Architectural Blueprint for ALEPH: Integrating Active Inference, Spiking Neural Networks, and Predictive Coding in Rust

Introduction to the Enactive Cognitive Architecture

The development of artificial intelligence has historically relied on feedforward or simple recurrent architectures that map inputs to outputs, functioning as passive information processors. The system known as ALEPH, currently integrating a Fractal Echo State Network (ESN) as a dynamic memory substrate and TinyLlama-1.1B as a semantic Narrative Cortex, operates within this paradigm. While this hybrid configuration allows for complex sequence generation and temporal pattern recognition, it inherently behaves as a reactive agent. It responds to prompts based on statistical likelihoods but lacks the intrinsic motivation, deep temporal horizon planning, and goal-directed agency required to operate autonomously in dynamic environments. Consequently, the system is highly susceptible to degenerative autoregressive loops, where highly predictable, repetitive outputs satisfy the objective function of the underlying language model but fail to generate meaningful or adaptive behavior.

To transcend these limitations and endow ALEPH with genuine, biologically plausible agency, the underlying operational mathematics must shift from standard maximum likelihood estimation to the Free Energy Principle (FEP). Formulated extensively in theoretical neurobiology and statistical physics, the Free Energy Principle asserts that all self-organizing systems must minimize their Variational Free Energy to resist the natural tendency toward thermodynamic entropy and structural dissolution.[1] For a cognitive agent, Variational Free Energy acts as a mathematically tractable upper bound on "surprise"—the negative log marginal likelihood of sensory observations.[2] By rearchitecting ALEPH to minimize surprise through an active perception-action loop, the system transitions from a passive observer to an active inference agent that continuously shapes its environment to fulfill its internal predictions.[3]

This comprehensive report delineates the architectural and mathematical blueprint required to construct this active inference system in the Rust programming language. It provides an exhaustive analysis of structuring the perception-prediction-action loop, specifically addressing the complex task of calculating a unified surprise metric from continuous reservoir states and discrete Large Language Model (LLM) logits. Furthermore, it outlines a rigorous computational mechanism for "boredom" driven by Expected Free Energy minimization, ensuring the agent actively seeks novelty to resolve systemic uncertainty.

Beyond the algorithmic loop, this report investigates the optimization of the physical computation layer. It evaluates the critical transition from rate-coded continuous weights in the Fractal ESN to a purely event-driven Spiking Neural Network (SNN) utilizing Leaky Integrate-and-Fire (LIF) neurons, proposing a Rust-native architecture that achieves maximum performance on standard Central Processing Units (CPUs). To support long-term, compositionally robust memory, the report details the implementation of Holographic Associative Memory via Vector Symbolic Architectures (VSA), providing a mechanism for graceful degradation that entirely bypasses the catastrophic forgetting inherent to standard vector databases. Finally, the analysis culminates in a structural design for emulating the six-layer canonical microcircuit of the mammalian neocortex, demonstrating how predictive coding and hierarchical message-passing can be efficiently simulated in software to drive top-down predictions and bottom-up sensory error corrections.

The Active Inference Loop in Rust

The core of an autonomous, biologically inspired system is the continuous cycle of sensory sampling, internal state updating, and environmental manipulation. Active inference provides a normative Bayesian framework that unifies these processes under a single imperative: the minimization of expected and variational free energy.[6] Implementing this in Rust requires a concurrent, event-driven architecture capable of handling asynchronous data streams from the environment, the internal state of the Spiking Neural Network, and the token generation of the LLM.

Architecting the Perception-Prediction-Action Cycle

The perception-prediction-action loop in an active inference agent relies on a generative model that captures the joint probability of observations and hidden environmental states. The architecture must continuously evaluate its beliefs about the world and take actions that either confirm those beliefs or change the world to match them.

