

ALAN 2.x: A Spectral-Phase Cognitive Architecture Integrating Koopman Operators, Lyapunov-Stabilized Oscillators, and Kernel-Efficient Concept Graphs

You + Elegans

May 2025

Abstract

We present ALAN 2.x, a next-generation cognitive system combining spectral Koopman operators, Lyapunov-constrained oscillator networks (Banksy), and structured concept graphs. This architecture unifies phase-based memory, eigenfunction-based semantics, and scalable kernel approximations to enable persistent, interpretable, and dynamically evolving reasoning. We describe the system’s formal structure, present implementation strategies including spectral kernel compression and asynchronous concept resonance, and propose pathways toward hybrid quantum-kernel expansion.

Contents

1	Introduction	2
2	System Components	3
2.1	2.1 Large Concept Network (LCN)	3
2.2	2.2 Banksy Oscillator Core	3
2.3	2.3 Koopman Morphic Engine	3
3	Spectral Kernel and Quantum Extensions	4
4	Spectral Kernel and Quantum Extensions	4
4.1	3.1 Spectral Kernel Compression	4
4.2	3.2 Asymmetric and Structured Kernels	4
4.3	3.3 NTK-Guided Spectral Filtering	4
4.4	3.4 Quantum Kernel Embeddings	4
5	Lyapunov-Stable Learning	5
6	Lyapunov-Stable Learning	5
6.1	4.1 Lemorphic Stability Coupling	5
6.2	4.2 Adaptive Coupling Dynamics	5
6.3	4.3 Gradient-Based Plasticity	5
7	Evaluation Dashboard	5
8	Evaluation Dashboard	5
8.1	5.1 Core Metrics Tracked	5
8.2	5.2 Dashboard Views and Alerts	6
8.3	5.3 Use Cases	6
9	Implementation Roadmap	6

10 Implementation Roadmap	6
10.1 6.1 Phase I — Offline Simulation Prototype	6
10.2 6.2 Phase II — Streaming Cognitive Core	7
10.3 6.3 Phase III — Hybrid Extensions	7
11 Discussion and Future Work	7
12 Discussion and Future Work	7
12.1 7.1 Open Challenges	7
12.2 7.2 Future Research Directions	7
A Appendix A: Koopman Morphic Memory Formalism	8
B Appendix B: Python Implementation Snippets	9
C Appendix C: Diagrams and Dashboard Visuals	10

1 Introduction

space1em ALAN 2.x represents a significant leap in cognitive architecture, synthesizing ideas from nonlinear dynamics, spectral theory, memory systems, and symbolic-graph reasoning into a cohesive framework. Rather than predicting tokens or symbols in a feedforward loop, ALAN 2.x evolves high-level semantic states across a dynamic memory space. It operates over a Large Concept Network (LCN), whose nodes represent stable, composable conceptual units with explicit temporal and topological semantics.

The core innovations include:

- A **Koopman Morphic Engine**, where concepts are eigenfunctions of a learned Koopman operator \mathcal{K} , encoding semantic flow in spectral space.
- A **Banksy Oscillator Core**, leveraging Lyapunov-stable phase oscillators for binding, synchrony, and dynamical memory.
- A **Spectral Kernel Learning Layer** for efficient, adaptive representation using low-rank kernel eigenbases and random feature expansions.
- Integration with **conceptual graph memory** via spectral attention, narrative walk decoding, and morphic resonance kernels.

ALAN 2.x aims to unify symbolic reasoning, dynamic memory, and spectral generalization—opening the door to interpretable, persistent, and concept-driven intelligence. It is intended as a foundational system for future cognitive agents capable of fluid long-term reasoning, multimodal abstraction, and persistent conceptual growth.

2 System Components

2.1 2.1 Large Concept Network (LCN)

The LCN is the high-level symbolic substrate of ALAN 2.x. It encodes concepts as graph nodes, each with persistent embeddings and dynamic metadata (e.g., frequency of use, entropy, relevance). Edges capture semantic relations, inference paths, and co-activation traces.

Key Modules:

- *Sparse Coupling Layer*: Computes TopK attention between oscillator queries and concept node keys.
- *Contextual Partitioning*: Selects subgraphs for focus, based on conceptual proximity, usage history, or task context.
- *Advanced Recall Solver*: Supports exact-match, associative, multi-hop, and constraint-based queries.
- *Incremental Update Engine*: Performs local SGD on embeddings using contrastive or relational loss.
- *Hierarchical Memory Organizer*: Maintains nested clusters of concepts to enable logarithmic-time search.

