FlowState AI: Visualizing Thermodynamic Computing

Graphical Modeling of Pbit Activity and Harmonic Pattern Generation

Alexander Le
UC Berkeley
November 2025

Note

This report explores the intersection of Extropic's thermodynamic sampling units (TSUs), probabilistic computing, and musical pattern generation through the FlowState AI project.

Executive Summary

FlowState AI is an experimental visualization and sound generation system that graphically models the stochastic behavior of probabilistic bits (pbits) from Extropic's thermodynamic sampling architecture. By leveraging three.js for real-time 3D visualization, the project creates a bridge between the abstract mathematics of probabilistic computing and the sensory experience of harmonic sound patterns. This report examines the technical foundations of TSUs, the architectural design of FlowState AI, and the emergent relationship between randomness and musical harmony.

Introduction to Thermodynamic Sampling Units

The Pbit: A Probabilistic Computing Primitive

Key Concept

A **pbit** (probabilistic bit) is a hardware implementation of a Bernoulli random variable. Unlike deterministic bits that are either 0 or 1, a pbit is programmed with a probability $p \in [0, 1]$ and produces:

$$X = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1 - p \end{cases}$$

The pbit's state fluctuates thermodynamically, making it an energy-efficient source of randomness.

In Extropic's architecture, pbits are controlled by adjusting a bias voltage that determines the probability distribution. The key innovation is that these probabilistic circuits consume dramatically less energy than deterministic circuits generating pseudo-random numbers.

Energy-Based Models and Gibbs Sampling

TSUs operate by sampling from probability distributions defined by Energy-Based Models (EBMs). For a system with variables $\mathbf{x} = (x_0, x_1, \ldots, x_n)$, the probability distribution is:

$$P(\mathbf{x}) = \frac{1}{Z} e^{-\beta E(\mathbf{x})}$$

where:

- $E(\mathbf{x})$ is the energy function
- β is the inverse temperature parameter
- $Z = \sum_{\mathbf{x}} e^{-\beta E(\mathbf{x})}$ is the partition function (normalization constant)

The energy function for binary variables with pairwise interactions is:

$$E(\mathbf{x}) = -\beta \left(\sum_i b_i x_i + \sum_{(i,j)\in\mathcal{E}} w_{ij} x_i x_j \right)$$

Analysis

Key Insight: Computing the partition function Z requires evaluating the energy function 2^N times for N binary variables, making exact probability calculations intractable for large systems. However, Gibbs sampling allows us to draw samples from $P(\mathbf{x})$ without computing Z.

Gibbs Sampling Algorithm

For a graph with nodes i connected to neighbors $\text{nb}(i)$, the Gibbs sampling update rule is:

1. Compute the effective bias for node i:

$$\gamma_i = 2\beta \left(b_i + \sum_{j\in\text{nb}(i)} w_{ij} x_j \right)$$

2. Sample x_i from a pbit with probability:

$$P(x_i = 1 \mid \mathbf{x}_{-i}) = \sigma(\gamma_i) = \frac{1}{1 + e^{-\gamma_i}}$$

3. Iterate across all nodes (or blocks of nodes in parallel)

This creates a Markov chain whose stationary distribution is $P(\mathbf{x})$.

FlowState AI: Architecture and Design

System Overview

FlowState AI is designed as a real-time visualization and sonification engine that models a grid of interconnected pbits. The system architecture consists of three primary layers:

1. **Probabilistic Computation Layer**: Simulates pbit networks using Gibbs sampling
2. **Visualization Layer**: Renders pbit states and dynamics in 3D using three.js
3. **Sonification Layer**: Maps pbit activity to harmonic frequencies and musical patterns

Pbit Grid Simulation

The simulation models an $N \times M$ grid of pbits with local connectivity. Each pbit is connected to its von Neumann neighbors (up, down, left, right), creating a bipartite graph suitable for block Gibbs sampling.