In a Rust-based implementation, this loop can be structured using asynchronous task orchestration, enabling non-blocking execution of the three primary phases:

1. **Prediction (Generative Phase):** The agent utilizes its internal model, parameterized by its current state matrices, to forecast the immediate future. The Narrative Cortex (TinyLlama-1.1B) generates a distribution over upcoming semantic tokens, while the deeper layers of the cortical column architecture generate expected driving currents for the sensory input layers. This phase establishes the agent's prior belief, $P(s)$, regarding the hidden states of the world before new sensory data arrives.[4]
2. **Perception (Inference Phase):** The system receives an observation o_t from the environment or from its own internal dialog generation. The system calculates the discrepancy between the top-down prediction and the bottom-up observation, yielding a prediction error. To minimize this immediate error, the system performs perceptual

inference by updating its internal representation, moving the approximate posterior distribution $Q(s)$ closer to the true posterior $P(s|o_t)$.[2] In Rust, this involves gradient-based optimization on the internal state vectors, updating the scaffolding of the agent's current context.

3. **Action (Active Phase):** When internal belief updating is insufficient to minimize the prediction error, or when projecting into the future reveals high expected uncertainty, the agent must act. Active inference differs fundamentally from classical reinforcement learning by casting action as a mechanism to fulfill predictions rather than explicitly maximize a scalar reward signal.[10] The agent evaluates a repertoire of available policies π (action sequences) and selects the one that minimizes Expected Free Energy.[12] The execution of this policy alters the external environment or triggers an internal tool, generating new observations that restart the cycle.

This closed-loop system requires robust state management in Rust, utilizing thread-safe structures to allow the generative model to continuously sample the environment while the inference engine updates the model weights in the background.

Calculating Surprise from a Hybrid Architecture

The fundamental challenge in the ALEPH architecture is unifying two disparate computational paradigms: the discrete, autoregressive token generation of the TinyLlama LLM and the continuous, dynamic state space of the Fractal ESN (or its SNN evolution). The Free Energy Principle dictates that the system must minimize surprise, defined mathematically as $-\ln P(o)$.[2] However, calculating the exact marginal likelihood of observations is computationally intractable for complex models, necessitating the use of Variational Free Energy ($\mathcal{F}$) as an upper bound.[5]

The Variational Free Energy is defined as the Kullback-Leibler (KL) divergence between the approximate posterior $Q(s)$ and the prior $P(s)$, minus the expected log-likelihood of the observations under the generative model:

$$\mathcal{F} = D_{KL}[Q(s)||P(s)] - \mathbb{E}_{Q(s)}[\ln P(o|s)]$$

[5]

To construct a cohesive scalar value representing the total systemic surprise in Rust, the architecture must project the distinct loss landscapes of both the LLM and the reservoir into a shared dimensional space.

The discrete surprise is derived from the LLM's forward pass. When TinyLlama processes the current context, it yields a vector of raw logits representing the vocabulary space. These logits

are transformed into a normalized probability distribution via the softmax function.[15] The expected log-likelihood of the actual observed semantic token, given the model's current hidden state, is represented by $\ln P(o_{text}|s_{text})$. The surprise generated by the text stream is mathematically equivalent to the cross-entropy loss between the LLM's predicted probability distribution and the one-hot encoded representation of the actual observation.[2] In the Rust implementation, this is tracked as a continuous scalar metric representing the semantic predictability of the current narrative trajectory.

Simultaneously, the continuous surprise must be calculated from the dynamic memory reservoir. Whether utilizing a continuous-variable ESN or a threshold-based SNN, the reservoir maintains a high-dimensional state matrix. The generative model within the deeper layers of the architecture predicts the subsequent state vector $\hat{s}_{res}$ based on the current transition dynamics. When the actual state s_{res} is realized after integrating bottom-up sensory input, the discrepancy between the expected and actual states constitutes the continuous prediction error.[16] This error is typically computed as a precision-weighted Mean Squared Error (MSE) between the state vectors.

The total Variational Free Energy of the ALEPH system is the sum of the precision-weighted discrete semantic surprise and the continuous dynamic surprise. The "precision" weighting is a crucial neurobiological concept representing the inverse variance or confidence the system has in a particular sensory stream.[19] If the LLM is generating highly uncertain text (e.g., a flat probability distribution over the vocabulary), the system dynamically down-weights the semantic surprise and relies more heavily on the continuous reservoir dynamics to guide inference, and vice versa. This dynamic balancing acts as an attention mechanism, pulling the agent's focus toward the most reliable streams of information.