2.2 2.2 Banksy Oscillator Core

Each concept node has a companion oscillator $\theta_i(t)$, updated by phase coupling and energy minimization. The update rule is:

$$\theta_i(t+1) = \theta_i(t) + A_i \sin(\tilde{\theta}_i(t)) + \alpha_g \tilde{\theta}_i(t) + \alpha_k [K\theta(t)]_i \quad (1)$$

where $\tilde{\theta}_i = \theta_i - S(t)$, the global synchrony anchor.

A Lyapunov function governs stability:

$$V(\theta) = -\frac{1}{2} \sum_{i,j} K_{ij} \cos(\theta_i - \theta_j) \quad (2)$$

Phase coherence encodes memory, enabling multiscale binding via nested synchrony bands.

2.3 2.3 Koopman Morphic Engine

This module evolves concepts in a spectral domain. Concepts are encoded as Koopman eigenfunctions:

$$\mathcal{K}\psi_i = \lambda_i \psi_i \quad (3)$$

Temporal evolution is linearized in this space, allowing stable extrapolation and recall.

Components:

- *Spectral Distance*: $d(\psi_i, \psi_j) = \|\psi_i - \psi_j\|_{L^2}$
- *Morphic Resonance Kernel*: $\kappa(\psi_i, \psi_j; \tau) = \int \psi_i(t) \psi_j(t + \tau) dt$
- *Lemorphic Loss*: Koopman eigenfunction fidelity + smoothness penalty:

$$\mathcal{L}_{\text{morphic}} = \sum_i \|\mathcal{K}\psi_i - \lambda_i \psi_i\|^2 + \alpha \|\nabla_{\theta} \psi_i\|^2$$

- *Narrative Flow*: Concept generation as a spectral walk: $\psi_{t+1} = \arg \max_j \langle \psi_t, \psi_j \rangle$

This engine turns spectral structure into temporal cognition.

3 Spectral Kernel and Quantum Extensions

4 Spectral Kernel and Quantum Extensions

4.1 3.1 Spectral Kernel Compression

To improve efficiency and generalization, ALAN 2.x applies low-rank spectral kernel methods for encoding similarity between concepts. Using the eigenfunctions of the kernel operator, we approximate mappings in high-dimensional or infinite RKHS:

$$k(x, x') = \sum_{i=1}^r \lambda_i \phi_i(x) \phi_i(x') \quad (4)$$

Only the top r modes are retained, balancing expressiveness and efficiency.

Dynamic Eigenbasis Update: We employ the SPEED method (Spectral Eigenfunction Decomposition) to adaptively update the kernel basis:

- Incrementally updates leading eigenfunctions from streaming data.
- Prunes noise-dominated directions to enhance robustness.
- Enables continual learning with sublinear memory growth.

4.2 3.2 Asymmetric and Structured Kernels

Classical RFF approximates shift-invariant kernels using Bochner’s theorem. ALAN 2.x extends this via:

- **Structured Sampling:** Spherical-radial quadrature for precise kernel approximations with fewer features.
- **AsK-RFF:** Generalization to asymmetric kernels using complex-valued Fourier transforms. Allows for non-metric similarity in directed graphs or logic-driven links.

4.3 3.3 NTK-Guided Spectral Filtering

Neural Tangent Kernel (NTK) theory informs how networks learn different frequencies. ALAN 2.x uses:

- *Bias-Free Linear Filter Layer* after Fourier embedding.
- Learns to amplify or suppress features adaptively per frequency.
- Preserves stability across iterations, mitigates spectral aliasing.

4.4 3.4 Quantum Kernel Embeddings

Quantum kernels define similarities via inner products in Hilbert spaces induced by quantum feature maps:

$$k(x, x') = |\langle \psi(x), \psi(x') \rangle|^2 \quad (5)$$

Quantum processors can efficiently sample from spectral distributions that are classically intractable. This opens:

- **Efficient RFF on quantum circuits** for shift-invariant kernels.
- **Exotic kernels** inspired by quantum entanglement or non-classical correlations.
- **Projected quantum kernels:** high-dimensional embeddings with classical post-processing.

ALAN 2.x anticipates quantum-enhanced cognition via structured, spectral embeddings and robust feature expansions.

5 Lyapunov-Stable Learning

6 Lyapunov-Stable Learning

The Banksy oscillator core maintains dynamical stability through a Lyapunov-based potential energy landscape:

$$V(\theta) = -\frac{1}{2} \sum_{i,j} K_{ij} \cos(\theta_i - \theta_j) \quad (6)$$

This ensures synchronized clusters converge to minima in phase space, enabling memory and coherent binding.

6.1 4.1 Lemorphic Stability Coupling

To unify structural stability with semantic coherence, we define a total loss:

$$\mathcal{L}_{\text{total}} = V_{\text{Banksy}} + \beta \cdot \mathcal{L}_{\text{morphic}} \quad (7)$$

This glues phase-based attractors with spectral purity in the Koopman engine.

