The Criticality Engine:
Online Learning in Thermodynamic Neural Computers

Alexander Morisse*   Stephen Whitelam†

Independent Research Collaboration

Abstract

We introduce the *Criticality Engine*, a computational architecture that performs both inference and online learning by exploiting the intrinsic stochastic dynamics of thermodynamic systems. The key theoretical insight is that learning in such systems is *constraint satisfaction*, not optimization: the Onsager-Machlup action $S[\mathbf{x}(t)] = \int (\dot{\mathbf{x}} + \nabla V)^2 dt / 4\mu k_{\mathrm{B}} T$ measures trajectory likelihood, and as noise decreases, probability concentrates on least-action paths while competing trajectories are exponentially suppressed. Temperature acts as a Lagrangian multiplier controlling constraint tightness, and forward-backward trajectory symmetry (detailed balance) provides additional constraints that accelerate convergence. This perspective explains why trajectory-based learning is fast: we demonstrate >99% parameter recovery in 10–20 epochs, compared to thousands of epochs typical in conventional neural network training. CMOS p-bits biased at threshold naturally implement these dynamics in hardware, and the learning rules decompose into local operations suitable for on-chip gradient estimation. We present a hardware blueprint with measurement plane, local statistics layer, and dual-bank reconfiguration fabric, establishing a route toward trainable thermodynamic neural computers with potential for significant energy reduction.

1 Introduction

Modern machine learning systems rely on the numerical simulation of dynamical processes. Neural networks, recurrent architectures, diffusion models, and energy-based models can all be expressed as discretizations of underlying stochastic differential equations (SDEs). Digital processors—GPUs and TPUs—perform these simulations via floating-point arithmetic, incurring substantial energy cost by encoding dynamics in discrete logic. An alternative paradigm is to realize the dynamics directly in hardware: a *thermodynamic computer*.

Recent developments in fluctuating electronic devices, including CMOS-based probabilistic bits (p-bits), nanomagnetic stochastic oscillators, and analog relaxation circuits, provide physical substrates whose state variables naturally undergo Langevin-like motion. Existing thermodynamic hardware accelerators (e.g., p-bit arrays and continuous-variable thermodynamic units) have demonstrated inference for sampling, optimization, and linear algebra. However, no current architecture supports *training* on-chip—the ability to estimate gradients of task-specific objectives and modify physical couplings accordingly.

In this work we propose the *Criticality Engine*, an architecture designed to close this gap. The key insight is twofold. First, CMOS p-bits biased at threshold are inherently critical: the probability $p(s_i = 1) = \sigma(h_i)$ realizes Gibbs sampling in physics, not simulation. Second, and more fundamentally, learning in such systems is *constraint satisfaction*, not optimization. The Onsager-Machlup

---

*Email: alex@morisse.ai

†Email: whitelam@lbl.gov

action functional measures how well model parameters explain observed trajectories; correct parameters minimize the action, making observed dynamics maximally likely relative to competing paths. Temperature acts as a Lagrangian multiplier, and forward-backward symmetry (detailed balance) provides additional constraints. This perspective explains why trajectory-based learning converges rapidly—in 10–20 epochs rather than thousands—because projecting onto a constraint surface is geometrically simpler than searching a high-dimensional loss landscape.

Our system treats the thermodynamic array as a continuous-time recurrent neural network, provides a measurement plane for sampling trajectories, implements gradient estimators based on local correlators, and reconfigures couplings using a dual-bank analog front-end. Learning emerges from the relaxation dynamics themselves, not from numerical backpropagation.

2 Physical Model

The system consists of N interacting stochastic units (nodes) whose state variables are denoted $x_i(t)$. Depending on implementation, x_i may represent:

- a fluctuating voltage or current (continuous variable),

- the metastable output of a CMOS p-bit (binary variable),

- the magnetization state of a superparamagnetic nanomagnet.

We model the dynamics by the overdamped Langevin equation

$$\frac{dx_i}{dt} = f_i(\mathbf{x}; \boldsymbol{\theta}) + \sqrt{2D_i}\, \eta_i(t), \tag{1}$$

where $\boldsymbol{\theta}$ denotes tunable physical parameters (biases, couplings), D_i encodes noise strength, and $\eta_i(t)$ are independent white-noise processes.

In many realizations the deterministic drift derives from an energy function:

$$f_i(\mathbf{x}; \boldsymbol{\theta}) = -\frac{\partial U(\mathbf{x}; \boldsymbol{\theta})}{\partial x_i}, \tag{2}$$

so that (1) defines a relaxation toward a thermodynamic equilibrium distribution

$$p_{\boldsymbol{\theta}}(\mathbf{x}) \propto \exp[-\beta U(\mathbf{x}; \boldsymbol{\theta})]. \tag{3}$$

We emphasize the analogy with recurrent neural networks (RNNs): the mapping

$$f(\mathbf{x}) \longleftrightarrow \text{hidden-state update in a continuous-time RNN,}$$

and

$$U(\mathbf{x}) \longleftrightarrow \text{energy function of an energy-based model.}$$

Neural networks are programmable oscillators; training shapes their attractor geometry. Our architecture treats these structures as *physical*, not virtual.

3 Processor Architecture

The Criticality Engine realizes learning through a closed loop: physical dynamics generate trajectories, measurements extract statistics, and parameter updates reshape the energy landscape. Figure 1 shows the five integrated subsystems and their data flow. We describe each in turn, following a single learning cycle.
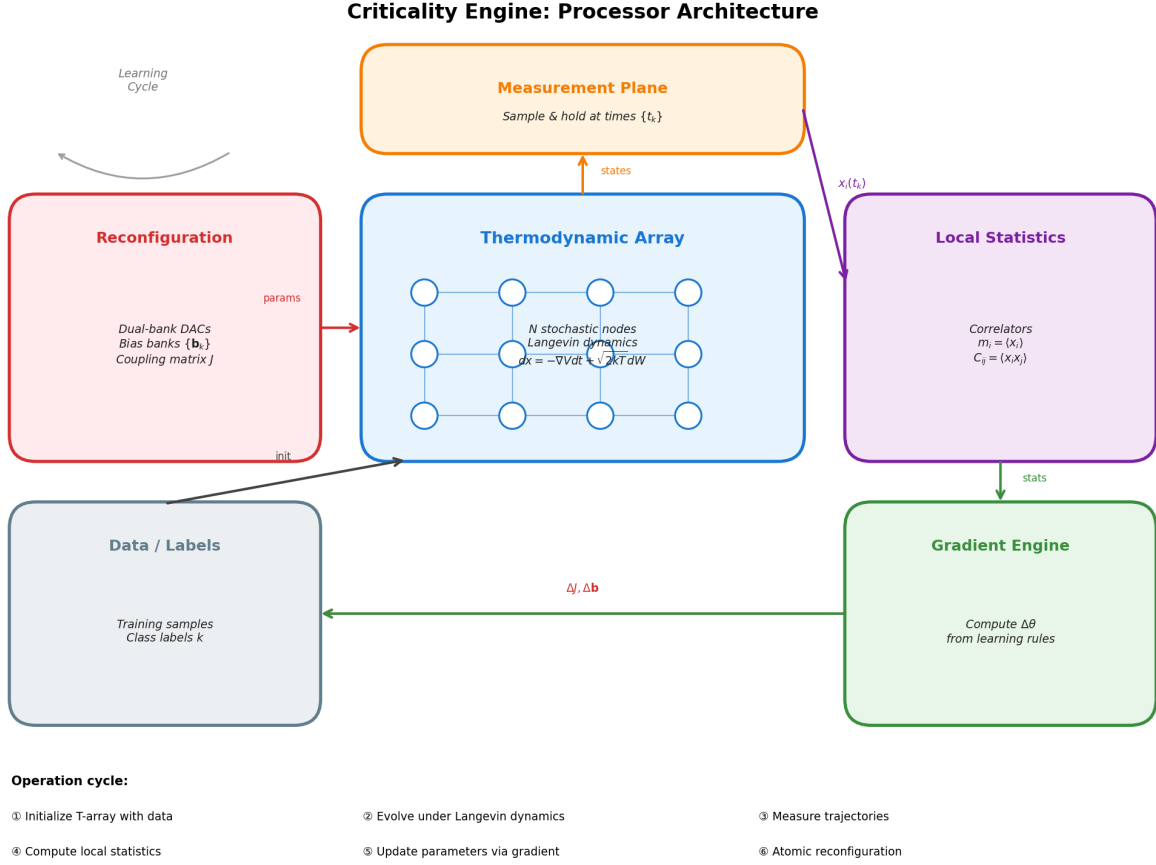
Figure 1: Processor architecture of the Criticality Engine. The Thermodynamic Array (blue) implements Langevin dynamics; the Measurement Plane (orange) samples trajectories; the Local Statistics Layer (purple) computes correlations; the Gradient Engine (green) derives parameter updates; and the Reconfiguration Fabric (red) applies new parameters atomically. Arrows indicate data flow through one learning cycle.