The Boredom Mechanism: Epistemic Value and Behavioral Change

A primary design goal for ALEPH is to eradicate the static looping behavior endemic to standard autoregressive agents. In a purely reactive LLM, the system generates text that maximizes the probability of the sequence, often leading to repetitive loops because repeating a highly probable phrase minimizes immediate cross-entropy loss. In the context of active inference, this loop produces highly predictable observations, driving the immediate Variational Free Energy (surprise) toward zero. However, a biological agent does not simply seek a dark, quiet room to minimize surprise; it must maintain its homeostasis in a dynamic world by looking forward in time.[14]

This temporal projection relies on Expected Free Energy (EFE), the objective function used to evaluate and select future action policies. EFE ($G(\pi)$) decomposes into two fundamental drives:

$$G(\pi) \approx \underbrace{-\mathbb{E}_Q[\ln P(o|C)]}_{\text{Pragmatic Value}} - \underbrace{\mathbb{E}_Q]}_{\text{Epistemic Value}}$$

21

The Pragmatic Value (often referred to as extrinsic or instrumental value) represents the drive to occupy states that the agent inherently expects or prefers, defined by a prior preference distribution C.[11] This aligns with traditional reward-seeking in reinforcement learning, ensuring the agent takes actions that maintain its operational integrity and fulfill its core directives.

The Epistemic Value (intrinsic motivation or information gain) represents the drive to reduce uncertainty about the hidden states of the world.[21] It quantifies the expected divergence between the agent's beliefs before an observation and its beliefs after an observation. Actions that yield high epistemic value are those that uncover novel information and refine the agent's internal generative model.[14]

The phenomenon of "boredom" occurs mathematically when the epistemic value of the current policy approaches zero.[19] When the agent is trapped in a repetitive loop, it already perfectly predicts its own output; thus, observing the output provides no new information, and the KL divergence term in the epistemic value equation vanishes.

To trigger a change in behavior, the Rust architecture must continuously monitor the moving average of the epistemic value of the active policy. When this value drops below a critical threshold, the agent registers high predictability (boredom). Because the overarching imperative is to minimize the total Expected Free Energy over a future temporal horizon, the policy evaluator will automatically begin to favor counterfactual policies that possess high uncertainty and, consequently, high epistemic affordance.[21]

This shifts the agent from a state of pragmatic exploitation to epistemic exploration, a process known as "epistemic foraging".[19] In practice, this mechanism triggers a behavioral pivot. The Rust control loop may inject targeted stochasticity into the LLM's sampling temperature, forcing the Narrative Cortex to explore alternative semantic trajectories. Alternatively, it may stimulate the motor/tool-use interface, prompting the agent to query an external API, search the internet, or physically manipulate its environment to generate novel sensory input.[28] By constantly seeking to resolve uncertainty, the active inference loop ensures that ALEPH remains dynamically engaged, curious, and permanently immunized against the static looping of reactive models.

Spiking Neural Networks vs. Rate Coding for Dynamic Memory

The dynamic memory component of ALEPH, currently instantiated as a Fractal Echo State

Network, relies on continuous weights and rate coding. While rate-coded recurrent networks are effective for general temporal pattern recognition, they are inherently limited by their abstraction of biological signaling. Transitioning the reservoir to a Spiking Neural Network (SNN) architecture offers profound advantages for both the temporal resolution of the agent's memory and the computational efficiency of its underlying hardware utilization.[30]

The Superiority of Temporal Processing in SNNs

In traditional artificial neural networks and standard Echo State Networks, information is transmitted via continuous, floating-point activation values. These values represent the mean firing rate of a biological neuron averaged over a specific time window—a scheme known as rate coding.[31] While mathematically convenient for gradient descent, rate coding is biologically implausible and operationally slow. To accurately convey a precise value, the system must wait for the entire observation window to elapse to calculate the frequency, inherently bottlenecking the speed of information transmission and increasing processing latency.[31]

Spiking Neural Networks abandon this continuous paradigm in favor of discrete, asynchronous event-driven computation, relying on binary spikes to encode and process data.[30] The choice of neural coding scheme within an SNN dramatically dictates its performance characteristics.