6.2 4.2 Adaptive Coupling Dynamics

The oscillator coupling matrix K can be learned via Hebbian-like rules:

$$\dot{K}_{ij} = \epsilon (\cos(\theta_j - \theta_i) - K_{ij}) \quad (8)$$

This adapts network topology to reinforce synchrony among co-active concepts, forming functional subnetworks.

6.3 4.3 Gradient-Based Plasticity

To fine-tune concept dynamics and reduce instability, we optimize:

$$\frac{d}{dt} W = -\nabla_W \mathcal{L}_{\text{total}} \quad (9)$$

where W includes K , oscillator parameters A_i , or concept embeddings. This results in plasticity driven by energy minimization and semantic resonance.

Together, Lyapunov-stabilized dynamics and morphic spectral learning create a globally convergent yet semantically expressive cognitive substrate.

7 Evaluation Dashboard

8 Evaluation Dashboard

To monitor, diagnose, and interpret ALAN 2.x’s cognitive processes, we propose a real-time evaluation dashboard. It integrates key metrics from the Banksy phase layer, Koopman morphic engine, and LCN concept graph.

8.1 5.1 Core Metrics Tracked

- **Phase Synchrony Score:** Mean pairwise phase coherence:

$$\rho(t) = \frac{1}{N(N-1)} \sum_{i \neq j} \cos(\theta_i(t) - \theta_j(t))$$

- **Lyapunov Energy Decay:** Tracks convergence of $V(\theta)$ over time.
- **Spectral Entropy:** Shannon entropy over Koopman eigenvalue magnitudes:

$$H = - \sum_i p_i \log p_i, \quad p_i = \frac{|\lambda_i|}{\sum_j |\lambda_j|}$$

- **Kernel Rank Estimate:** Number of active kernel eigenfunctions retained by SPEED.
- **Resonance Matrix Heatmap:** Visualizes $\kappa(\psi_i, \psi_j; \tau)$ for selected τ .
- **Narrative Flow Coherence:** Average spectral transition overlap:

$$\mathbb{E}_t[\langle \psi_t, \psi_{t+1} \rangle]$$

8.2 5.2 Dashboard Views and Alerts

- **Oscillator Health View:** Stability plots, desynchronization alerts, phase drift indicators.
- **Spectral Mode Tracker:** Real-time Koopman eigenvalue drift and activation heatmaps.
- **Entropy Anomaly Alerts:** Triggers when concept spectrum collapses or spikes (e.g., forgetting, overfitting).
- **Memory Load Monitor:** Tracks node/edge growth, pruning events, and recall rates.

8.3 5.3 Use Cases

- **Debugging:** Visualize oscillatory breakdowns, spectral mode degeneration, or anomalous recalls.
- **Training:** Optimize morphic + Lyapunov losses and verify convergence.
- **Interpretability:** Explain why a narrative was generated by tracing its spectral path.
- **Evaluation:** Compare cognitive health across checkpoints or configurations.

The dashboard transforms ALAN 2.x from a black box into a living cognitive field with interpretable signals.

9 Implementation Roadmap

10 Implementation Roadmap

This roadmap outlines a three-phase deployment strategy for ALAN 2.x, from prototype simulation to full hybrid cognitive integration.

10.1 6.1 Phase I — Offline Simulation Prototype

- **Concept Trajectories:** Generate synthetic or curated sequences of concept embeddings.
- **Koopman Approximation:** Use Extended Dynamic Mode Decomposition (EDMD) to compute Koopman eigenfunctions.
- **Narrative Flow Tests:** Simulate walks in Koopman mode space and validate coherence.
- **Banksy Dynamics:** Evaluate oscillator phase convergence, synchrony metrics, and Lyapunov loss.
- **Diagnostic Visuals:** Plot phase trajectories, spectral heatmaps, and concept recall distributions.

10.2 6.2 Phase II — Streaming Cognitive Core

- **Incremental DMD:** Replace EDMD with streaming Koopman updates using SPEED or similar.
- **Concept Graph Coupling:** Integrate live concept insertions into Banksy + Koopman flow.
- **Adaptive Pruning:** Use resonance metrics and entropy thresholds to prune or compress nodes.
- **Spectral-Guided Recall:** Replace transformer-based decoders with spectral narrative walks.

10.3 6.3 Phase III — Hybrid Extensions

- **FKEA Integration:** Use Fourier-based entropy approximations to track distributional diversity.
- **Quantum Kernel Support:** Experiment with projected quantum circuits for Koopman-RFF fusion.
- **Fractal Graph Memory:** Extend LCN to recursively-nested hierarchies for scalable abstraction.
- **Neuromorphic Implementation:** Explore spintronic or photonic oscillator substrates for ALAN.