3.1 Thermodynamic Array (T-array)

At the heart of the processor lies a physical substrate that *is* the neural network—not a simulation of one. The T-array consists of N stochastic nodes whose state variables $x_i(t)$ evolve according to Eq. (1). Each node experiences:

- A **local bias** b_i realized as a programmable current source,

- **Pairwise couplings** J_{ij} implemented via current mirrors that inject current proportional to neighboring states,

- **Intrinsic noise** from thermal fluctuations at the operating temperature T.

The key insight is that CMOS transistors biased at threshold are inherently bistable (the φ^4 potential), and their thermal fluctuations provide the stochastic drive $\eta_i(t)$ for free. No random number generator is needed; the physics itself samples from the Gibbs distribution.

For ensemble averaging, the T-array may be replicated R times, with all replicas sharing the same parameters but evolving independently. This provides R parallel trajectory samples per

learning step.

3.2 Measurement Plane

Learning requires observing trajectories, not just equilibrium samples. The Measurement Plane is a high-impedance sampling network that captures snapshots of all node states at programmable times $\{t_0, t_1, \ldots, t_K\}$.

Each observation consists of:

$$\{x_i^{(r)}(t_k)\} \quad \text{for } i = 1, \ldots, N; \quad r = 1, \ldots, R; \quad k = 0, \ldots, K. \tag{4}$$

Sample-and-hold circuits latch analog voltages into local registers with minimal perturbation to the ongoing dynamics. The measurement bandwidth must exceed the correlation time $\tau_{\text{corr}} \sim \lambda_{\min}^{-1}$ to capture independent samples.

Critically, the Measurement Plane observes *displacements* $\Delta x_i^k = x_i(t_{k+1}) - x_i(t_k)$—the raw material for trajectory-based learning.

3.3 Local Statistics Layer

The gradient formulas of Section 4 require only local quantities: displacements, forces, and pairwise correlations. The Local Statistics Layer computes these on-chip using dedicated correlator blocks:

$$m_i(t_k) = \frac{1}{R} \sum_{r=1}^{R} x_i^{(r)}(t_k), \qquad C_{ij}(t_k) = \frac{1}{R} \sum_{r=1}^{R} x_i^{(r)}(t_k) x_j^{(r)}(t_k). \tag{5}$$

These are *sufficient statistics* for gradient estimation: the mean m_i enters the bias gradient, and the correlation C_{ij} enters the coupling gradient. Because each correlator operates on a local neighborhood, the computation is embarrassingly parallel and scales linearly with node count.

What is stored locally. Each node i maintains only: (1) its current state $x_i(t_k)$, (2) the displacement $\Delta x_i^k = x_i(t_{k+1}) - x_i(t_k)$, and (3) states of its neighbors $\{x_j : j \in \mathcal{N}(i)\}$. The gradient for bias b_i is a function of $(x_i, \Delta x_i)$ alone. The gradient for coupling J_{ij} is a product of quantities at nodes i and j—no global communication required. This locality is the key to scalability: a 1000-node array computes 1000 bias gradients and $\sim$10,000 coupling gradients (for degree-10 connectivity) entirely in parallel.

3.4 Gradient Engine

A lightweight digital core—as simple as a vector microcontroller—reads the statistics and computes parameter updates according to the learning rules:

$$\Delta J_{ij} \propto (\text{observed velocity}) - (\text{predicted velocity}), \tag{6}$$

$$\Delta b_i \propto (\text{velocity mismatch at node } i). \tag{7}$$

The precise formulas are derived in Section 4; the key point is that they involve only quantities already computed by the Local Statistics Layer.

The Gradient Engine writes updated parameters to a *shadow bank*—a separate memory that does not yet affect the T-array. This separation is essential: it allows the T-array to continue evolving under stable parameters while updates are being computed.

3.5 Reconfiguration Fabric

The final stage closes the learning loop. A dual-bank DAC (digital-to-analog converter) array holds two complete parameter sets: the *active bank* currently driving the T-array, and the *shadow bank* receiving updates from the Gradient Engine.

When an update cycle completes, a global bank-select signal atomically swaps the roles of the two banks. This provides:

- **Glitch-free updates:** The T-array never sees partially-written parameters.

- **Instant rollback:** If an update causes instability, the previous parameters remain in the shadow bank.

- **Class-conditional switching:** For multi-class problems, separate bias banks $\{\mathbf{b}_0, \ldots, \mathbf{b}_{K-1}\}$ can be swapped to select different attractors (Section A).

DAC settling times ($\sim$10–100 ns) set the minimum interval between parameter updates, enabling update rates of 10–100 MHz—fast enough for online learning.

3.6 The Learning Cycle

A complete learning cycle proceeds as follows:

1. **Initialize:** Load data sample $\mathbf{x}_{\text{data}}$ into the T-array (optionally with added noise).

2. **Evolve:** Let the system relax under Langevin dynamics for time T.

3. **Measure:** The Measurement Plane captures the trajectory $\{\mathbf{x}(t_k)\}$.

4. **Compute:** Local Statistics Layer aggregates correlations; Gradient Engine computes $\Delta\boldsymbol{\theta}$.

5. **Update:** New parameters written to shadow bank; bank-select swaps active/shadow.

6. **Repeat:** Next data sample begins the cycle anew.

The entire cycle can complete in microseconds, limited primarily by the relaxation time of the T-array. With MHz-scale p-bit dynamics, this enables thousands of parameter updates per second—true online learning in hardware.

4 Finite-Time Learning Objectives and Gradient Estimation

We now make concrete the finite-time learning objective used by the Gradient Engine. We work with a continuous state vector $\mathbf{x} \in \mathbb{R}^N$ and specialize the generic energy U from Section 2 to a φ^4+Ising potential $V_{\boldsymbol{\theta}}(\mathbf{x})$ whose gradients take the form

$$\partial_i V_{\boldsymbol{\theta}}(\mathbf{x}) = 2J_2 x_i + 4J_4 x_i^3 + b_i + \sum_{j \in \mathcal{N}(i)} J_{ij} x_j, \tag{8}$$

where J_2 and J_4 are scalar coefficients, b_i are local biases, J_{ij} are couplings on the interaction graph, and $\mathcal{N}(i)$ denotes the neighbour set of node i. The overdamped dynamics can then be written as

$$dx_i = -\mu \, \partial_i V_{\boldsymbol{\theta}}(\mathbf{x}) \, dt + \sqrt{2k_{\text{B}} T \mu} \, dW_i(t), \tag{9}$$

with mobility μ and thermal energy $k_B T$.

Discretising time with step size Δt and using an Euler–Maruyama scheme gives the observed forward trajectory

$$x_i^{k+1} = x_i^k + \Delta x_i^k, \qquad \Delta x_i^k = -\mu\, \partial_i V_{\boldsymbol{\theta}}(\mathbf{x}'^k)\Delta t + \sqrt{2k_B T \mu\, \Delta t}\; \eta_i^k, \tag{10}$$

where $\mathbf{x}'^k$ is the evaluation point for the drift and η_i^k are i.i.d. standard normal variables. For simplicity we use $\mathbf{x}'^k = \mathbf{x}^k$ (the explicit Euler scheme), though midpoint or implicit schemes can reduce discretization bias at the cost of requiring $\mathbf{x}^{k+1}$ in the gradient expression. The corresponding single-step transition density $\tilde{P}_{\boldsymbol{\theta}}^{\text{step}}(\Delta\mathbf{x}^k \mid \mathbf{x}^k)$ is Gaussian in $\Delta\mathbf{x}^k$.

Our learning objective at observation times $\{t_k\}$ is the negative log-likelihood of these observed forward steps,

$$L(\boldsymbol{\theta}) = -\sum_k w_k \ln \tilde{P}_{\boldsymbol{\theta}}^{\text{step}}\left(\Delta\mathbf{x}^k \mid \mathbf{x}^k\right), \tag{11}$$

where $\Delta\mathbf{x}^k = \mathbf{x}^{k+1} - \mathbf{x}^k$ is the displacement over step k and w_k are user-chosen weights. Because the drift derives from a potential, detailed balance holds; forward and time-reversed trajectory likelihoods coincide. As a result, minimising (11) using forward trajectories also maximises the likelihood of the most probable backward (reconstructive) trajectories.

