Learning Explicit Structure of Dynamical Systems

Erik Arne Mathiesen-Dreyfus

Gimle Labs, `www.gimlelabs.com`

The Problem: Implicit Learning Sacrifices Structure

The dominant paradigm for learning dynamics is *implicit*: Neural ODEs [2], Physics-Informed Neural Networks [4], and related architectures learn black-box approximations of the form $\dot{x} = f_\theta(x)$, where f_θ is a neural network. While flexible, this approach sacrifices the structural properties that make classical system representations useful:

- **Interpretability**: Black-box dynamics cannot be inspected or validated against domain knowledge.
- **Composability**: Learned components cannot be meaningfully combined with analytical models.
- **Verifiability**: Standard tools for stability analysis and formal verification do not apply.
- **Generalization**: Implicit representations fail to transfer across operating regimes.

Classical methods (SINDy [1], symbolic regression [5]) discover explicit structure but require substantial domain expertise to design appropriate function libraries, scale poorly to high-dimensional systems, and struggle with hybrid or stochastic dynamics.

Key insight: What if we could learn *explicit* system representations (structured, typed, compositional), using the same scalable techniques that power modern foundation models?

A Typed Circuit Language for Dynamics

We introduce a formal language for dynamical systems inspired by traced monoidal categories [3] and proof theory. A circuit is a directed graph of typed computational elements composed via three combinators:

- **Composition** (sequential): $f \circ g$ — output of f feeds input of g.
- **Monoidal product** (parallel): $f \otimes g$ — independent, side-by-side execution.
- **Trace** (feedback): $\mathrm{Tr}(f)$ — routes an output back to an input, enabling recurrence and differential equations.

Each circuit has a *degree type* $(m \to n)$ specifying input and output wire counts. Composition requires matching degrees; the type system enforces well-formedness at parse time.

Atomic operations include constants, addition, multiplication, integration (`register`), differentiation (`deregister`), and transcendental functions. Complex systems are built by composing simpler circuits: e.g., a controller and plant are composed sequentially, while independent subsystems combine in parallel. Equations written in standard mathematical notation (e.g., $\dot{x} = -cx + u$) compile automatically to circuits via rewrite rules with trace wiring for feedback variables.

Three interchangeable calculi. The same circuit executes under different mathematical interpretations: *stream calculus* (deterministic ODEs via coinductive streams), *stochastic calculus* (SDEs via Itô or Stratonovich calculus), and *finite difference calculus* (difference equations and sequences). Execution is fully differentiable via JAX; feedback loops are resolved by fixed-point iteration on streams, and gradients flow end-to-end via automatic differentiation.

A Foundation Model for Circuit Discovery

Given observed trajectories from an unknown system, our foundation model discovers the circuit that generated them. Unlike symbolic regression over equation strings, it operates directly in the space of typed circuit structures.

Architecture. A transformer autoregressively generates tokenized circuit representations, conditioned on behavioral embeddings of the target trajectories. Crucially, generation is *grammar-constrained*: at each decoding step, an interactive LALR parser restricts the output to syntactically valid tokens. This guarantees 100% valid circuits, no post-hoc filtering required.

Training pipeline. Because the simulator can execute any well-typed circuit, we generate unlimited synthetic training data. The model proposes circuits, the simulator scores them, and the signal drives the next round of improvement. Crucially, before scoring, the parameters of each predicted circuit are optimized via gradient descent against the target trajectory, made possible by the full differentiability of the simulator. This decouples *structure search* from *parameter estimation*, letting the model focus on discovering the right topology. Training proceeds in three phases:

1. **Curriculum learning**: Supervised next-token prediction on synthetic (circuit, trajectory) pairs, progressing from simple atomics through compositions to full dynamical systems with trace and register. A REINFORCE-style behavioral loss is blended in as complexity increases: the model samples a complete circuit, the simulator scores it, and the reward updates the policy. Replay buffers prevent catastrophic forgetting across stages.
2. **Reward-ranked fine-tuning**: For each input, candidate circuits are sampled and scored by simulation fidelity. The best candidate becomes a supervised training target, iteratively sharpening the model toward high-reward circuits.
3. **Self-play search**: AlphaZero-style MCTS over token sequences [6], guided by learned policy and value heads. The model discovers circuits beyond what single-shot generation achieves.

Inference. Given observed trajectories, the model first generates a candidate circuit via autoregressive decoding. MCTS then explores variations of this initial candidate to improve structural fit. The best candidate is parameter-optimized against the target data via gradient descent through the differentiable simulator. The resulting circuit is read back as an interpretable equation, e.g., given oscillator trajectories, the model recovers $\ddot{x} + c\dot{x} + kx = 0$ rather than an opaque $\ddot{x} = \mathrm{NN}(x, \dot{x})$.

Preliminary Results

We evaluate on a benchmark of circuit recovery tasks spanning three difficulty levels: *algebraic* circuits (static compositions of atomics), *first-order dynamics* (exponential decay, growth, integration, circuits with a single traced register), and *second-order dynamics* (harmonic oscillators and spring systems, circuits with chained registers implementing $\dot{x} = v,\ \dot{v} = f(x) + u$). The benchmark progression directly tests whether the model can recover explicit dynamical structure of increasing complexity.

All results below are from a 23M-parameter model trained on a single laptop: 390k synthetic examples across 4 curriculum stages ($\sim$10M tokens), 20k RAFT examples, and 12.5k self-play games. For comparison, the smallest GPT-1 used $\sim$120M parameters on $\sim$1B tokens, our model operates at roughly $1/5$ the parameters and $1/100$ the data.

Training Phase	Degree	Recovery Rate			Opt.
	Match	Algebraic	1st-order	2nd-order	MSE
Curriculum (supervised)	84%	99%	60%	17%	0.06
+ RAFT	84%	100%	74%	20%	0.05
+ Self-play (MCTS)	82%	100%	79%	19%	0.09

The supervised curriculum baseline already achieves strong algebraic recovery and degree matching. The *optimized MSE* column isolates structural quality: after the model predicts a circuit, we run 50 steps of gradient descent on its parameters. This separates what the model must learn (topology) from what the simulator handles (parameter fitting). The low optimized MSE confirms the model finds structurally sound circuits even when initial parameters are imprecise.

RAFT improves first-order dynamical system recovery from 60% to 74% as well as secord-order recovery from 17% to 20% in only 3 iterations. Self-play via MCTS yields further gains over RAFT, improving recovery from 74% to 79% on second-order systems. While MCTS is the least explored stage of the pipeline, it already demonstrates the promise of search-guided refinement as a way to generate more complex circuits, including coupled systems.

Even at this early stage, these results are encouraging: they demonstrate that a small transformer, trained on modest data, with the right training setup, can learn to construct explicit, interpretable dynamical systems from observed behavior alone.

Future Work

Scaling. The current model (23M parameters) is comparable in size to a small GPT-1. The results are already promising, but we are in the infancy of scale, emergent capabilities in language models appeared only at 100–1000$\times$ this size.

Richer dynamics. The circuit language already supports stochastic and hybrid systems via interchangeable calculi. Extending the training data to include SDEs, coupled systems, hybrid automata, and more transcendentals is a natural next step.

Formal Verification. Explicit circuit representations enable stability analysis and formal verification. Verifiable properties can be incorporated as training objectives and at inference time to generate systems with guaranteed properties.

References

[1] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[2] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. Best Paper Award.

[3] André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.

[4] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[5] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[6] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.