This roadmap sets the stage for ALAN 2.x to evolve from simulation to real-time, cross-modal, hybrid cognition.

11 Discussion and Future Work

12 Discussion and Future Work

ALAN 2.x combines spectral, dynamical, and symbolic cognition into a single operating framework. While its modules are theoretically grounded and modularly designed, several open research directions remain.

12.1 7.1 Open Challenges

- **Stochastic Koopman Extensions:** Real-world concept graphs are noisy. Extending the Koopman operator to stochastic or data-driven variants (e.g., kernel Koopman, delay embeddings) is an ongoing direction.
- **Memory Saturation:** Mechanisms for forgetting or merging concept nodes must be developed to prevent overload. Spectral entropy and resonance decay could be used to triage concepts.
- **Stability Guarantees:** While the Banksy Lyapunov energy offers convergence guarantees, full-system stability (oscillators + spectral + graph) is not yet fully characterized.
- **Interpretability of Spectral Modes:** Methods to map Koopman eigenfunctions back to interpretable concept subgraphs or sequences remain an active need.

12.2 7.2 Future Research Directions

- **Multi-Agent Synchrony:** Investigate morphic resonance across distributed ALAN agents—can narrative flows synchronize?
- **Fractal and Nested Memory:** Extend the LCN structure to recursive graphs with shared subpatterns, allowing hierarchical and compositional reasoning.
- **Quantum Koopman Prototypes:** Simulate quantum Koopman approximations using Qiskit or PennyLane, validating small-scale quantum spectral kernels.
- **Autonomous Concept Evolution:** Let the system grow its concept graph via internally generated narrative walks, influenced by entropy or curiosity signals.

ALAN 2.x offers a fertile foundation—capable of uniting memory, rhythm, logic, and spectrum into a coherent model of emergent intelligence. The next frontier is not just cognition, but cognition that learns to restructure itself.

A Appendix A: Koopman Morphic Memory Formalism

Koopman Morphic Memory Formalism

A.1 Koopman Operator Definition

Let the latent state of the system be $\theta(t) \in \mathbb{R}^n$. Define an observable $f : \mathbb{R}^n \rightarrow \mathbb{C}$.

The Koopman operator \mathcal{K} evolves observables rather than states:

$$(\mathcal{K}f)(\theta) = f(\Phi^t(\theta)) \quad (10)$$

where Φ^t is the underlying dynamical flow. \mathcal{K} is linear (even if Φ^t is not).

A.2 Eigenfunction Structure

A Koopman eigenfunction ψ_i satisfies:

$$\mathcal{K}\psi_i = \lambda_i\psi_i \quad (11)$$

This implies $\psi_i(\theta(t)) = e^{\lambda_i t}\psi_i(\theta(0))$.

A.3 Concept Encoding

Each concept is represented as a dominant eigenfunction ψ_i , forming a set:

$$\mathcal{C} = \{\psi_1, \psi_2, \dots, \psi_m\}$$

Compositional concepts are formed by:

$$\psi_{i \wedge j} = \psi_i \cdot \psi_j \quad (12)$$

A.4 Spectral Distance and Resonance

Define an L^2 distance:

$$d(\psi_i, \psi_j) = \left(\int |\psi_i(\theta) - \psi_j(\theta)|^2 d\mu(\theta) \right)^{1/2} \quad (13)$$

Define a resonance kernel over time lag τ :

$$\kappa(\psi_i, \psi_j; \tau) = \int \psi_i(\theta(t)) \cdot \overline{\psi_j(\theta(t + \tau))} dt \quad (14)$$

A.5 Lemorphic Loss

The morphic loss function aligns observed and predicted spectral flow:

$$\mathcal{L}_{\text{morphic}} = \sum_i \|\mathcal{K}\psi_i - \lambda_i\psi_i\|^2 + \alpha \sum_i \|\nabla_{\theta}\psi_i\|^2 \quad (15)$$

This enforces spectral purity and smooth semantic fields.

A.6 Schrödinger Extension (Optional)

If $\mathcal{K} = e^{i\hat{H}t}$, the Koopman engine admits a Schrödinger form:

$$i \frac{\partial \psi}{\partial t} = \hat{H}\psi \quad (16)$$

This allows for interference-based concept evolution and dynamic concept superposition.

A.7 Koopman Wavelet Decomposition

Define multiscale concept activation with continuous wavelets:

$$W_\psi(s, \tau) = \int \psi(t) \cdot \overline{\varphi_{s,\tau}(t)} dt \quad (17)$$

where $\varphi_{s,\tau}$ is a scaled and translated mother wavelet.