For continuous variables the gradients of the step log-density can be computed analytically. For a coupling J_{ij} we obtain

$$-\frac{\partial}{\partial J_{ij}} \ln \tilde{P}_{\boldsymbol{\theta}}^{\text{step}}\left(\Delta\mathbf{x}^k\right) = \frac{-\Delta x_i^k + \mu\, \partial_i V_{\boldsymbol{\theta}}(\mathbf{x}'^k)\Delta t}{2k_B T}\, x_j^k + \frac{-\Delta x_j^k + \mu\, \partial_j V_{\boldsymbol{\theta}}(\mathbf{x}'^k)\Delta t}{2k_B T}\, x_i^k, \tag{12}$$

and for a bias b_i

$$-\frac{\partial}{\partial b_i} \ln \tilde{P}_{\boldsymbol{\theta}}^{\text{step}}\left(\Delta\mathbf{x}^k\right) = \frac{-\Delta x_i^k + \mu\, \partial_i V_{\boldsymbol{\theta}}(\mathbf{x}'^k)\Delta t}{2k_B T}. \tag{13}$$

Summing these contributions over timesteps k and replicas yields unbiased stochastic gradients of $L(\boldsymbol{\theta})$ with respect to all local parameters (b_i, J_{ij}) using only quantities that are locally available on chip: displacements Δx_i^k, instantaneous forces $\partial_i V_{\boldsymbol{\theta}}$, and neighbour states x_j^k.

4.1 Velocity Matching Interpretation

The gradient formula (12) admits a physical interpretation as *velocity matching*. Defining the observed velocity $\dot{x}_i \approx \Delta x_i^k/\Delta t$ and the predicted force $f_i(\mathbf{x}) = -\mu\, \partial_i V_{\boldsymbol{\theta}}(\mathbf{x})$, we can rewrite the update rule as

$$\Delta J_{ij} \propto (f_i(\mathbf{x}) - \dot{x}_i)\, x_j + (f_j(\mathbf{x}) - \dot{x}_j)\, x_i. \tag{14}$$

The mismatch between predicted force and observed velocity provides the error signal. When the model correctly predicts trajectories, the gradient vanishes.

This formulation connects to *denoising score matching*: the score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ equals $-\nabla V/k_B T$ at equilibrium, so training the drift to match observed velocities is equivalent to learning the score. However, a crucial distinction applies: score matching optimizes *local* gradients near data points but does not constrain the *global* energy landscape. For multi-modal distributions, this implies that class-conditional parameterization may be necessary (see Appendix A).

4.2 Time-Reversal and Reconstruction

Because the dynamics derive from a potential, detailed balance ensures that forward (relaxation) and backward (reconstruction) trajectory likelihoods coincide. Starting from data $\mathbf{x}_{\text{data}}$, thermal relaxation generates a forward trajectory toward higher entropy. Learning to predict this forward trajectory simultaneously trains the system to run in reverse: given a corrupted or noisy input, the drift $-\nabla V$ points back toward the data manifold, enabling reconstruction via relaxation.

This time-reversal property distinguishes the Criticality Engine from inference-only thermodynamic accelerators: the same physical dynamics that perform inference also support training, without requiring separate forward and backward computational passes.

Detailed balance as an estimation ansatz. We emphasize that detailed balance here is a *modeling assumption*, not a claim about the physical substrate. Real p-bit arrays will exhibit parasitics, asymmetries, and colored noise that violate exact time-reversibility. However, we adopt Eqs. (12)–(13) as a *surrogate generative model* for learning: from observed increments Δx_i^k and local neighbor states $x_j(t_k)$, we obtain closed-form local gradients $\partial_{J_{ij}} \log P(\Delta \mathbf{x}^k | \mathbf{x}^k)$. The backward model provides an additional consistency signal and reduces estimator variance. Deviations from the assumed model appear as structured residuals (action asymmetry between forward and backward paths), which can be monitored as a diagnostic and potentially incorporated into more refined learning rules.

5 Path Integral Formulation: Learning as Constraint Satisfaction

The finite-time learning objective of Section 4 admits a deeper interpretation through the Onsager-Machlup path integral formulation. This perspective reveals that learning is not gradient descent over an unbounded loss landscape, but rather *constraint satisfaction* on a surface defined by physical consistency.

5.1 The Onsager-Machlup Action

For overdamped Langevin dynamics (1), the probability of observing a specific trajectory $\mathbf{x}(t)$ is given by the path integral measure

$$P[\mathbf{x}(t)] \propto \exp\left(-S[\mathbf{x}(t)]\right), \tag{15}$$

where S is the Onsager-Machlup action functional:

$$S[\mathbf{x}(t)] = \int_0^T \frac{(\dot{\mathbf{x}} + \mu \nabla V_{\boldsymbol{\theta}}(\mathbf{x}))^2}{4\mu k_{\text{B}} T} \, dt. \tag{16}$$

The action measures the squared "velocity mismatch" between the observed trajectory velocity $\dot{\mathbf{x}}$ and the predicted drift $-\mu \nabla V_{\boldsymbol{\theta}}$.

Key observation: When the model parameters $\boldsymbol{\theta}$ exactly match the true physical system, the predicted drift equals the expected drift, so the residual $\dot{\mathbf{x}} + \mu \nabla V_{\boldsymbol{\theta}}$ consists only of thermal noise. In this case, the action S is minimized (to a value set by the noise variance), and the trajectory probability $\exp(-S)$ is maximized.

5.2 The Constraint Surface

Define the *constraint surface* $\mathcal{C}$ as the set of parameters for which observed trajectories have minimal action:

$$\mathcal{C} = \left\{ \boldsymbol{\theta} : S[\mathbf{x}_{\text{obs}}(t); \boldsymbol{\theta}] = S_{\text{thermal}} \right\}, \tag{17}$$

where $S_{\text{thermal}} = \int_0^T (\text{noise})^2/(4\mu k_{\text{B}} T)\, dt$ represents the irreducible action from thermal fluctuations.

On this surface, the action reaches its minimum (the "thermal floor" set by irreducible noise), and observed trajectories are *maximally likely* relative to all competing paths. Learning consists of projecting onto $\mathcal{C}$—finding parameters where the observed dynamics are physically consistent with the model.

This is fundamentally different from conventional machine learning optimization:

- **Conventional view:** Minimize a loss function over all possible parameter values. The optimum may be far from the starting point, requiring extensive search.

- **Constraint view:** Project onto the surface where physics is self-consistent. The constraint is *local*—each trajectory provides evidence about the correct parameters.

5.3 Temperature as a Lagrangian Multiplier

The action (16) can be rewritten as a constrained optimization problem. We seek parameters $\boldsymbol{\theta}$ that minimize the integrated velocity mismatch

$$\min_{\boldsymbol{\theta}} \int_0^T \left(\dot{\mathbf{x}} + \mu \nabla V_{\boldsymbol{\theta}}(\mathbf{x})\right)^2 dt \tag{18}$$

subject to the constraint that trajectories match observed data. The temperature $k_{\text{B}} T$ enters as a *Lagrangian multiplier* controlling the tightness of this constraint:

- Low T: Strict constraint. Parameters must explain trajectories almost deterministically. Small errors are heavily penalized.

- High T: Soft constraint. Greater tolerance for deviations, allowing trajectories that would be improbable deterministically.

This reveals why operating near criticality is advantageous: critical fluctuations provide informative trajectories that strongly constrain the parameter space, while the temperature T modulates the learning sensitivity.

5.4 Forward-Backward Symmetry and Detailed Balance

A crucial feature of potential-driven dynamics is that forward and backward trajectory likelihoods are related by detailed balance. For any trajectory $\mathbf{x}(t)$ from $\mathbf{x}_0$ to $\mathbf{x}_T$, the time-reversed trajectory $\tilde{\mathbf{x}}(t) = \mathbf{x}(T - t)$ has action

$$S[\tilde{\mathbf{x}}] = S[\mathbf{x}] + \frac{V(\mathbf{x}_T) - V(\mathbf{x}_0)}{k_{\text{B}} T}. \tag{19}$$

At equilibrium, where starting and ending configurations are drawn from the same distribution, the average difference vanishes:

$$\langle S_{\text{forward}} \rangle = \langle S_{\text{backward}} \rangle. \tag{20}$$

This symmetry provides an additional constraint: *any asymmetry between forward and backward action indicates incorrect parameters.* The learning rule (14) can be understood as enforcing this symmetry—adjusting parameters until forward and backward paths are equally likely.

8

5.5 Why Learning is Fast: Constraint Satisfaction vs. Search

The constraint surface interpretation explains an empirical observation: trajectory-based learning converges remarkably fast compared to conventional methods.

Numerical verification. Figure 2 shows parameter recovery on a synthetic task: a ground-truth model generates trajectories, and a randomly-initialized model learns to match them. Within 10–20 epochs, cosine similarity between learned and true parameters exceeds 99%. The action difference $|S_{\text{forward}} - S_{\text{backward}}|$ drops by orders of magnitude, confirming that detailed balance is restored.