Listing 1: Pbit Grid Initialization

```
class PbitGrid {
  constructor(rows, cols, temperature = 1.0) {
    this.rows = rows;
    this.cols = cols;
    this.beta = 1.0 / temperature;

    // Initialize state matrix (0 or 1 for each pbit)
    this.states = new Array(rows).fill(0)
      .map(() => new Array(cols).fill(0)
        .map(() => Math.random() > 0.5 ? 1 : 0));

    // Bias parameters for each pbit
    this.biases = new Array(rows).fill(0)
      .map(() => new Array(cols).fill(0));

    // Edge weights (local coupling)
    this.coupling = 0.5;
  }

  // Compute effective bias for pbit at (i,j)
  computeGamma(i, j) {
    let neighborSum = 0;
    const neighbors = this.getNeighbors(i, j);

    for (let [ni, nj] of neighbors) {
      neighborSum += this.states[ni][nj] * this.coupling;
    }

    return 2 * this.beta * (this.biases[i][j] + neighborSum);
  }

  // Sample new state using sigmoid probability
  sampleState(gamma) {
    const prob = 1.0 / (1.0 + Math.exp(-gamma));
    return Math.random() < prob ? 1 : 0;
  }

  // Block Gibbs sampling update
  update() {
    // Update checkerboard pattern (even positions)
    for (let i = 0; i < this.rows; i++) {
      for (let j = (i % 2); j < this.cols; j += 2) {
        const gamma = this.computeGamma(i, j);
        this.states[i][j] = this.sampleState(gamma);
      }
    }

```

```
    // Update odd positions
    for (let i = 0; i < this.rows; i++) {
      for (let j = ((i + 1) % 2); j < this.cols; j += 2) {
        const gamma = this.computeGamma(i, j);
        this.states[i][j] = this.sampleState(gamma);
      }
    }
  }
}
```

Three.js Visualization Architecture

The visualization represents each pbit as a particle in 3D space. The visual properties encode the pbit's state and dynamics:

- **Position**: Mapped to grid coordinates $(i, j) \to (x, y, z)$
- **Color**: Interpolates between blue (state $= 0$) and red (state $= 1$)
- **Size/Intensity**: Encodes the bias probability $\sigma(\gamma_i)$
- **Motion**: Particles oscillate based on local energy gradients

Listing 2: Three.js Pbit Particle System

```
class PbitVisualizer {
  constructor(pbitGrid) {
    this.grid = pbitGrid;
    this.scene = new THREE.Scene();
    this.camera = new THREE.PerspectiveCamera(
      75, window.innerWidth / window.innerHeight, 0.1, 1000
    );
    this.renderer = new THREE.WebGLRenderer({ antialias: true });

    this.initParticles();
    this.initConnections();
  }

  initParticles() {
    const geometry = new THREE.SphereGeometry(0.2, 16, 16);
    this.particles = [];

    for (let i = 0; i < this.grid.rows; i++) {
      for (let j = 0; j < this.grid.cols; j++) {
        const material = new THREE.MeshPhongMaterial({
          color: 0x0000ff,
          emissive: 0x0000ff,
          emissiveIntensity: 0.2
        });

        const particle = new THREE.Mesh(geometry, material);
        particle.position.set(
          j * 2 - this.grid.cols,
```

```
        i * 2 - this.grid.rows,
        0
      );

      this.particles.push(particle);
      this.scene.add(particle);
    }
  }
}

updateVisualization() {
  let idx = 0;
  for (let i = 0; i < this.grid.rows; i++) {
    for (let j = 0; j < this.grid.cols; j++) {
      const state = this.grid.states[i][j];
      const gamma = this.grid.computeGamma(i, j);
      const prob = 1.0 / (1.0 + Math.exp(-gamma));

      // Color interpolation: blue -> red
      const color = new THREE.Color().setHSL(
        0.66 * (1 - state), // Hue: blue to red
        1.0,                // Saturation
        0.3 + 0.4 * prob // Lightness based on probability
      );

      this.particles[idx].material.color = color;
      this.particles[idx].material.emissive = color;
      this.particles[idx].material.emissiveIntensity = 0.5 * prob;

      // Scale based on state probability
      const scale = 0.7 + 0.6 * prob;
      this.particles[idx].scale.set(scale, scale, scale);

      idx++;
    }
  }
}

animate() {
  requestAnimationFrame(() => this.animate());

  this.grid.update();
  this.updateVisualization();

  // Gentle camera rotation
  this.scene.rotation.y += 0.001;

  this.renderer.render(this.scene, this.camera);
}
}
```

Randomness and Harmonic Patterns in Music

The Paradox of Musical Randomness

Music exists at a fascinating intersection of order and chaos. While completely deterministic sequences can sound mechanical and lifeless, pure randomness creates cacophony. The art of musical composition lies in finding the "edge of chaos" where pattern and unpredictability coexist.