Neural Coding Scheme	Data Representation	Latency	Synaptic Operations	Primary Use Case
Rate Coding	Spike frequency over time	High	High	Static image processing, legacy ANN equivalence
Time-to-First-Spike (TTFS)	Precise timing of initial spike	Very Low	Very Low	Rapid temporal inference, low power edge computing
Phase Coding	Spike timing relative to background oscillation	Moderate	Moderate	High-noise environments, sensory encoding
Burst Coding	Rapid sequence of	Low	Low	Hardware fault tolerance,

	spikes followed by silence			network compression

Comparative analysis of neural coding schemes indicating the performance advantages of TTFS over traditional rate coding.[31]

For ALEPH's dynamic memory, adopting Time-to-First-Spike (TTFS) coding within a network of Leaky Integrate-and-Fire (LIF) neurons yields the optimal configuration. TTFS relies on the exact microsecond timing of a single spike to convey rich informational content, bypassing the need for lengthy observation windows.[30] Empirical evaluations demonstrate that TTFS coding achieves inference convergence with up to a 7.5x reduction in processing latency and a 6.5x reduction in total synaptic operations compared to rate-coded baselines.[31]

The LIF neuron model serves as the foundational computational unit for this architecture. Unlike a static non-linear activation function, a LIF neuron maintains an internal state variable—the membrane potential, $V(t)$ —which integrates incoming synaptic currents and exponentially leaks back to a resting state over time.[33] A spike is only emitted when this membrane potential crosses a critical threshold, after which the neuron enters a refractory period.[33] This inherent spike-time variability and threshold-driven non-linearity provide a vastly richer, more chaotic dynamical substrate than rate-coded continuous activations.[35] In the context of reservoir computing, this dynamic richness is essential; it allows the un-trained internal weights of the reservoir to naturally maintain activity at the "edge of chaos," maximizing the high-dimensional projection of temporal input signals and enabling highly complex predictive capabilities in the readout layers.[34]

CPU Optimization and Rust Framework Integration

While the theoretical advantages of SNNs are well established, achieving performance parity on conventional von Neumann architectures—specifically CPUs—presents a formidable software engineering challenge. CPUs and Graphics Processing Units (GPUs) are fundamentally designed to accelerate dense matrix multiplications, the cornerstone of traditional deep learning.[38] SNNs, conversely, generate sparse, asynchronous binary events. Applying dense matrix operations to SNNs on a CPU results in severe computational waste, as the processor expends cycles multiplying zeros for the vast majority of neurons that remain quiescent during any given time step.[38]

Evaluating the burn Framework: The Rust ecosystem possesses a highly capable deep learning framework known as burn. Designed with a focus on extreme flexibility, portability, and compute efficiency, burn features dynamic computation graphs, automatic kernel fusion, and a swappable backend architecture that allows seamless deployment across CPUs, CUDA, Metal,

and WebAssembly.[40] It is an exceptional framework for managing the dense matrix operations required by the TinyLlama-1.1B Narrative Cortex, providing the necessary infrastructure for tensor manipulation, autodifferentiation, and gradient optimization.[41]

However, burn is fundamentally constructed around dense multi-dimensional arrays.[44] If the entire LIF spiking reservoir were implemented using standard burn tensors, the CPU would be forced to process massive, dense weight matrices at every integration time step, entirely negating the energy efficiency and speed advantages of the sparse spiking paradigm.[46] While surrogate gradient descent methods can be used to train SNNs within dense frameworks like PyTorch or burn [30], these techniques do not translate into fast, sparse execution during deployment and inference.

The Custom Sparse Struct Architecture: To achieve genuine performance on a CPU, the Spiking Neural Network serving as ALEPH's reservoir must be implemented outside the dense tensor paradigm using custom, highly optimized Rust data structures. Experimental Rust crates focused on neuromorphic computing, such as omega-snn, micro-hnsw-wasm, and spiking_neural_networks, highlight the necessity of this approach.[33] These libraries achieve high-speed biological simulation by leveraging Rust's memory safety and zero-cost abstractions to implement event-driven architectures.