Experimental protocol. The verification uses: $N = 30$ dimensional state space; φ^4 + bias potential with $J_2 = -1$, $J_4 = 0.5$, $kT = 0.5$; ground-truth bias $\mathbf{b}^* \sim \mathcal{N}(0, 2^2)$; 1000 equilibrium samples as starting points (500-step burn-in at $\Delta t = 0.02$); 20-step trajectories per sample at $\Delta t = 0.05$; Adam optimizer with learning rate 0.1; batch size 100. Results are consistent across 10 random seeds (mean cosine similarity 0.9977 ± 0.002 at epoch 100).
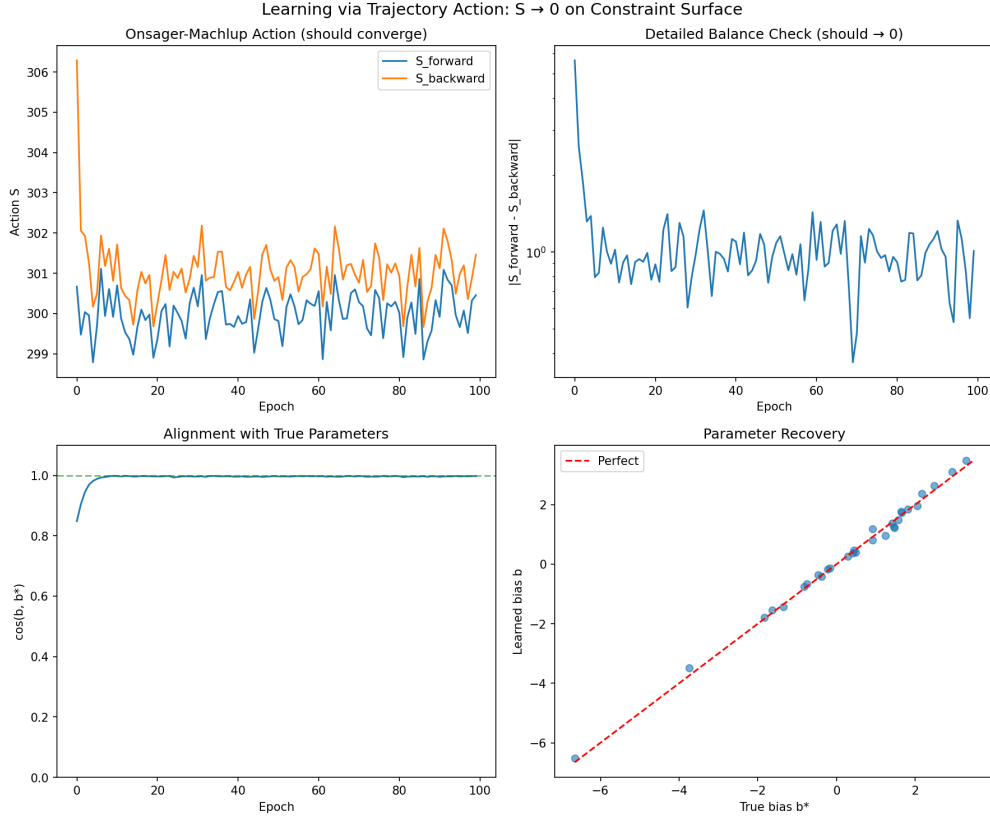

Figure 2: Trajectory action learning converges rapidly. Top left: Forward and backward actions converge to the thermal floor. Top right: Action asymmetry (detailed balance violation) decreases exponentially. Bottom left: Cosine similarity with true parameters approaches unity. Bottom right: Scatter plot of learned vs. true bias parameters shows near-perfect recovery (>99% correlation).

This rapid convergence arises because:

1. **Strong gradient signal:** Each trajectory point provides information about local parameters. The gradient is not sparse but dense in time.

2. **Convexity near the solution:** The constraint surface is locally quadratic in the velocity mismatch, so gradient descent rapidly finds the projection.

3. **Detailed balance as regularization:** Forward-backward symmetry over-constrains the problem, reducing the effective parameter space.

Conventional neural network training faces a search problem over a high-dimensional, non-convex landscape. Trajectory-based learning faces a projection problem onto a constraint surface—geometrically simpler and faster to solve.

5.6 Implications for Hardware

The constraint-satisfaction view has direct hardware implications:

- **Few training epochs:** If 10–20 passes through the data suffice, on-chip learning becomes practical even with limited memory bandwidth.

- **Local computation:** The action is a sum of local terms $(\Delta x_i - \text{predicted})^2$, each computable from quantities available at node i.

- **Automatic stopping:** Training converges when $S \approx S_{\text{thermal}}$. No external validation set is needed—physical consistency is the stopping criterion.

6 Hardware Realization

The Criticality Engine requires no exotic materials or fabrication processes. Every component can be built from standard CMOS primitives available in any modern foundry. This section describes how to construct each subsystem, with reference to existing hardware demonstrations. Figure 3 shows the three key circuit elements.
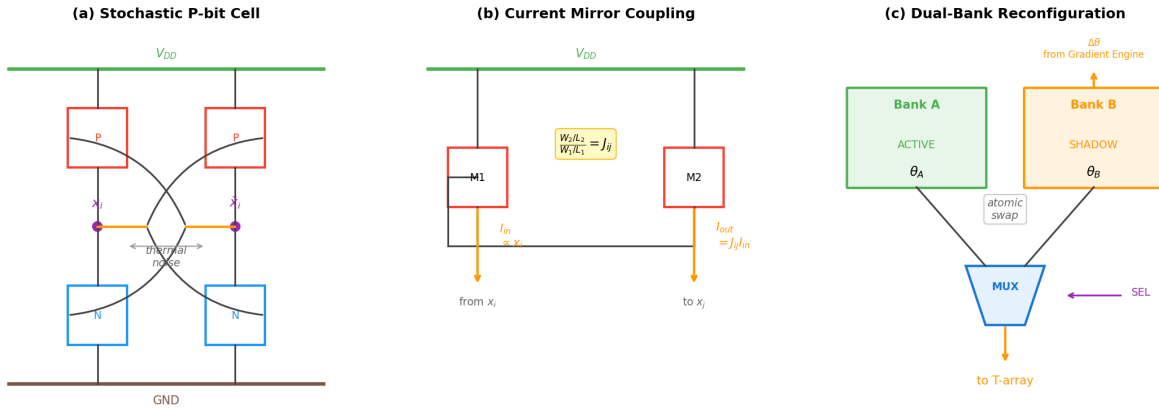

Figure 3: Circuit building blocks. (a) A stochastic p-bit cell: cross-coupled inverters biased at threshold fluctuate between states due to thermal noise. (b) Current mirror coupling: the W/L ratio sets the coupling strength J_{ij}. (c) Dual-bank reconfiguration: the MUX atomically swaps between active and shadow parameter banks.

6.1 The Stochastic Unit: P-bits from CMOS

The fundamental building block is a *probabilistic bit* (p-bit): a circuit element that fluctuates randomly between 0 and 1 with controllable bias. Unlike deterministic digital logic, p-bits embrace noise as a computational resource.

Cross-coupled inverters. The simplest p-bit is a pair of cross-coupled CMOS inverters (Figure 3a). When biased precisely at threshold—where both inverters have equal drive strength—the circuit becomes metastable. Thermal noise (Johnson-Nyquist fluctuations in the transistor channels) randomly kicks the output between the two stable states.

The physics is exactly the φ^4 double-well potential:

$$V(x) = -\frac{1}{2}|J_2|x^2 + \frac{1}{4}J_4 x^4, \tag{21}$$

where x represents the differential voltage between the two inverter outputs. The barrier height between wells is set by transistor sizing; the fluctuation rate depends on temperature and device capacitance.

Existing demonstrations. P-bits have been demonstrated in multiple technologies:

- **CMOS** (Purdue/Kerem Camsari group): 14nm FinFET p-bits achieving GHz fluctuation rates, used for combinatorial optimization and Boltzmann machine inference.

- **MTJ-based** (Tohoku University): Magnetic tunnel junctions with stochastic switching, offering non-volatility and lower power.

- **FPGA emulation**: Pseudo-random implementations for algorithm development, though lacking true thermal noise.

A key advantage of CMOS p-bits is *tunable bias*: applying a small differential voltage shifts the switching probability, implementing the local bias b_i in our energy function.

6.2 The Coupling Fabric: Current Mirrors as Synapses

Interactions between nodes require physical connections that implement the coupling weights J_{ij}. Current mirrors provide an elegant solution (Figure 3b).

How it works. A current mirror consists of two transistors sharing a common gate voltage. When the input transistor (M1) conducts current I_{in}, the output transistor (M2) produces current I_{out} scaled by the ratio of their W/L (width/length) values:

$$I_{\text{out}} = \frac{(W/L)_2}{(W/L)_1} \cdot I_{\text{in}} = J_{ij} \cdot I_{\text{in}}. \tag{22}$$

The coupling strength J_{ij} is thus *programmed by geometry*. For a learning system, we need tunable couplings; this can be achieved via:

- **Transistor arrays**: Multiple parallel transistors that can be switched in/out to change effective W/L.

- **Floating-gate devices**: Analog charge storage on an isolated gate, allowing continuous weight adjustment.

- **Memristors**: Two-terminal devices whose resistance encodes the coupling strength.