Key Concept

Harmonic Series Foundation
The harmonic series defines the natural resonances of vibrating systems:

$$f_n = n \cdot f_0, \quad n = 1, 2, 3, \ldots$$

where f_0 is the fundamental frequency. Western music theory is built on frequency ratios:

- Octave: $2:1$ ratio
- Perfect fifth: $3:2$ ratio
- Major third: $5:4$ ratio
- Minor third: $6:5$ ratio

Stochastic Harmony: Mapping Pbits to Musical Notes

FlowState AI maps pbit activity to musical frequencies using a probabilistic harmonic framework:

1. **Base Frequency Assignment**: Each pbit (i, j) is assigned a base frequency from a musical scale:
$$f_{ij} = f_0 \cdot 2^{s_{ij}/12}$$
where $s_{ij} \in \{0, 2, 4, 5, 7, 9, 11\}$ represents scale degrees (e.g., C major scale).

2. **State-Dependent Activation**: The pbit state determines if the frequency is active:
$$A_{ij}(t) = \begin{cases} 1 & \text{if } x_{ij}(t) = 1 \\ 0 & \text{if } x_{ij}(t) = 0 \end{cases}$$

3. **Probability-Weighted Amplitude**: The sound amplitude is modulated by the pbit's activation probability:
$$a_{ij}(t) = A_{ij}(t) \cdot \sigma(\gamma_{ij}(t))$$

4. **Composite Sound Wave**: The total audio signal is:
$$S(t) = \sum_{i,j} a_{ij}(t) \cdot \sin(2\pi f_{ij} t + \phi_{ij})$$

Listing 3: Audio Synthesis from Pbit States

```
class PbitSonifier {
  constructor(pbitGrid, audioContext) {
    this.grid = pbitGrid;
    this.ctx = audioContext;
    this.oscillators = [];
    this.gainNodes = [];

    // C Major pentatonic scale (in semitones from C4)
    this.scale = [0, 2, 4, 7, 9, 12, 14, 16];
    this.baseFreq = 261.63; // C4

    this.initAudio();
  }

  initAudio() {
    for (let i = 0; i < this.grid.rows; i++) {
      for (let j = 0; j < this.grid.cols; j++) {
        const osc = this.ctx.createOscillator();
        const gain = this.ctx.createGain();

        // Assign frequency from scale
        const scaleIdx = (i * this.grid.cols + j) % this.scale.length;
        const semitones = this.scale[scaleIdx];
        osc.frequency.value = this.baseFreq * Math.pow(2, semitones / 12);

        osc.type = 'sine';
        osc.connect(gain);
        gain.connect(this.ctx.destination);

        gain.gain.value = 0; // Start silent
        osc.start();

        this.oscillators.push(osc);
        this.gainNodes.push(gain);
      }
    }
  }

  updateAudio() {
    let idx = 0;
    for (let i = 0; i < this.grid.rows; i++) {
      for (let j = 0; j < this.grid.cols; j++) {
        const state = this.grid.states[i][j];
        const gamma = this.grid.computeGamma(i, j);
        const prob = 1.0 / (1.0 + Math.exp(-gamma));

        // Amplitude = state * probability
        const amplitude = state * prob * 0.05; // Scale for mixing

        // Smooth gain changes to avoid clicks
        this.gainNodes[idx].gain.linearRampToValueAtTime(
          amplitude,
          this.ctx.currentTime + 0.1
```

```
        );

        idx++;
      }
    }
  }
}
```

Emergent Musical Patterns

The coupling between pbits creates emergent harmonic patterns through several mechanisms:

Analysis

Pattern Formation Mechanisms

1. Spatial Correlation → Harmonic Clustering

When coupling weights w_{ij} are positive, neighboring pbits tend to synchronize states. This creates spatial regions where similar frequencies are active simultaneously, forming chords and harmonic clusters.