A performant architectural plan for the SNN reservoir within ALEPH involves three critical design elements:

1. **Sparse Adjacency Graphs:** Instead of representing the fixed synaptic connections of the reservoir as a dense $N \times N$ weight matrix, the network topography must be stored using Compressed Sparse Row (CSR) formats or adjacency lists.[50] This ensures that memory consumption scales linearly with the number of actual synapses rather than the square of the neuron count.
2. **Event-Driven Integration Queues:** The simulation loop must decouple the progression of time from the computation of network state. Rather than updating every neuron at every time step, the system utilizes a priority queue or ring buffer of spike events.[38] When a neuron fires, its target synapses are placed into the active queue for the next processing cycle. The CPU only processes the integration equations for neurons actively receiving incoming spikes, achieving massive computational sparsity.
3. **SIMD Intrinsics for Membrane Decay:** While synaptic integration is event-driven, the exponential decay of the membrane potential (the "leak") affects all neurons continuously. To handle this efficiently, the array of membrane potentials can be processed in bulk using Single Instruction, Multiple Data (SIMD) intrinsics provided by Rust's std::arch module. This allows a single CPU instruction to calculate the decay across multiple continuous memory blocks simultaneously, minimizing overhead.[51]

By coupling the dense, auto-differentiating power of the burn framework for the LLM processing with a custom, sparse, event-driven Rust structure for the LIF reservoir, ALEPH

achieves a hybrid architecture. This design leverages the strengths of both paradigms, ensuring that the heavy semantic processing is hardware-accelerated while the dynamic temporal memory operates with ultra-low latency and computational efficiency.[52]

Holographic Associative Memory via Vector Symbolic Architectures

As ALEPH continuously samples its environment and generates predictions, it requires a robust mechanism for storing and retrieving experiential data. Modern AI architectures heavily rely on standard vector databases, which store discrete, localized representations of information and utilize nearest-neighbor algorithms (e.g., cosine similarity) for retrieval. While effective for simple semantic search, these systems are fundamentally limited. They lack the compositional logic necessary for higher-order reasoning, and as the database reaches capacity, new entries force the overwriting of older data, resulting in catastrophic forgetting.

To provide a biologically plausible, long-term memory system capable of complex cognition, the architecture must transition to Vector Symbolic Architectures (VSA), widely known within the literature as Hyperdimensional Computing (HDC).[53]

The Mechanics of Hyperdimensional Computing

Hyperdimensional Computing operates on the mathematical premise that rich cognitive and perceptual representations can be modeled using extremely high-dimensional, pseudo-random vectors—typically referred to as hypervectors—ranging from 1,000 to 10,000 dimensions.[56] The foundational attribute of this hyperdimensional space is that any two randomly generated vectors are nearly orthogonal to one another, meaning they have a similarity score approaching zero.[57]

Information processing in VSA does not rely on deep backpropagation. Instead, complex, hierarchical data structures are assembled from basic atomic concepts using three deterministic algebraic operations [53]:

VSA Operation	Algebraic Symbol	Functional Mechanism	Cognitive Application	Mathematical Property
Binding	$\otimes$	Element-wise multiplication, XOR (for binary vectors), or Circular Convolution.	Associating variables with values to create key-value pairs (e.g., linking a specific object	The resulting hypervector is entirely dissimilar (orthogonal) to both of its constituent

			to its spatial location).	input vectors.
Bundling	$\oplus$	Element-wise addition followed by normalization or majority-rule thresholding.	Creating sets, superpositions, or generalized conceptual classes from multiple distinct entities.	The resulting hypervector is highly similar (non-orthogonal) to all of its constituent input vectors.
Permutation	Π	Cyclic bit shifting or matrix transformation.	Encoding sequences, order, and syntax (e.g., distinguishing 'A causes B' from 'B causes A').	The resulting hypervector is orthogonal to the original unshifted vector.

Summary of core Vector Symbolic Architecture operations, illustrating the mapping between algebraic mechanisms and cognitive functions.[53]

Through the recursive application of binding, bundling, and permutation, the agent can encode highly complex, tree-like relational graphs into a single hypervector of fixed dimensionality. For example, a complete interaction between the user, the LLM's semantic output, and the reservoir's temporal state can be bound together and bundled into a single experiential memory trace.