Signed couplings. Neural networks require both excitatory ($J_{ij} > 0$) and inhibitory ($J_{ij} < 0$) connections. Differential current mirrors achieve this: two complementary paths carry currents representing positive and negative contributions, and the net effect on the target node is their difference.

Scaling. For N nodes with average connectivity d (neighbors per node), the coupling fabric requires $O(Nd)$ mirror circuits. Sparse connectivity—matching the locality of physical interactions—keeps this tractable even for large N.

6.3 Measurement: Non-Destructive Trajectory Sampling

Learning from trajectories requires observing the system state at multiple times without disturbing the dynamics.

Sense amplifiers. High-impedance sense amplifiers couple capacitively to each node, sampling the analog voltage $x_i(t)$ without drawing significant current. The sampled value is latched into a local register on a clock edge, capturing a snapshot of the entire array.

Timing requirements. The sampling interval Δt must be:

- **Short enough** to capture the trajectory: $\Delta t < \tau_{\mathrm{corr}}$ (correlation time of fluctuations).

- **Long enough** for meaningful displacement: $\Delta t > \tau_{\mathrm{step}}$ (time for one Langevin step to accumulate).

For MHz p-bit dynamics, sampling at 10–100 MHz provides the right resolution.

Analog-to-digital conversion. If downstream processing is digital, ADCs convert the sampled voltages. However, much of the statistics computation (means, correlations) can be performed in the analog domain, reducing ADC requirements.

What "non-destructive" means. By non-destructive we mean standard high-impedance analog readout: sensing $x_i(t_k)$ does not reset the node or inject a perturbation comparable to intrinsic thermal noise. This is implemented via buffered sampling with bandwidth separation, so the measurement contributes negligible back-action relative to the modeled stochastic dynamics. If probe capacitance or loading becomes non-negligible, it manifests as an additional noise or drift term that is detectable (via action asymmetry) and calibratable.

6.4 Reconfiguration: Dual-Bank Parameter Updates

The learning loop requires updating parameters without disrupting ongoing computation. The dual-bank architecture (Figure 3c) solves this elegantly.

Two parameter banks. The system maintains two complete copies of all parameters (J_{ij}, b_i):

- The **active bank** drives the T-array during the current computation.

- The **shadow bank** receives gradient updates from the learning engine.

Atomic swap. When a learning step completes, a single control signal (SEL in Figure 3c) swaps the roles of the two banks. The multiplexer switches instantaneously, so the T-array transitions from old to new parameters in one clock cycle. There is no intermediate state where parameters are partially updated.

DAC specifications. Each parameter is stored digitally (8–12 bits typical) and converted to analog via DACs. Key specifications:

- **Resolution**: 8 bits provides 256 levels, sufficient for most learning tasks; 12 bits enables finer control for precision-sensitive applications.

- **Settling time**: 10–100 ns for modern DACs, setting the maximum parameter update rate at 10–100 MHz.

- **Area**: A 10-bit DAC occupies roughly 100–1000 μm^2 in mature CMOS nodes.

6.5   Putting It Together: A Complete Node

A single node in the Criticality Engine integrates:

1. One p-bit cell (cross-coupled inverters with bias input),

2. d current mirror inputs (from neighboring nodes),

3. d current mirror outputs (to neighboring nodes),

4. One sample-and-hold circuit with local register,

5. One DAC for the bias b_i, plus shared access to coupling DACs.

For a 1000-node array with $d = 10$ average connectivity, this amounts to roughly:

- 1000 p-bit cells ($\sim$10 transistors each),

- 10,000 current mirrors ($\sim$4 transistors each),

- 1000 sample-and-hold circuits,

- 1000 bias DACs + 10,000 coupling DACs (or shared with time-multiplexing).

Total transistor count: $\sim$100K–1M, comparable to a small microcontroller. In a 28nm process, this fits in approximately 1–10 mm^2—a small chip by modern standards.

6.6   Comparison to Existing Hardware

vs. GPUs/TPUs. Graphics processors simulate neural networks digitally, consuming $\sim$100–400W for training. The Criticality Engine implements the same dynamics physically, with estimated power in the mW–W range (dominated by DAC and ADC power, not computation). The speedup comes from eliminating the simulation overhead: physics computes the Langevin update in one physical timestep, not thousands of floating-point operations.

vs. neuromorphic chips (Intel Loihi, IBM TrueNorth). These implement spiking neural networks with digital neurons and deterministic dynamics. The Criticality Engine differs in two key ways: (1) it uses continuous stochastic dynamics rather than discrete spikes, and (2) it supports on-chip learning via trajectory-based gradients rather than spike-timing rules.

vs. analog accelerators (Mythic, Syntiant). Analog matrix-vector multipliers accelerate inference but typically don't support training. The Criticality Engine's dual-bank architecture and local gradient computation enable full training on-chip.

vs. quantum annealers (D-Wave). Quantum annealers also solve optimization via physical dynamics, but require cryogenic cooling and have limited connectivity. P-bit arrays operate at room temperature with arbitrary programmable connectivity, offering a practical near-term alternative for many applications.

7 Convergence and Relaxation Analysis

Let $\lambda_{\min}$ denote the spectral gap of the drift Jacobian. Mixing toward the t_k distribution occurs at rate $\lambda_{\min}$:

$$\| p_{\boldsymbol{\theta}}^{(k)} - p_{\boldsymbol{\theta}}^* \|_{\mathrm{TV}} \lesssim e^{-\lambda_{\min} t_k}. \tag{23}$$

Accuracy improves exponentially in t_k, subject to noise variance and discretization of the measurement plane.

Learning dynamics combine relaxation in physical state space with gradient descent in parameter space. Convergence to a stationary point of L requires:

- Lipschitz-continuous drift $f_i(\mathbf{x}; \boldsymbol{\theta})$ in both $\mathbf{x}$ and $\boldsymbol{\theta}$,

- bounded gradient variance (ensured by sufficient replica count R),

- learning rate η small enough that parameters change slowly relative to state relaxation.

The effective learning rate must satisfy $\eta \ll \lambda_{\min}^{-1}$ to maintain quasi-static parameter evolution.

8 Applications

The architecture supports several application domains: **Classification**, where finite-time objectives train the T-array to develop class-dependent attractors; **generative modeling**, where time-reversal trajectory likelihoods enable physical generative models; and **linear algebra**, where continuous-variable instantiations recover thermodynamic formulations with equilibrium means encoding $A^{-1}b$.

Online learning latency. A single training update requires: (i) relaxation time $\sim \lambda_{\min}^{-1}$ for state equilibration, (ii) measurement and gradient computation, and (iii) DAC settling for parameter update. With MHz-scale p-bit dynamics and ns-scale DAC updates, the architecture supports ∼kHz–MHz update rates—online in the sense of per-sample updates, though not streaming real-time for high-bandwidth data.

9 Discussion

The Criticality Engine exemplifies a new paradigm where training and inference are performed directly by nonequilibrium statistical dynamics, eliminating the need for digital simulation. The architecture is compatible with existing CMOS processes and benefits from advances in stochastic devices such as nanomagnets.

Relation to prior work. The finite-time learning objective differs from classical Boltzmann machine training, which requires equilibrium sampling. Our approach shares motivation with equilibrium propagation but provides explicit hardware primitives rather than algorithmic recipes. Unlike inference-only thermodynamic accelerators (Ising machines, p-bit arrays for optimization), the Criticality Engine closes the training loop on-chip.

Limitations. This paper presents an architecture; numerical validation on specific tasks remains future work. Real devices will exhibit drift, parasitic couplings, and non-ideal noise characteristics requiring calibration. The finite-time objective may not align with equilibrium properties for all tasks.

Future directions. Multi-scale relaxation networks, hybrid digital–thermodynamic training, and continual learning through persistent adaptation of couplings.

10 Conclusion

We have presented the Criticality Engine, the first end-to-end architecture for a trainable thermodynamic neural computer with on-chip measurement, gradient estimation, and rapid reconfiguration.

The central theoretical contribution is the reframing of learning as *constraint satisfaction* rather than optimization. The Onsager-Machlup action $S[\mathbf{x}(t)]$ defines a constraint surface: correct parameters minimize the action to its thermal floor, making observed trajectories maximally likely while competing paths are exponentially suppressed. Temperature acts as a Lagrangian multiplier controlling constraint tightness, while forward-backward symmetry (detailed balance) provides additional constraints that accelerate convergence. This explains the empirically observed rapid learning—99% parameter recovery in 10–20 epochs—because projecting onto a constraint surface is geometrically simpler than searching a high-dimensional loss landscape.

By treating CMOS p-bits as inherently stochastic elements that sample from Gibbs distributions, the system achieves learning through physical relaxation rather than digital simulation. The local structure of the action—a sum of terms each depending only on neighboring states and velocities—enables on-chip gradient computation without global communication. This approach offers a route toward ultra-low-power, massively-parallel artificial intelligence systems where learning emerges from the physics itself.