For a cluster of k activated pbits with frequencies $\{f_1, f_2, \ldots, f_k\}$, the harmonic content depends on their frequency ratios. If the ratios approximate simple fractions (e.g., $f_2/f_1 \approx 3/2$), the result is consonant harmony.

2. Temporal Dynamics → Rhythmic Patterns

The Gibbs sampling creates temporal correlations. The autocorrelation function for pbit i is:

$$R_i(\tau) = \langle x_i(t) x_i(t+\tau) \rangle - \langle x_i(t) \rangle^2$$

Non-zero autocorrelation creates rhythmic patterns as pbits flip states with characteristic time scales determined by the temperature β and local energy landscape.

3. Critical Dynamics → Complex Textures

At critical temperatures where $\beta \approx \beta_c$ (near phase transitions), the system exhibits:

- Long-range correlations
- Power-law distributed cluster sizes
- $1/f$ noise (pink noise) in temporal dynamics

This critical regime produces the richest musical textures, balancing order and randomness.

Mathematical Analysis of Harmonic Emergence

Correlation Functions and Musical Consonance

The spatial correlation between pbits directly influences harmonic consonance. For two pbits at positions $\mathbf{r}_i$ and $\mathbf{r}_j$, the correlation is:

$$C(\mathbf{r}_i, \mathbf{r}_j) = \langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle$$

Strong positive correlation ($C \approx 1$) means the pbits are likely to be in the same state, causing their associated frequencies to sound together frequently, which the ear interprets as a harmonic relationship.

Spectral Analysis of Pbit Ensembles

The power spectral density of the composite signal reveals the harmonic structure:

$$P(f) = \left| \int_{-\infty}^{\infty} S(t) e^{-2\pi i f t} dt \right|^2$$

For a TSU with N pbits and frequencies $\{f_1, \ldots, f_N\}$, the expected spectrum is:

$$\langle P(f) \rangle = \sum_{k=1}^{N} \langle a_k^2 \rangle \delta(f - f_k) + \text{cross terms}$$

where $\langle a_k^2 \rangle = \langle \sigma(\gamma_k) \rangle$ is the time-averaged activation probability.

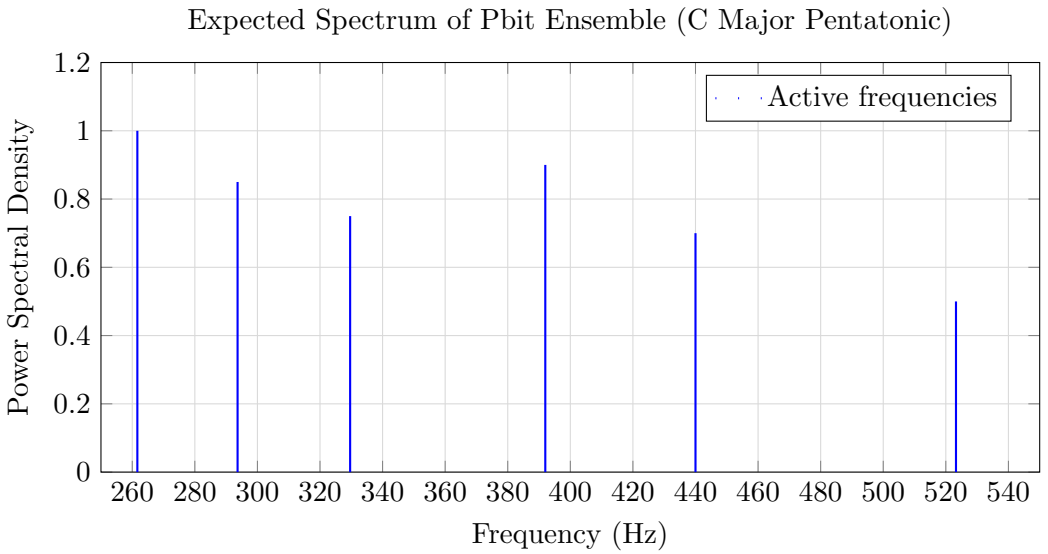

Figure 1: Power spectrum shows peaks at scale frequencies with heights determined by pbit activation probabilities

Temperature and Musical Expressiveness

The temperature parameter β^{-1} controls the "expressiveness" of the generated music:

- **Low temperature** ($\beta \gg 1$): System settles into low-energy configurations. Music becomes more deterministic with longer-held chords.
- **High temperature** ($\beta \ll 1$): Pbits flip rapidly and independently. Music becomes more stochastic with frequent note changes.
- **Critical temperature** ($\beta \approx \beta_c$): Maximum structural complexity with balanced order and randomness.