Graceful Degradation and Holistic Recall

The most significant advantage of VSA over standard vector databases is its distributed nature. In a conventional database, each memory is isolated. If a hard drive sector is corrupted, or if a capacity limit forces an overwrite, the specific memory is permanently destroyed. In VSA, memory representations are holographic; the information is not stored in a single element but is distributed globally across all dimensions of the hypervector.[53]

When multiple experiential hypervectors are bundled together to form a long-term memory store, the system operates in a state of superposition.[55] As the agent accumulates thousands of experiences, the memory structure does not fail catastrophically. Instead, as the capacity boundary of the high-dimensional space is approached, the system experiences "graceful

degradation".[60] The signal-to-noise ratio of retrieval operations smoothly declines, meaning that older, less frequently accessed, or less salient memories begin to blur, losing specific episodic details while retaining their generalized semantic structures.[55] This perfectly mimics human biological memory and provides ALEPH with a stable, lifelong learning trajectory devoid of abrupt knowledge collapse.

Implementation Strategies in Rust

Implementing VSA in Rust is highly advantageous due to the language's capacity for low-level memory management and hardware-intrinsic acceleration. For maximum efficiency, the architecture should utilize Binary Spatter Codes (BSC), a variant of VSA that restricts vector elements to binary values (0 and 1).

In a binary HDC system, hypervectors can be densely packed into arrays of 64-bit unsigned integers. A standard 10,000-dimensional vector requires an array of just 157 u64 integers ([u64; 157]), ensuring an exceptionally minimal memory footprint and perfect alignment with CPU cache architectures.[62]

Because the data is binary, the core VSA operations become incredibly fast, executing natively at the processor register level [62]:

- **Binding:** Executed via a simple bitwise XOR (^) instruction.
- **Permutation:** Executed via hardware bit shifts (<< and >>).
- **Similarity Search:** Calculated using the Hamming distance. Rust's std::arch intrinsics provide hardware-accelerated population count functions (popcnt or count_ones()), allowing the system to compare thousands of dimensions in a single CPU cycle.[62]

The Rust ecosystem offers several robust crates to facilitate this implementation. The amari-holographic and minuet crates provide production-ready toolkits for building cognitive memory architectures, featuring checkpoint persistence and abstraction layers for associative retrieval.[58] The hypervector crate offers implementations for Bipolar vectors (MBAT) and Semantic Spatial Pointers (SSP), supporting high-level bundling and similarity checks.[65] By integrating these crates, ALEPH's active inference loop can seamlessly bind the discrete semantic embeddings from TinyLlama with the continuous latent states of the SNN reservoir, creating unified, gracefully degrading episodic traces that continuously inform the generative model's prior expectations.

Emulating Cortical Columns for Predictive Coding

The ultimate realization of a biologically plausible, surprise-minimizing agent requires aligning the software topology with the functional architecture of the mammalian brain. The cerebral cortex is not a flat, feedforward network; it is organized vertically into highly repetitious, deeply interconnected, six-layer structures known as cortical columns.[66] These columns act as the canonical microcircuits of intelligence, and theoretical neuroscience heavily implicates this

specific geometry as the required hardware for implementing hierarchical predictive coding.[66]

The Canonical Microcircuit and Message Passing

Predictive coding theory upends the classical view of sensory processing. Instead of sensory data passively flowing upward from receptors to higher association areas, the cortex is dominated by top-down projections.[68] Higher cortical areas continuously generate probabilistic predictions about the states of lower areas. The lower areas compare these descending predictions against the actual incoming bottom-up sensory data to calculate a prediction error. Crucially, only the unresolved prediction errors—the "surprise"—are propagated back up the hierarchy to update the internal models.[68]

This dynamic minimizes Variational Free Energy across the entire network by suppressing predictable signals at the lowest possible level. It is also an extraordinarily bandwidth-efficient paradigm, akin to delta-encoding in data transmission; communication between nodes only occurs when the system's internal model fails to predict reality.[62]

This computational logic maps precisely onto the anatomical layers of the cortical column [68]:

- **Layer 4 (The Input Hub):** Acts as the primary recipient of feedforward sensory data from the thalamus or lower hierarchical columns.[73]
- **Layers 5 and 6 (The Generative Model):** House the "Prediction Neurons." Layer 5 pyramidal cells maintain the deep latent state representation of the environment ($Q(s)$) and generate the top-down predictions of expected sensory input.[17] Layer 6 acts to rapidly relay this information, providing feedback modulation to lower areas.[71]
- **Layers 2 and 3 (The Error Comparators):** House the "Error Neurons." These superficial layers compute the discrepancy between the bottom-up input arriving at Layer 4 and the top-down prediction descending from Layer 5.[68]
- **Inhibitory Interneurons:** The precise calculation of error relies on specific classes of interneurons (Parvalbumin [PV], Somatostatin, and Vasoactive Intestinal Peptide [VIP]). These local microcircuits generate both positive prediction errors (when bottom-up signals exceed top-down predictions) and negative prediction errors (when top-down predictions are not met by bottom-up input), maintaining a strict excitatory-inhibitory balance.[72]