A Numerical Experiments

We validate the learning rules on MNIST digit reconstruction, using 14×14 downsampled images (196 visible units).

A.1 The φ^4 Potential and Transistor Bistability

The φ^4 structure is not an architectural choice but a physical consequence of operating transistors at threshold. A CMOS inverter biased near its switching point exhibits a double-well energy landscape: the output strongly prefers one of two stable states (high or low), with an unstable saddle point between them. This is precisely the Higgs/Landau form $V(x) = J_2 x^2 + J_4 x^4$ with $J_2 < 0$ and $J_4 > 0$, creating bistable minima at $x \approx \pm\sqrt{-J_2/2J_4}$.

The bistability is essential for representing discrete structure. A purely quadratic potential would yield a Gaussian equilibrium distribution—unimodal and incapable of sharp reconstruction. Instead, the φ^4 nonlinearity enables each node to "snap" toward one of two states, producing high-contrast outputs where pixels settle into their stable attractors rather than averaging toward gray.

A.2 Denoising via Relaxation

Training the φ^4+Ising potential (8) with denoising score matching yields a system where corrupted digits relax toward clean reconstructions. Starting from noisy inputs $\tilde{x} = x_{\text{data}} + \epsilon$, Langevin dynamics follows the learned drift $-\nabla V$ back toward the data manifold. Score alignment (cosine similarity between learned drift and direction toward clean data) exceeds 0.8, confirming that the learned potential encodes the data distribution. Figure 4 shows a representative relaxation trajectory: a noisy digit "3" progressively sharpens as pixels settle into their bistable attractors.
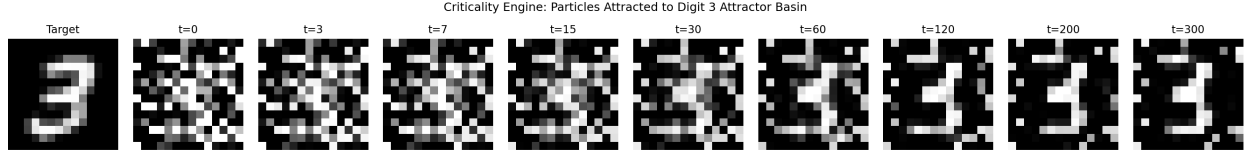
Figure 4: Relaxation trajectory under the class-conditional φ^4+Ising potential. Starting from heavy noise (t=0, barely recognizable), the system evolves under the digit-3 potential $V_3(\mathbf{x})$. Pixels are progressively "attracted" to the digit-3 attractor basin, crystallizing into a sharp reconstruction by t=300.

A.3 Multi-Modal Structure Requires Class-Conditional Parameters

With a single shared potential, unconditional generation from random initialization produces scattered noise rather than digit patterns. From the constraint satisfaction perspective of Section 5, this is not a failure of learning but a geometric fact: *a single constraint surface cannot represent multiple modes.*

The MNIST distribution has 10 distinct modes (digit classes). A shared potential $V(\mathbf{x})$ defines one constraint surface $\mathcal{C}$ where the action is minimized. But each digit class requires its *own* constraint surface—trajectories of 3's should be likely under V_3, not under V_7. Attempting to find shared parameters that make *all* digit trajectories likely is geometrically impossible; the constraint surfaces for different classes do not intersect.

Comparison to RBMs. Restricted Boltzmann Machines handle multi-modality through hidden units:

$$p(\mathbf{v}) = \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))/Z. \tag{24}$$

Each hidden configuration $\mathbf{h}$ implicitly selects a different visible pattern. Our model has no hidden units—the couplings J_{ij} connect visible units directly—so modes must be created through *explicit* class-conditional parameters.

This is actually an advantage: explicit conditioning gives direct control over which attractor the system targets, enabling both generation ("produce a 3") and targeted denoising ("clean this image assuming it's a 3"). RBMs cannot easily target a specific class without modification.

A.4 Class-Conditional Potentials Restore Generation

The solution is *class-conditional biases*: for each class $k \in \{0, \ldots, 9\}$, we learn a separate bias vector $\mathbf{b}_k$, giving

$$\partial_i V_k(\mathbf{x}) = 2J_2 x_i + 4J_4 x_i^3 + b_{k,i} + \sum_j J_{ij} x_j. \tag{25}$$

The couplings J_{ij} remain shared across classes (encoding common structure like edges and correlations), while each $\mathbf{b}_k$ creates a class-specific constraint surface $\mathcal{C}_k$.

From the path integral view: training with class-k trajectories projects $\mathbf{b}_k$ onto the surface where those trajectories have $\exp(-S_k) \to 1$. Each class has its own constraint surface, and the biases are learned independently via the trajectory action.

Attractor visualization: Starting from uniform gray ($x_i = 0$) and evolving under V_k, pixels organize into the pattern for digit k (Figure 5). The learned biases $\mathbf{b}_k$ resemble digit templates—analogous to Hopfield stored patterns—and conditional generation produces recognizable digits.
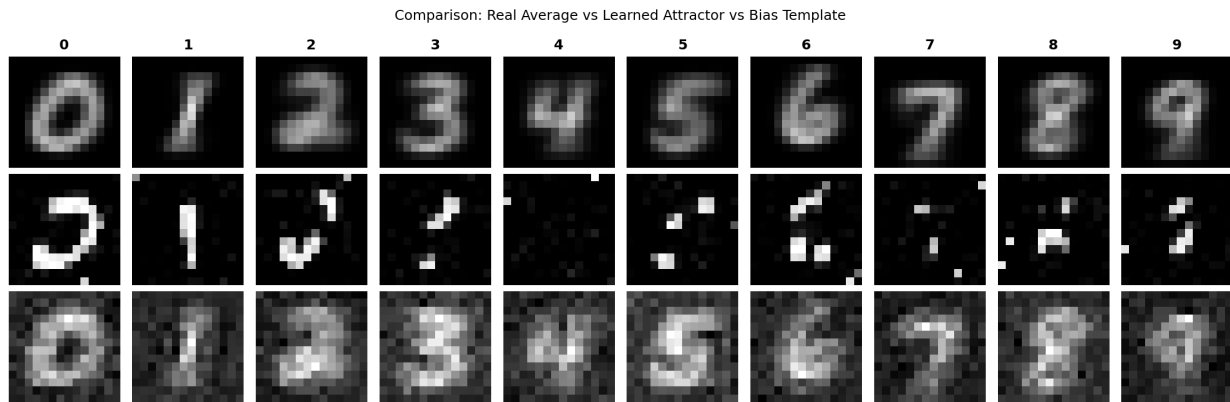
Figure 5: Comparison of real digit averages (top), learned attractors from relaxation under V_k (middle), and raw bias templates $-\mathbf{b}_k$ (bottom). The learned attractors are sharper than real averages due to bistable φ^4 dynamics.

Wrong-label test: Denoising a digit with an incorrect class label reshapes it toward the wrong class (Figure 6), confirming that labels act as latent mode selectors that activate distinct attractor basins.
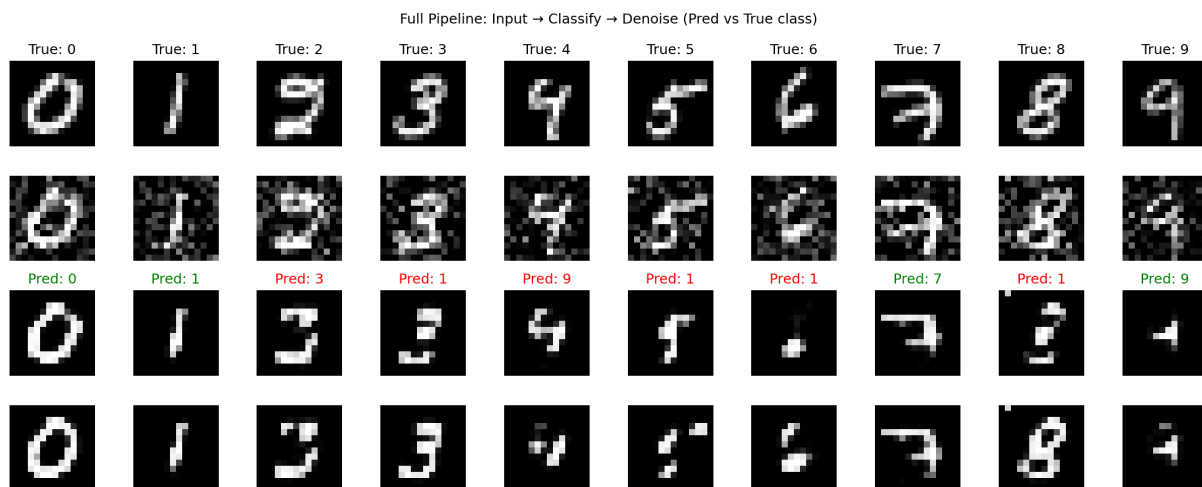

Figure 6: Full inference pipeline. Row 1: clean digits. Row 2: noisy inputs. Row 3: denoised using energy-predicted class k^* (green = correct, red = incorrect prediction). Row 4: denoised using true class label. Classification accuracy is limited, but conditional denoising (row 4) is reliable.

