty Neurocoder

Structured Surrogates for Scientific Simulation Kernels

Christopher J. Mungall
Lawrence Berkeley National Laboratory

The Core Problem

Many DOE science programs depend on large simulation codes that are:

- scientifically valuable
- expensive to run
- hard to calibrate against real data
- full of local submodels that are partly known and partly uncertain

Common fallback:

- fit a black-box neural surrogate

Common failure:

- it may reproduce outputs, but loses scientific structure and meaning

What Is a "Kernel" Here?

A kernel here is a **small, repeated scientific computation**, not the whole simulator.

Typical shape:

```
state_next = step(state, params, forcing)
```

or:

```
dstate_dt = rhs(state, params, forcing)
```

It is the local rule that says:

> given the current scientific state, how does it change?

Full Simulator vs Kernel

The full code usually looks like:

```
state = s0
for t in range(n_steps):
    state = step(state, params, forcing[t])
```

The **simulator** includes:

- timestepping
- I/O
- parallelization
- diagnostics
- checkpointing

The **kernel** is the `step(...)` or `rhs(...)` update rule.

Example Kernel

```
def update_pools(state, moisture, temp, params):
    decay_fast = k_fast * moisture_response(moisture) * state.fast
    decay_slow = k_slow * temp_response(temp) * state.slow
    return {
        "fast": state.fast - decay_fast,
        "slow": state.slow + decay_fast - decay_slow,
        "atm": state.atm + decay_slow,
    }
```

This is a good kernel because it is:

- scientifically meaningful
- called repeatedly
- small enough to isolate
- a plausible place to learn uncertain pieces

Why Not Learn the Whole Simulator?

Replacing the entire simulation with one neural net is usually a bad abstraction.

You lose:

- variable roles
- conservation structure
- module boundaries
- known physics vs unknown physics
- interpretability

For large models such as ecosystem simulators with many interacting variables, that loss of structure becomes a real scientific risk, not just an aesthetic problem.

What This Project Tries to Do

Extract a small scientific kernel from a larger codebase, then:

1. represent the known structure symbolically
2. compile that structure into a differentiable neural module
3. keep trusted physics fixed
4. make only uncertain parts learnable
5. fit those parts from data

Goal:

structured surrogate modeling, not unrestricted black-box replacement

The Pipeline

```
Scientific code
    ↓
Extract kernel / mini-module
    ↓
Encode known program structure
    ↓
Compile to NSAM / differentiable module
    ↓
Learn unknown parameters or subfunctions
    ↓
Use as calibrated module or fast surrogate
```

This repo is currently a proof of concept for that pipeline.

Architecture: From Simulator to Structured Surrogate

```
Large scientific codebase
    ↓
Extract one kernel / mini-module
    ↓
step(state, params, forcing)
    ↓
Represent the known scaffold symbolically
    ├─ variable roles stay explicit
    ├─ known physics stays fixed
    └─ unknown pieces become learnable "holes"
    ↓
Compile to NSAM / differentiable module
    ↓
Train on simulator traces, observations, or both
    ↓
Fast, calibrated, more interpretable local module
```

This is the intended architecture boundary:

What Gets Learned?

Three realistic modes:

1. **Parameter fitting**
    - known equation form
    - unknown constants
2. **Small unknown function fitting**
    - known scaffold
    - unknown response curve or sub-expression
3. **Local surrogate fitting**
    - preserve module boundary
    - replace an expensive local update with a cheaper learned approximation

Worked Example: EcoSIM-Style Soil Carbon Kernel

```
decay_fast = k_fast * f_theta(moisture) * state.fast
decay_slow = k_slow * temp_response(temp) * state.slow
```

Interpretation:

- `state.fast`, `state.slow`, `state.atm` are carbon pools
- `temp_response(temp)` is known or trusted
- `f_theta(moisture)` is uncertain and learnable
- pool-to-pool transfers and mass balance stay fixed

Training options:

- fit `f_theta` from expensive simulator traces
- fit `f_theta` or `k_fast` from field observations
- do both: simulator as prior, observations as correction

Where Does the Training Signal Come From?

Potential sources:

- **simulation outputs**
  - emulate an expensive code path
- **real observations**
  - calibrate model components to the world
- **hybrid workflows**
  - use both simulation traces and experimental or field data

So this can support:

- surrogate modeling
- parameter calibration
- partial system identification

Why Not Just Use PINNs?

PINNs are useful, but they solve a different problem.

Roughly:

- **PINNs**
  - learn solutions or latent functions while enforcing physics in training
- **This approach**
  - keeps an explicit program/module structure and learns only selected unknown parts

The point is not "better than PINNs at everything."