Furthermore, empirical evidence indicates that these streams are segregated by temporal frequency. Feedforward prediction errors are broadcast upward via high-frequency gamma band oscillations, while top-down predictions are transmitted downward via slower beta and alpha bands.[62]

Software Architecture and Code Implementation in Rust

Simulating tens of thousands of detailed biophysical neurons per column is computationally

infeasible for a real-time AI agent. However, normative computational models have successfully distilled the complex biological logic of the cortical column into mathematically efficient network algorithms.[78]

To construct this within ALEPH, the Rust architecture must define a discrete CorticalColumn module that abstracts the biological layers into interacting tensor sub-populations.

1. State Definitions and Structures:

The module must encapsulate the distinct roles of the layers:

- L5_GenerativeState: A tensor representing the current hidden state belief, initialized via top-down context from higher layers (e.g., the LLM's semantic prior).
- L4_SensoryInput: A buffer receiving incoming data streams.
- L23_PredictionError: A tensor representing the mismatch. It is biologically separated into PE_positive and PE_negative structs to mimic the non-negative nature of neural spiking.[76]

2. The Hierarchical Message-Passing Algorithm:

Instead of executing sequential forward and backward passes across all layers, the column utilizes a continuous, localized integration loop:

- **Prediction Generation:** The L5_GenerativeState is multiplied by a local feedback matrix to generate an expected observation, which is projected into the superficial layers.
- **Error Computation:** The superficial layers compare the incoming L4_SensoryInput against the prediction. The resulting delta constitutes the L23_PredictionError.[17]
- **State Update via Local Learning:** The network does not rely on biologically implausible, globally coordinated backpropagation through time. Instead, the L5_GenerativeState is updated continuously via gradient descent driven solely by the local L23_PredictionError signal.[17] Synaptic weights are modified using local Hebbian plasticity rules, which respond strictly to the co-activation of the error nodes and the state nodes.[72]

3. Integration with the Narrative Cortex: In the unified ALEPH system, the array of Spiking Neural Network columns functions as the lower-order sensory and temporal processors, while the TinyLlama-1.1B LLM acts as the highest-order association cortex. The LLM provides the deep, semantic top-down priors that cascade down into the L5_GenerativeState of the SNN columns. Conversely, when the environment shifts unexpectedly and the SNN columns cannot resolve the discrepancy locally, massive bursts of L23_PredictionError (simulated gamma-band spikes) are transmitted upward.[62] This error signal forcefully disrupts the LLM's context window, updating the global semantic scaffold and driving the active inference agent to orient toward the source of the surprise.

Conclusion

The architectural transformation of ALEPH fundamentally redefines the operational boundaries

of artificial intelligence. By discarding the limitations of reactive, feedforward language models and static vector databases, the system embraces the mathematical rigor of the Free Energy Principle. The implementation of an active inference perception-action loop ensures that ALEPH possesses intrinsic motivation, actively minimizing both discrete semantic surprise and continuous temporal prediction errors to maintain homeostasis.

To support this enactive framework, the underlying computational hardware is fully optimized for CPU execution. The shift from rate-coded Echo State Networks to sparse, event-driven Spiking Neural Networks utilizing Time-to-First-Spike coding drastically reduces latency and processing overhead. The integration of Holographic Associative Memory through Vector Symbolic Architectures provides a mathematically robust, compositionally rich memory substrate that gracefully degrades under capacity constraints, mimicking the resilience of biological cognition. Finally, the structural emulation of the six-layer cortical column localizes learning and limits message passing strictly to unresolved prediction errors. Unified within a highly performant Rust architecture, these components synthesize an AI agent that does not merely process data, but actively anticipates, explores, and adapts to its environment.