B Appendix B: Python Implementation Snippets

Appendix B: Python Implementation Snippets

B.1 Koopman Mode Estimation (EDMD)

```
def compute_observables(data):
    # Example: identity or Fourier/lifted features
    return np.stack([np.cos(data), np.sin(data)], axis=-1)

def koopman_modes(data):
    # data: (T x n) trajectory
    F = compute_observables(data)
    X, X_prime = F[:-1], F[1:]
    K = np.linalg.lstsq(X, X_prime, rcond=None)[0] # Koopman approximation
    evals, evecs = np.linalg.eig(K)
    return evals, evecs, K
```

B.2 Spectral Distance and Resonance

```
def spectral_distance(psi_i, psi_j):
    return np.linalg.norm(psi_i - psi_j)

def resonance_kernel(psi_i, psi_j, tau=1):
    return np.sum(psi_i[:-tau] * np.conj(psi_j[tau:])) / len(psi_i)
```

B.3 Narrative Walk Generator

```
def narrative_walk(koopman_graph, start_index, steps):
    path = [start_index]
    for _ in range(steps):
        current = path[-1]
        neighbors = koopman_graph[current]
        next_node = max(neighbors, key=lambda j: abs(np.dot(neighbors[j], neighbors[current])))
        path.append(next_node)
    return path
```

B.4 Phase Synchrony and Lyapunov Energy

```
def phase_synchrony_score(thetas):
    N = len(thetas)
    return np.mean([np.cos(thetas[i] - thetas[j])
                    for i in range(N) for j in range(N) if i != j])

def lyapunov_energy(K, thetas):
```

```

E = 0.0
for i in range(len(thetas)):
    for j in range(len(thetas)):
        E -= 0.5 * K[i, j] * np.cos(thetas[i] - thetas[j])
return E

```

B.5 Incremental Koopman Update (SPEED-like)

```

def update_koopman(K_prev, x_t, x_tp1, eta=0.01):
    # Rank-1 update
    x_t = x_t[:, None]
    x_tp1 = x_tp1[:, None]
    delta = x_tp1 - K_prev @ x_t
    K_next = K_prev + eta * delta @ x_t.T / (np.linalg.norm(x_t)**2 + 1e-6)
    return K_next

```

B.6 Visual Monitoring (Dashboard Scaffold)

```

import matplotlib.pyplot as plt

def plot_phase_trajectories(thetas):
    for theta in thetas.T:
        plt.plot(theta)
    plt.title("Phase Trajectories")
    plt.show()

def plot_spectral_entropy(eigenvalues):
    weights = np.abs(eigenvalues)
    p = weights / np.sum(weights)
    entropy = -np.sum(p * np.log(p + 1e-8))
    plt.bar(range(len(p)), p)
    plt.title(f"Spectral Entropy = {entropy:.3f}")
    plt.show()

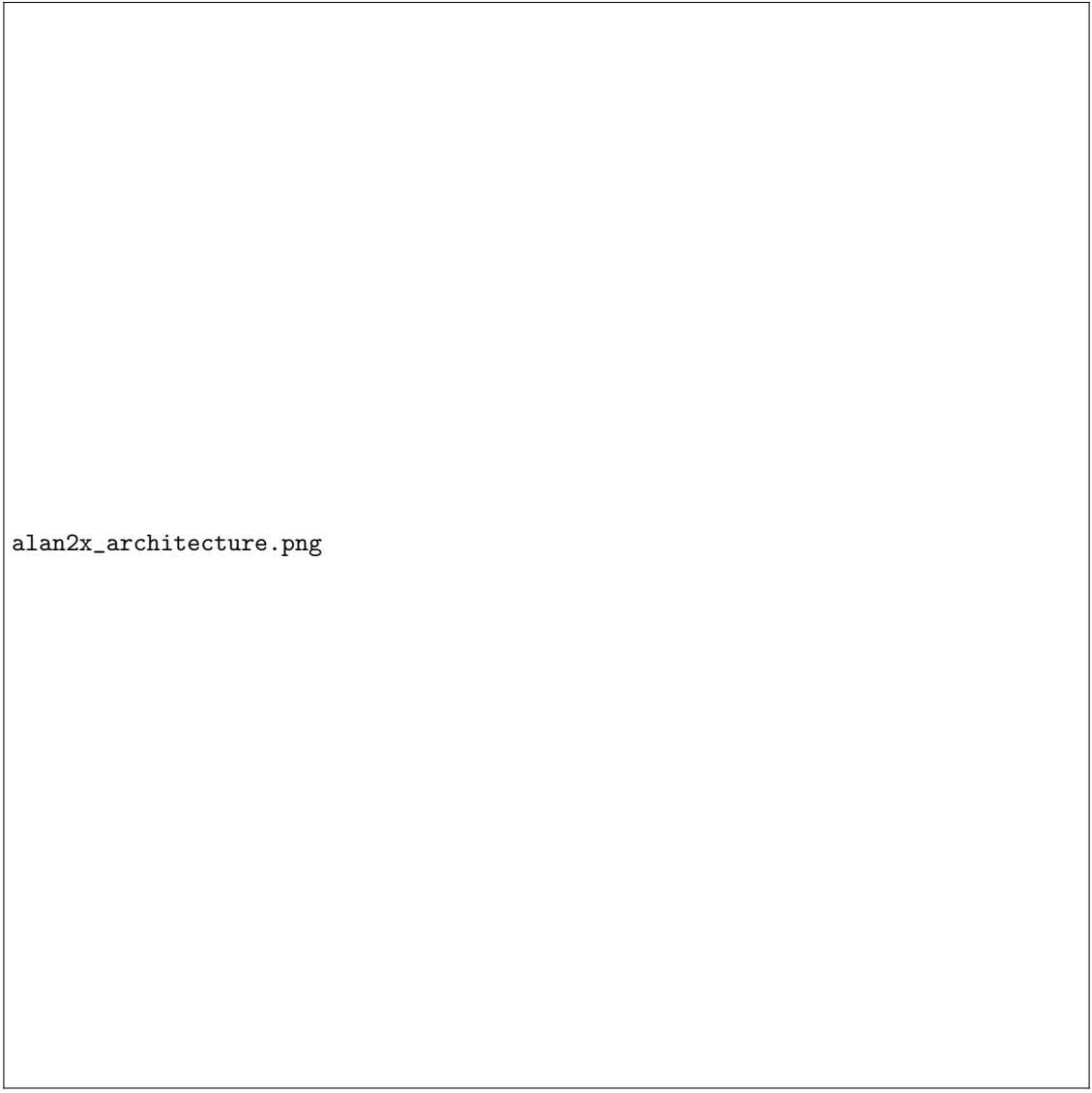
```