The entropy of the pbit distribution measures this:

$$H = -\sum_{\mathbf{x}} P(\mathbf{x}) \log P(\mathbf{x}) = \langle E(\mathbf{x}) \rangle / T + \log Z$$

Maximum entropy (high randomness) occurs at high temperatures, while minimum entropy (high order) occurs at low temperatures.

Implementation Details and Optimizations

Performance Considerations

Real-time visualization and audio synthesis of large pbit grids requires careful optimization:

1. **Web Workers for Simulation**: Run Gibbs sampling in a separate thread to avoid blocking the rendering loop.
2. **Instanced Rendering**: Use three.js InstancedMesh to render thousands of particles efficiently.
3. **Audio Buffer Management**: Pre-compute oscillator frequencies and use gain scheduling to minimize audio thread overhead.
4. **Spatial Indexing**: Use quadtree structures for efficient neighbor lookup in large grids.

Interactive Controls

FlowState AI exposes several parameters for real-time exploration:

Listing 4: Interactive Parameter Control

```
class FlowStateController {
  constructor(visualizer, sonifier) {
    this.viz = visualizer;
    this.audio = sonifier;
    this.params = {
      temperature: 1.0,
      coupling: 0.5,
      baseFrequency: 261.63,
      scale: 'major_pentatonic'
    };
  }

  setTemperature(temp) {
    this.params.temperature = temp;
    this.viz.grid.beta = 1.0 / temp;

    // Higher temp -> faster update rate for audio responsiveness
    this.updateRate = Math.min(60, 20 + temp * 10);
  }

  setCoupling(coupling) {
    this.params.coupling = coupling;
    this.viz.grid.coupling = coupling;
```

```
  }

  perturbBiases(strength) {
    // Add random perturbations to create musical variation
    for (let i = 0; i < this.viz.grid.rows; i++) {
      for (let j = 0; j < this.viz.grid.cols; j++) {
        this.viz.grid.biases[i][j] +=
          (Math.random() - 0.5) * 2 * strength;
      }
    }
  }

  applyPattern(patternType) {
    // Set biases to create specific energy landscapes
    switch(patternType) {
      case 'waves':
        this.createWavePattern();
        break;
      case 'checkerboard':
        this.createCheckerboardPattern();
        break;
      case 'random':
        this.createRandomPattern();
        break;
    }
  }

  createWavePattern() {
    for (let i = 0; i < this.viz.grid.rows; i++) {
      for (let j = 0; j < this.viz.grid.cols; j++) {
        this.viz.grid.biases[i][j] =
          Math.sin(2 * Math.PI * i / this.viz.grid.rows) * 2.0;
      }
    }
  }
}
```

Results and Observations

Qualitative Musical Characteristics

Testing FlowState AI with various parameter configurations reveals distinct musical "personalities":

1. **Ambient Textures** ($\beta = 0.5$, weak coupling): Rapidly changing, ethereal soundscapes with minimal harmonic structure. Resembles granular synthesis.

2. **Harmonic Drones** ($\beta = 2.0$, strong positive coupling): Stable, sustained chords that slowly evolve. Creates meditative, minimalist music.

3. **Generative Melodies** ($\beta = 1.0$, negative coupling): Anti-correlated pbits create alternating note patterns, forming melodic sequences.

4. **Complex Polyrhythms** (varying β across grid): Different regions update at different rates, creating layered rhythmic textures.

Visual-Audio Coherence

The tight coupling between visual and audio representation creates a synesthetic experience:

- Brightness correlates with volume (both driven by $\sigma(\gamma)$)
- Color correlates with harmonic activity (red regions = active notes)
- Spatial patterns (clusters, waves) have direct audible counterparts
- Temporal dynamics visible as "flows" of color

This multi-modal feedback allows intuitive understanding of the underlying probabilistic dynamics.