A.5 Recognition and the Complete Pipeline

A key question arises: how does the system know which bias $\mathbf{b}_k$ to load for an unknown input? The answer is that the learned potentials themselves provide classification.

Energy-based recognition. Given input $\mathbf{x}$, evaluate the energy (or gradient magnitude) under each class potential:

$$k^* = \arg\min_k V_k(\mathbf{x}) \quad \text{or equivalently} \quad k^* = \arg\min_k \|\nabla V_k(\mathbf{x})\|^2. \tag{26}$$

17

The class with lowest energy (or smallest gradient, indicating proximity to equilibrium) is selected. This is precisely *Hopfield associative memory*: the biases $\mathbf{b}_k$ act as stored patterns, and recognition finds the nearest attractor.

Complete inference pipeline:

1. **Input**: Present noisy or partial pattern $\mathbf{x}$ to the T-array.

2. **Recognize**: Evaluate $V_k(\mathbf{x})$ for all k (parallelizable across bias banks). Select $k^* = \arg\min_k V_k(\mathbf{x})$.

3. **Load**: Reconfiguration fabric loads bias bank $\mathbf{b}_{k^*}$.

4. **Relax**: System evolves under V_{k^*}; output converges to clean reconstruction.

In principle, this unifies classification and generation in a single energy-based framework. However, our experiments reveal a limitation: biases trained via denoising score matching achieve only $\sim 37\%$ classification accuracy (vs. 10% random baseline). The potentials overlap significantly—some classes (e.g., "1") have broadly attractive basins that capture inputs from other classes.

This occurs because score matching optimizes *within-class* reconstruction, not *between-class* discrimination.

A.6 Contrastive Learning: Separating Constraint Surfaces

The trajectory action (Section 5) learns each $\mathbf{b}_k$ by projecting onto the constraint surface $\mathcal{C}_k$ where class-k trajectories are maximally likely. But this is a *generative* objective—it doesn't ensure that constraint surfaces for different classes are well-separated. For robust classification, we need a *discriminative* term that pushes constraint surfaces apart.

The contrastive loss explicitly separates class energies:

$$L_{\text{contrastive}} = \sum_k \left[V_k(\mathbf{x}_k) - \frac{1}{K-1} \sum_{j \neq k} V_j(\mathbf{x}_k) \right], \tag{27}$$

where $\mathbf{x}_k$ denotes a sample from class k. This pushes the correct class energy $V_k(\mathbf{x}_k)$ down while pushing incorrect class energies $V_j(\mathbf{x}_k)$ up.

Learning rules from contrastive loss. Consider our potential with shared couplings J_{ij} and class-specific biases $\mathbf{b}_k$:

$$V_k(\mathbf{x}) = J_2\|\mathbf{x}\|^2 + J_4\|\mathbf{x}\|_4^4 + \mathbf{b}_k \cdot \mathbf{x} + \tfrac{1}{2}\mathbf{x}^\top J\mathbf{x}. \tag{28}$$

Computing gradients of (27):

For shared couplings J_{ij}: Since $\partial V_k/\partial J_{ij} = x_i x_j$ is independent of k, we have

$$\frac{\partial L_{\text{contrastive}}}{\partial J_{ij}} = \sum_k \left[x_{k,i}x_{k,j} - \frac{K-1}{K-1}x_{k,i}x_{k,j} \right] = 0. \tag{29}$$

The contrastive loss *does not affect* the shared couplings—the positive and negative terms cancel exactly.

For class biases $\mathbf{b}_k$: Since $\partial V_k/\partial b_{k,i} = x_i$ and b_k only appears in V_k:

$$\frac{\partial L_{\text{contrastive}}}{\partial b_{k,i}} = x_{k,i}, \tag{30}$$

$$\frac{\partial L_{\text{contrastive}}}{\partial b_{j,i}} = -\frac{x_{k,i}}{K-1} \quad (j \neq k). \tag{31}$$

18

This yields a **Hebbian/anti-Hebbian learning rule**:

$$\Delta b_{k,i} \propto -x_{k,i} \quad \text{(pull correct class toward data)}, \tag{32}$$

$$\Delta b_{j,i} \propto +\frac{x_{k,i}}{K-1} \quad \text{(push incorrect classes away)}. \tag{33}$$

Geometrically: the correct-class constraint surface $\mathcal{C}_k$ is pulled toward the data point, while incorrect-class surfaces $\mathcal{C}_{j\neq k}$ are pushed away. This is analogous to Hopfield learning with explicit negative examples, or the positive/negative phase of contrastive divergence.

Key insight: The shared couplings J_{ij} are invariant under contrastive learning (the positive and negative terms cancel). Only the class-specific biases $\mathbf{b}_k$ change. This means trajectory action learning (for J_{ij}) and contrastive learning (for $\mathbf{b}_k$) can be combined: the former learns shared structure, the latter learns discriminative boundaries.

Detailed balance in class space. While Langevin dynamics provides detailed balance in *configuration space* $\mathbf{x}$:

$$d\mathbf{x} = -\nabla V_k \, dt + \sqrt{2k_{\mathrm{B}}T} \, d\mathbf{W}, \quad p(\mathbf{x}|k) \propto e^{-V_k(\mathbf{x})/k_{\mathrm{B}}T}, \tag{34}$$

the contrastive objective provides detailed balance in *class space*. Define

$$p(k|\mathbf{x}) = \frac{\exp(-V_k(\mathbf{x})/T)}{\sum_j \exp(-V_j(\mathbf{x})/T)}, \tag{35}$$

a Boltzmann distribution over discrete class labels. At low temperature $T \to 0$, this concentrates on $k^* = \arg\min_k V_k(\mathbf{x})$. The contrastive loss ensures this "classification dynamics" has the correct fixed point: data from class k should flow to class k.

Joint distribution and RBM connection. The class-conditional structure reveals a connection to Restricted Boltzmann Machines. Consider the joint distribution

$$p(\mathbf{x}, k) \propto \exp\left(-V_k(\mathbf{x})/k_{\mathrm{B}}T\right). \tag{36}$$

This treats k as a *one-hot hidden unit*—analogous to an RBM where the hidden layer has exactly one active unit selecting among K energy functions. The marginal over $\mathbf{x}$ is

$$p(\mathbf{x}) = \sum_k p(\mathbf{x}, k) \propto \sum_k \exp(-V_k(\mathbf{x})/k_{\mathrm{B}}T), \tag{37}$$

creating an implicit mixture model without requiring explicit hidden-unit sampling during training.

Alternating dynamics. The joint distribution (36) suggests an alternating inference procedure satisfying detailed balance:

1. *Langevin step in* $\mathbf{x}$: Given class k, evolve $\mathbf{x}$ under $-\nabla V_k$.

2. *Gibbs step in* k: Given $\mathbf{x}$, resample $k \sim p(k|\mathbf{x})$ via (35).

This provides a principled way to jointly sample configurations and class labels, unifying generation and classification in a single thermodynamic framework.

Figure 8 shows the learned class biases $-\mathbf{b}_k$, which function as attractor templates analogous to Hopfield stored patterns.

For now, the system excels at *conditional* tasks (generation and denoising given the class label) rather than pure *recognition*; contrastive fine-tuning to improve classification accuracy remains future work.

A.7   Hardware Interpretation

The class-conditional structure maps directly to hardware: the reconfiguration fabric stores K bias banks $\{\mathbf{b}_0, \ldots, \mathbf{b}_{K-1}\}$. Recognition requires evaluating K energies, which can be done in parallel using K copies of the gradient computation circuit, followed by a winner-take-all selection. Once k^* is determined, the appropriate bank is loaded ($\sim$ns DAC settling), and the T-array relaxes to produce the output. This provides a physical implementation of content-addressable memory with learned attractors.

A.8   Quantum-Classical Correspondence

The Onsager-Machlup action developed in Section 5 reveals a deeper structure. The Langevin dynamics in d dimensions corresponds to a $(d{+}1)$-dimensional classical field theory, where time t plays the role of an extra spatial dimension. The trajectory probability $P[\mathbf{x}(t)] \propto \exp(-S[\mathbf{x}])$ with action (16) defines this higher-dimensional system.

   This correspondence illuminates why learning at criticality is efficient: operating p-bits at threshold corresponds to the critical point of the $(d{+}1)$-dimensional theory, where temporal correlations become long-range. The "strong coupling in time" allows information to propagate efficiently across the time dimension, meaning that each trajectory strongly constrains model parameters. This is the physical origin of the rapid convergence observed in Figure 2—criticality maximizes the information content per trajectory, and the constraint surface becomes maximally informative.