C Appendix C: Diagrams and Dashboard Visuals

Appendix C: Diagrams and Dashboard Visuals

C.1 ALAN 2.x System Diagram

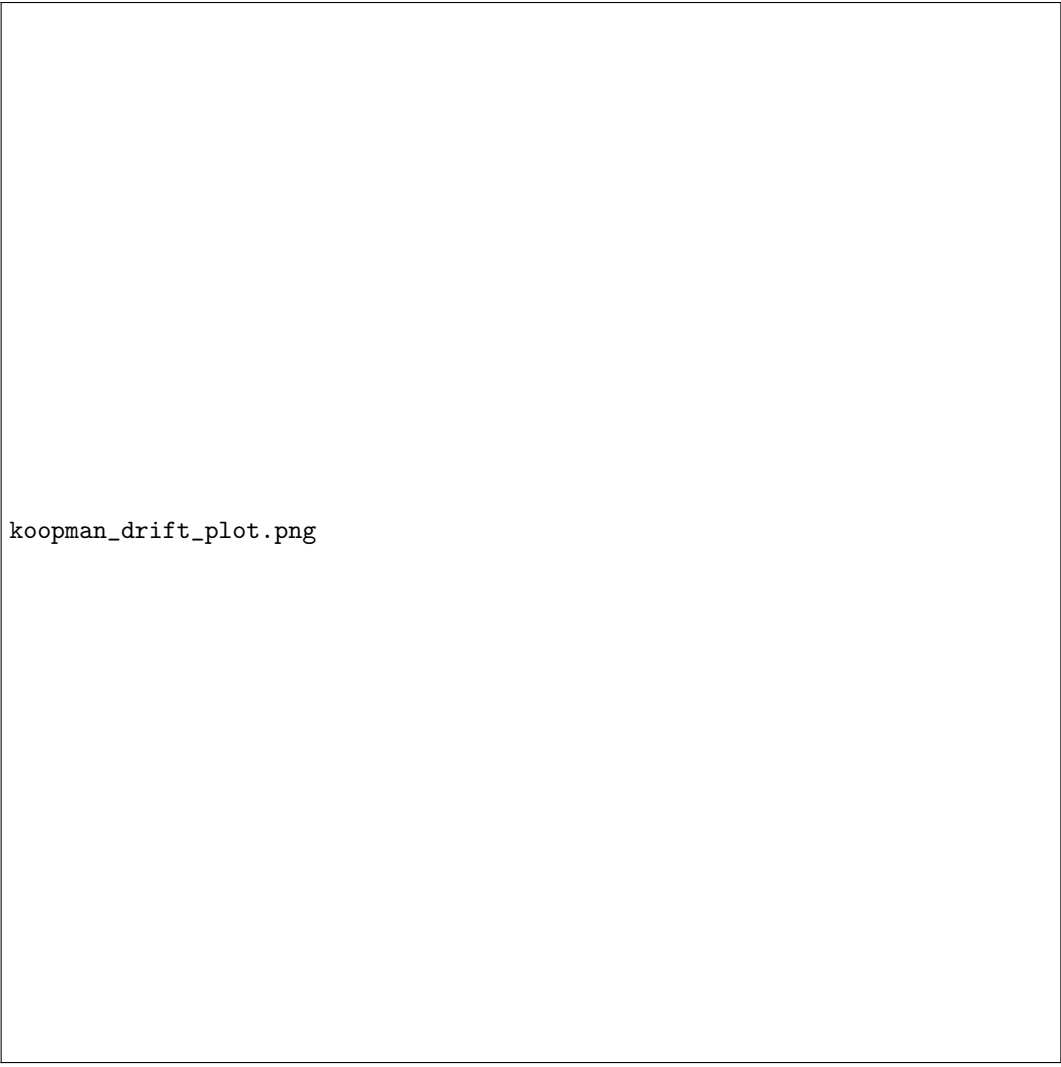
Figure 1: Full architecture of ALAN 2.x, illustrating interactions between the LCN graph, Banksy oscillator core, Koopman morphic engine, and spectral kernel layer. Arrows denote flow of concepts, synchronization, and eigenfunction updates.