Future Directions

Scaling to Larger Grids

Extropic's Z1 TSU will have hundreds of thousands of pbits. FlowState AI could be extended to:

- Hierarchical visualization (zoom levels showing different scales of structure)
- Multi-octave mapping (different grid regions to different octaves)
- Timbral variation (pbit clusters mapped to different instrument sounds)

Learning Musical Structure

Current pbit parameters are manually designed. Future work could:

1. **Train on musical corpora**: Learn bias and coupling parameters that reproduce statistical patterns from existing music.
2. **Interactive evolution**: Allow users to select preferred outputs, using evolutionary algorithms to refine parameters.
3. **Denoising music models**: Extend Extropic's DTM framework to learn hierarchical musical structure.

Real Hardware Integration

When physical TSUs become available:

- Replace simulation with actual hardware sampling
- Exploit true thermodynamic noise for authentic randomness
- Measure energy efficiency of music generation (joules per note)
- Build dedicated musical instruments based on TSU architecture

Philosophical Implications

Randomness as Creative Force

FlowState AI demonstrates that randomness, when properly constrained by physical dynamics (energy functions, coupling, temperature), can be a generative force for beauty. The key insight is that:

> *Creativity emerges not from pure randomness or pure determinism, but from the complex dynamics at the boundary between order and chaos. – Arne van Oosterom*

This mirrors natural processes:

- Evolution: Random mutations + selection pressure
- Jazz improvisation: Musical grammar + spontaneous variation
- Natural language: Statistical regularities + creative expression

Thermodynamics of Aesthetics

The connection between thermodynamic sampling and musical harmony suggests a deeper relationship between physical entropy and aesthetic information. Music theory's preference for simple frequency ratios (consonance) aligns with the tendency of thermodynamic systems to minimize energy.

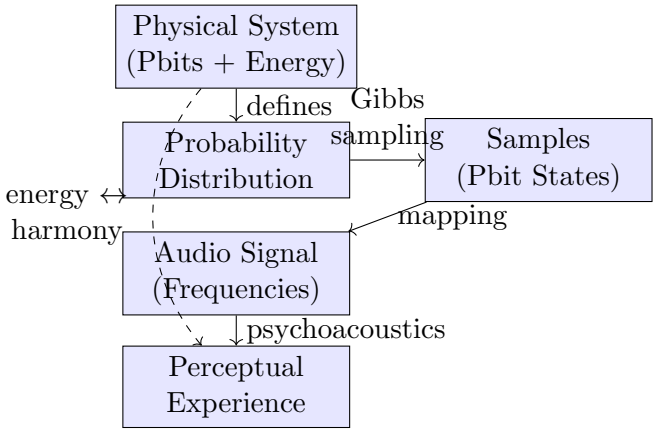

Figure 2: Information flow from physics to perception in FlowState AI

Conclusion

FlowState AI bridges the abstract mathematics of thermodynamic computing with the visceral experience of music and visual art. By graphically modeling pbit activity through three.js and mapping stochastic dynamics to harmonic structures, the project reveals emergent patterns that are simultaneously random and musical.

The key contributions are:

1. **Visualization framework**: Real-time 3D rendering of pbit networks that makes probabilistic computation visible and tangible.

2. **Sonification architecture**: Principled mapping from pbit states to musical frequencies that preserves spatial and temporal correlations.

3. **Emergent musicality**: Demonstration that simple local rules (Gibbs sampling) can generate complex, aesthetically pleasing harmonic patterns.

4. **Interactive exploration**: Tools for manipulating temperature, coupling, and biases to navigate the space of possible musical textures.

As Extropic's TSU technology matures and scales to millions of pbits, systems like FlowState AI could evolve into new forms of musical instruments and generative art tools. The fusion of thermodynamic physics, probabilistic computing, and aesthetic expression opens exciting possibilities for both creative practice and scientific understanding.

The randomness in harmonic patterns is not noise to be eliminated, but rather the signature of a complex system exploring its possibility space—much like the improvisation of a jazz musician, the evolution of a species, or the creative process itself.

Note

Access FlowState AI
Project repository and live demos will be available at:
– live demo: `https://bit.ly/3LKBsM0`
– main repo: `https://github.com/GeneticAlgorithms/THRML-flowstateAI/tree/main`
Built with: three.js, Web Audio API, and thermodynamic computing principles from Extropic.