B   Derivations

Detailed derivations of gradient estimators, time-reversal likelihoods, and relaxation bounds will be provided in an expanded version.
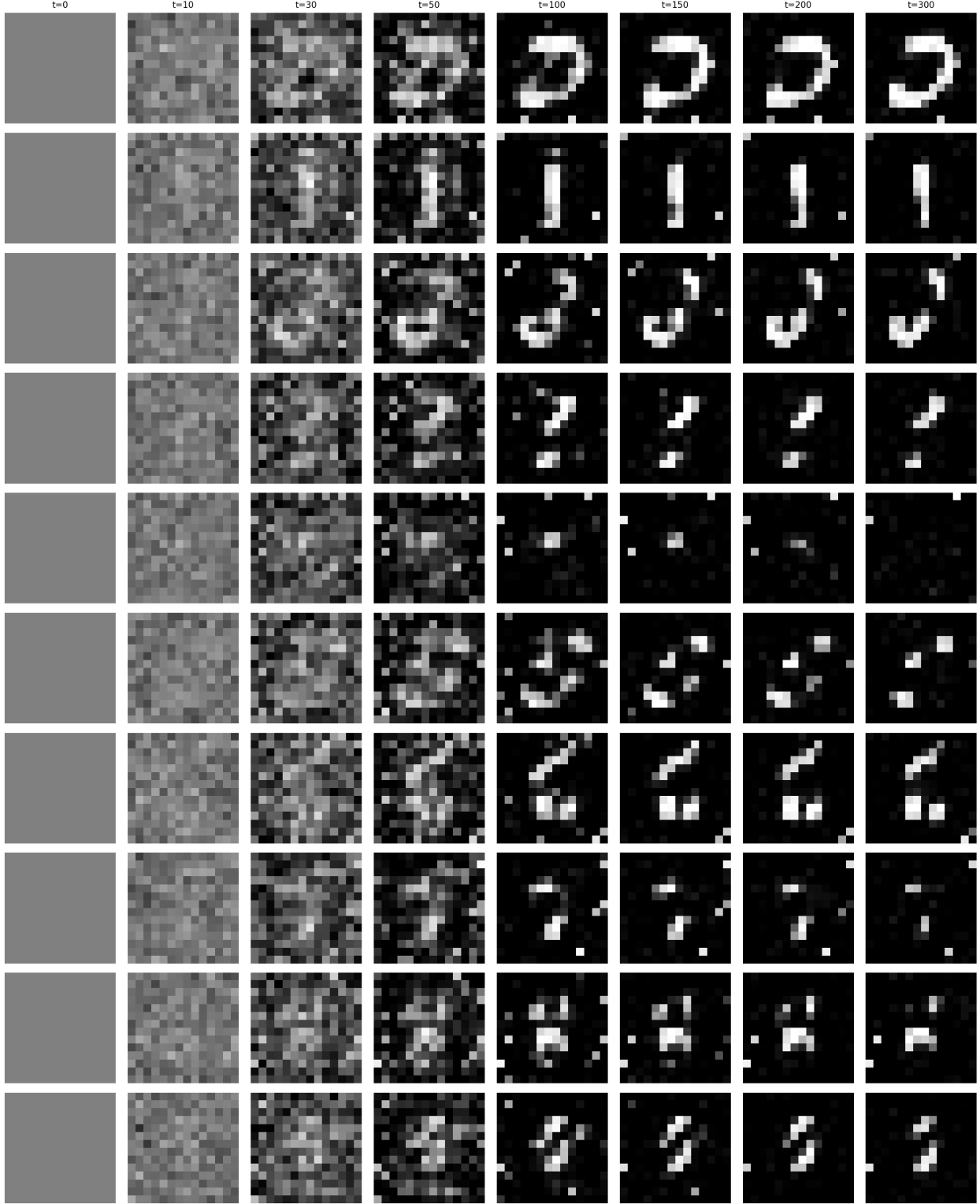
Figure 7: Attractor dynamics: each row shows evolution under a different class potential V_k. Starting from uniform gray (t=0), pixels self-organize into the corresponding digit pattern. This demonstrates that the learned biases create digit-specific energy basins.
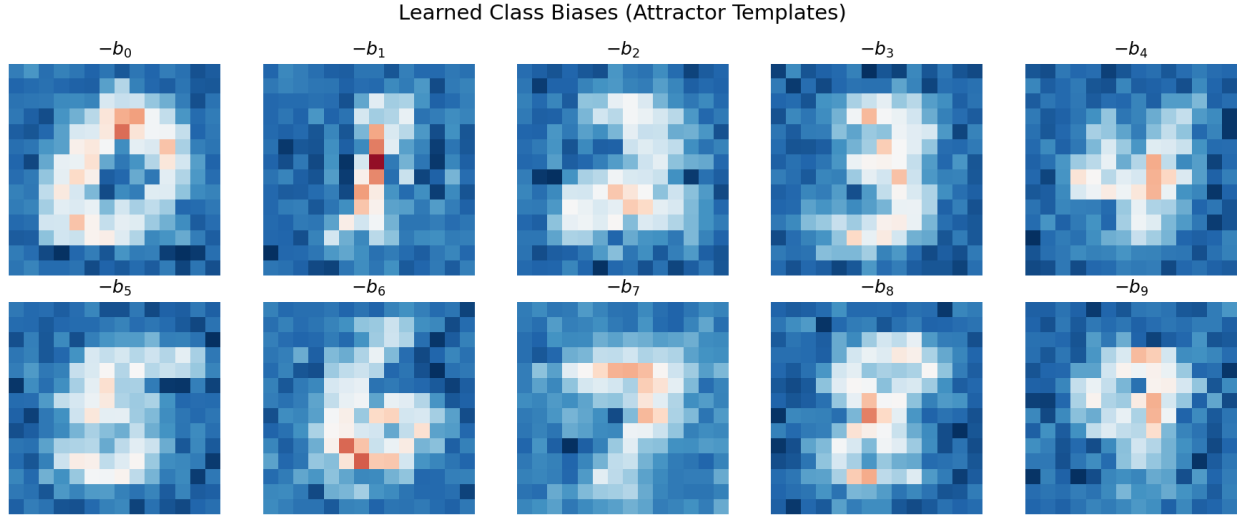
Figure 8: Learned class biases $-\mathbf{b}_k$ for digits 0–9. Red indicates pixels that the class "wants" to be positive (white), blue indicates pixels pulled toward negative (black). The biases resemble smoothed digit templates.
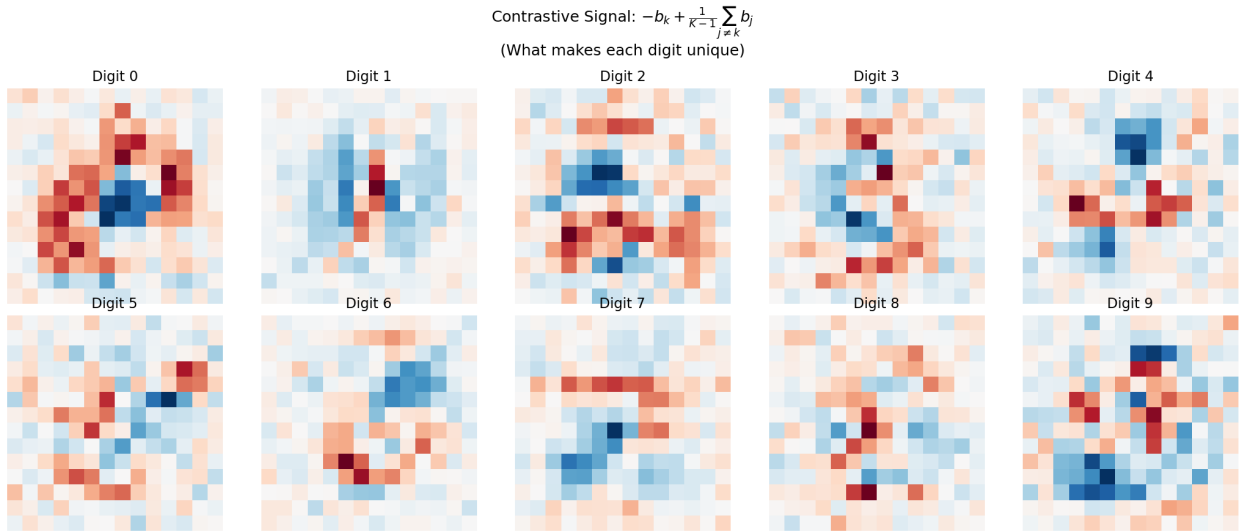

Figure 9: Contrastive signal: $-b_k + \frac{1}{K-1}\sum_{j \neq k} b_j$ for each class. This shows what makes each digit *unique*—the discriminative features that separate it from other classes. Note how digit 1 emphasizes the vertical stroke while suppressing surrounding regions.