alan2x_architecture.png

C.2 Koopman Mode Drift

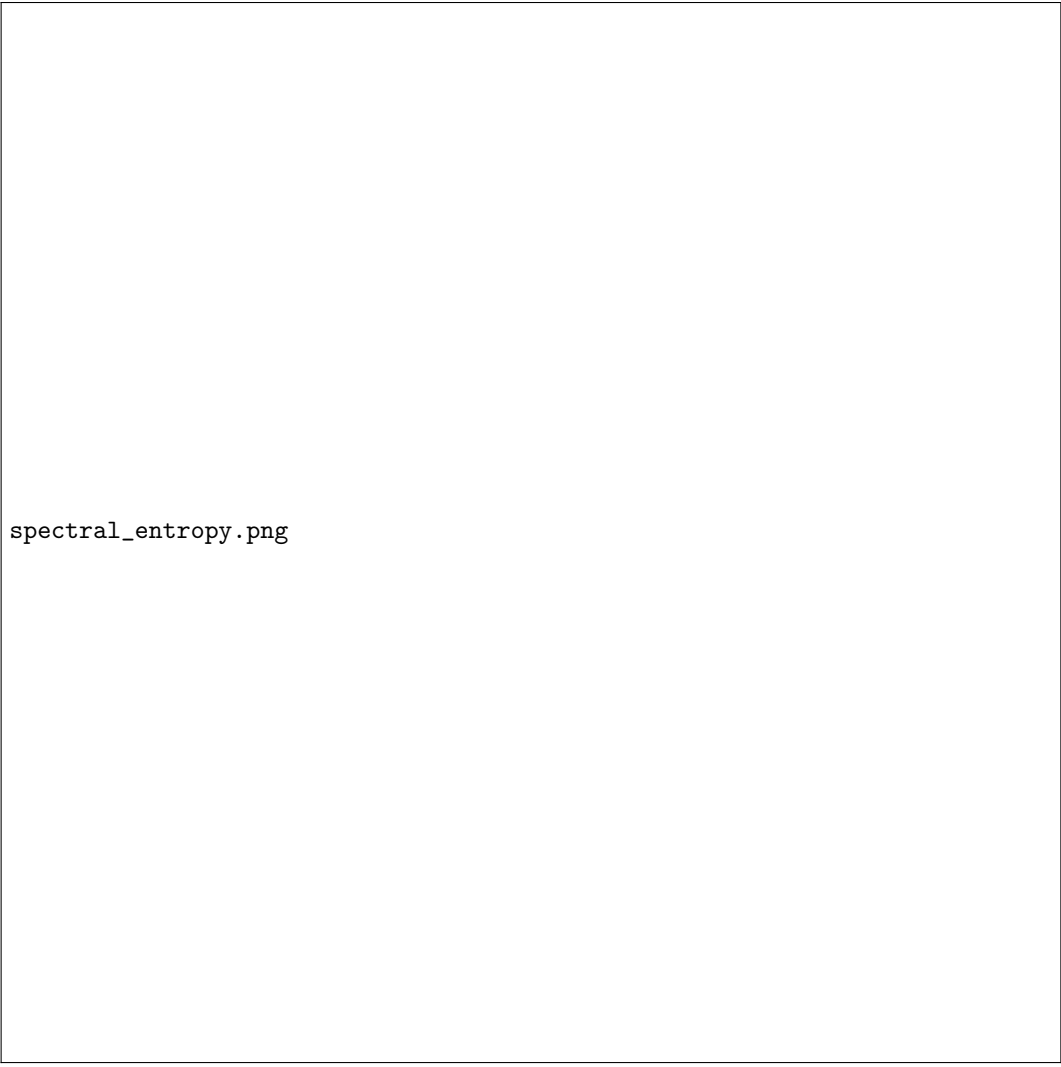
Figure 2: Real-time plot showing Koopman eigenvalue magnitudes and phase over time. This visualizes which conceptual modes dominate or decay.