The point is:

more modular, more interpretable, and closer to the original scientific code structure

Comparison

Approach	What it learns	What it keeps	Strength	Weakness
Black-box surrogate	whole input-output map	very little	flexible, fast inference	hard to interpret or verify
PINN	solution or latent function under physics constraints	some physics in loss	physics-aware training	hard training, often not modular
NSAM kernel approach	parameters or subfunctions inside a known scaffold	program structure, variable roles, invariants	structured, interpretable, learnable	requires kernel extraction; still early-stage

The Value Proposition

Take a small scientific kernel from a simulation.

Then:

- preserve the known scientific structure
- compile it into a differentiable module
- learn only the uncertain pieces
- keep more semantics than a black-box surrogate
- potentially get a cheaper module for repeated use

Short version:

> Faster and learnable local scientific modules without throwing away the scientific story.

Why "Semantics" Matters

Semantics here means more than names.

It includes facts like:

- this variable is carbon in pool A
- this term is a branching ratio and should stay in `[0, 1]`
- this update should conserve total mass
- this sub-expression is known physics
- only this response function is uncertain

A black-box vector-to-vector model can hide all of that.

This approach tries to preserve it.

Why This Fits DOE-Style Simulation Work

DOE simulations often contain:

- expensive local update rules
- partial knowledge, not complete ignorance
- strong scientific constraints
- a need for calibration and surrogate modeling

That makes them a good match for:

- extracting a kernel
- preserving the scaffold
- learning the uncertain part

This is especially attractive where full black-box replacement would be scientifically or operationally unacceptable.

What the Repo Demonstrates Today

Eight working examples across DOE science domains:

Model	Domain	What's learned	Key result
Exponential decay	Foundation	rate k	recovered exactly
Coupled pools	Earth science	transfer α	recovered exactly
Unknown function	Earth science	Hill equation	decompiled from MLP
CENTURY-Lite	Earth science	Q10 + Hill	both forms recovered
Decay chain	Nuclear	branching ratios	exact (10^{-6})
Battery fade	Energy storage	SEI growth law	parabolic law recovered
Chemical kinetics	Combustion	Arrhenius rate	A=2.01, E=4.99
EcoSIM decomp	Earth science	T + water response	extracted from Fortran

Controlled Comparison: Cajal vs PINN vs Black-Box

Identical data, identical task (reversible reaction A⇌B):

	Black-box	PINN (λ=10)	Cajal
Interpolation MSE	6.2×10^{-3}	7.7×10^{-3}	**9.3×10^{-7}**
Conservation error	5.4×10^{-3}	6.5×10^{-3}	**1.6×10^{-7}**
Extrapolation MSE	3.1×10^{-2}	5.0×10^{-2}	**1.3×10^{-2}**
Sample efficiency (2 traj)	8.6×10^{-2}	—	**1.2×10^{-3}**
Interpretable?	No	No	**$k=1.97\cdot\exp(-4.86/T)$**

PINN conservation penalty *hurts* extrapolation (worse than black-box).

2-Paragraph Pitch

DOE science depends on large simulation codes that are expensive to run and difficult to calibrate, while many important submodels are only partially known. Black-box neural surrogates can reduce cost, but they often discard the scientific structure that tells us what variables mean, what should be conserved, and which parts of the model are trusted versus uncertain.

Trusty Neurocoder targets that gap by extracting small scientific kernels, preserving their known symbolic structure, and compiling them into differentiable neural modules. This makes it possible to learn parameters or unknown subfunctions from simulation traces, observations, or both, while keeping more semantics, interpretability, and verifiability than a generic neural surrogate. The current repo is a proof of concept, but the long-term goal is clear: structured, learnable surrogates for scientific kernels.

One-Slide Executive Summary

- **Problem:** DOE simulations are expensive, and their uncertain submodels are hard to calibrate.
- **Risk with black-box surrogates:** they lose scientific meaning and constraints.
- **Approach:** extract a kernel, keep known structure, compile to differentiable module, learn only uncertain parts.
- **Results:** 8 working models, 6,700× better than black-box, exact conservation, decompiles to interpretable math.
- **End-to-end:** agent reads EcoSIM Fortran → builds Cajal surrogate → trains → decompiles → verifies.
- **Status:** working prototype with PINN comparison; scaling to full EcoSIM output next.

Key Insight

> Do not replace the whole simulator.

Replace or calibrate the **right small kernel**.

Keep the scientific scaffold.
Learn the uncertain parts.

Trusty Neurocoder aims at structured surrogate modeling for scientific code.