koopman_drift_plot.png

C.3 Spectral Entropy Timeline

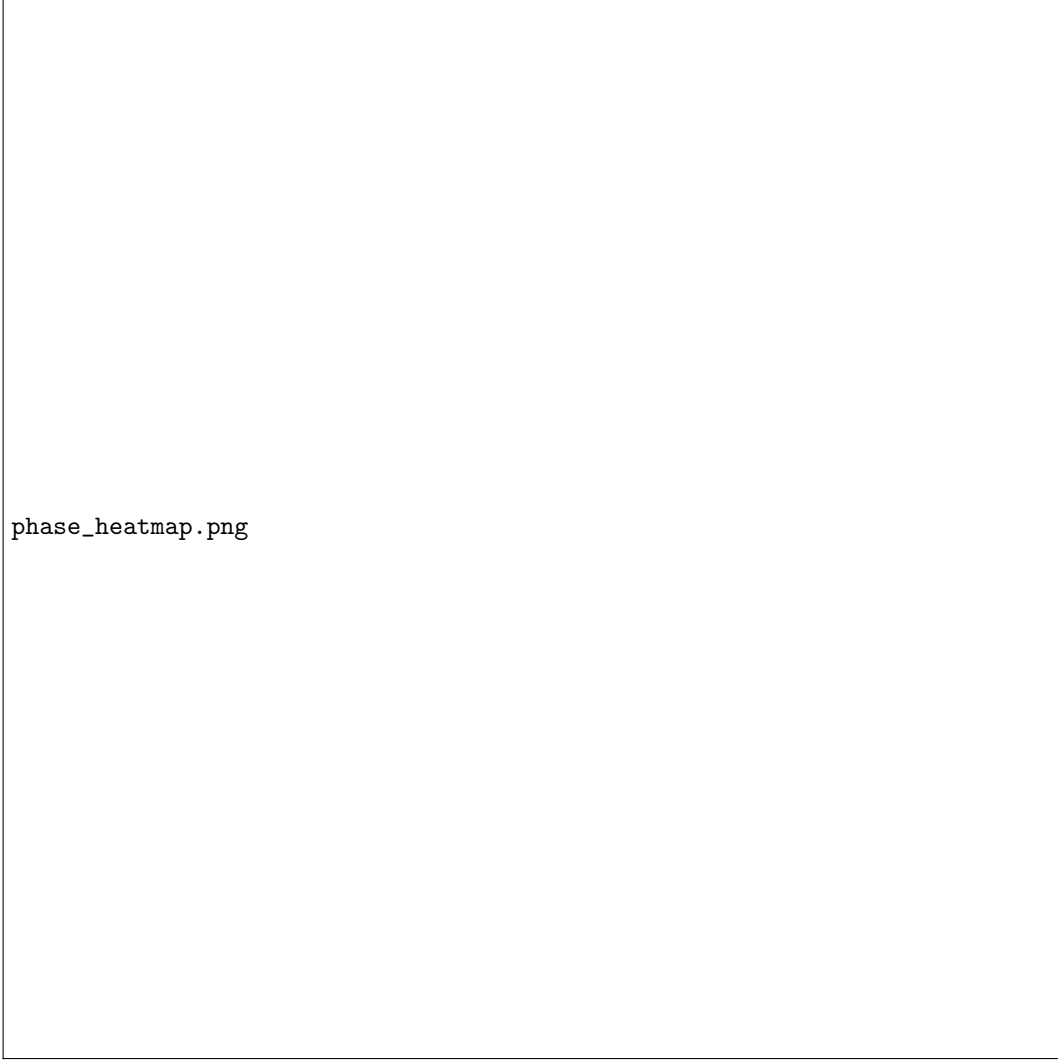
Figure 3: Timeline view of entropy in the Koopman spectral distribution. Sudden dips may indicate forgetting; spikes suggest narrative branching or hallucination.



spectral_entropy.png

C.4 Phase Synchrony Heatmap

Figure 4: Pairwise phase difference heatmap among concept-bound oscillators. Clustered dark blocks represent synchronous concept assemblies.



C.5 Resonance Kernel Map

Figure 5: Kernel matrix $\kappa(\psi_i, \psi_j; \tau)$ across top concept eigenfunctions at a selected time lag τ . Bright regions indicate spectral echoes or resonance zones.

resonance_kernel_matrix.png