Works cited

1. Free energy principle - Wikipedia, accessed February 19, 2026, https://en.wikipedia.org/wiki/Free_energy_principle
2. Computing the variational free energy in categorical models - pymdp's documentation!, accessed February 19, 2026, https://pymdp-rtd.readthedocs.io/en/latest/notebooks/free_energy_calculation.html
3. Tutorial on Active Inference. Active inference is the Free Energy... | by Oleg Solopchuk | Medium, accessed February 19, 2026, https://medium.com/@solopchuk/tutorial-on-active-inference-30edcf50f5dc
4. Cooperation and Social Rules Emerging From the Principle of Surprise Minimization - PMC, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC7858259/
5. Active Inference for Self-Organizing Multi-LLM Systems: A Bayesian Thermodynamic Approach to Adaptation - arXiv.org, accessed February 19, 2026, https://arxiv.org/html/2412.10425v2
6. On Epistemics in Expected Free Energy for Linear Gaussian State Space Models - MDPI, accessed February 19, 2026, https://www.mdpi.com/1099-4300/23/12/1565
7. Active Inference as a Model of Agency - arXiv.org, accessed February 19, 2026, https://arxiv.org/html/2401.12917v1
8. A step-by-step tutorial on active inference and its application to empirical data - PMC, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC8956124/
9. Perception-Prediction-Planning Loop - Emergent Mind, accessed February 19, 2026, https://www.emergentmind.com/topics/perception-prediction-planning-loop
10. [2410.00240] Demonstrating the Continual Learning Capabilities and Practical

Application of Discrete-Time Active Inference - arXiv, accessed February 19, 2026, https://arxiv.org/abs/2410.00240
11. The Missing Reward: Active Inference in the Era of Experience - arXiv.org, accessed February 19, 2026, https://arxiv.org/html/2508.05619v1
12. Active Inference and Epistemic Value in Graphical Models - Frontiers, accessed February 19, 2026, https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2022.794464/full
13. The Problem of Meaning: The Free Energy Principle and Artificial Agency - Frontiers, accessed February 19, 2026, https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2022.844773/full
14. Generalised free energy and active inference - PMC - NIH, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC6848054/
15. How LLMs Choose Their Words: A Practical Walk-Through of Logits, Softmax and Sampling, accessed February 19, 2026, https://machinelearningmastery.com/how-llms-choose-their-words-a-practical-walk-through-of-logits-softmax-and-sampling/
16. Active inference on discrete state-spaces: A synthesis - PMC, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC7732703/
17. Modelling Predictive Coding in the Primary Visual Cortex (V1) - bioRxiv.org, accessed February 19, 2026, https://www.biorxiv.org/content/10.1101/2025.11.01.686040v1.full.pdf
18. variational_free_energy - active-inference - Obsidian Publish, accessed February 19, 2026, https://publish.obsidian.md/active-inference/knowledge_base/free_energy_principle/mathematics/variational_free_energy
19. Forgetting ourselves in flow: an active inference account of flow states and how we experience ourselves within them - Frontiers, accessed February 19, 2026, https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2024.1354719/full
20. Forgetting Ourselves in Flow: An Active Inference Account of Flow States and How We Experience Ourselves Within Them - ResearchGate, accessed February 19, 2026, https://www.researchgate.net/publication/376482108_Forgetting_Ourselves_in_Flow_An_Active_Inference_Account_of_Flow_States_and_How_We_Experience_Ourselves_Within_Them
21. Discussion Paper Active inference and epistemic value - FIL | UCL, accessed February 19, 2026, https://www.fil.ion.ucl.ac.uk/~karl/Active%20inference%20and%20epistemic%20value.pdf
22. Born to act: Deferred action and desire as active inference - PMC, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC12615558/
23. Active Inference, epistemic value, and vicarious trial and error - PMC - NIH, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC4918783/

24. education - active-inference - Obsidian Publish, accessed February 19, 2026, https://publish.obsidian.md/active-inference/knowledge_base/free_energy_principle/applications/education
25. Active inference and epistemic value - PubMed, accessed February 19, 2026, https://pubmed.ncbi.nlm.nih.gov/25689102/
26. The Neural Correlates of Novelty and Variability in Human Decision-Making under an Active Inference Framework - eLife, accessed February 19, 2026, https://elifesciences.org/reviewed-preprints/92892
27. Active Inference and Cognitive Consistency - PMC - NIH, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC6191887/
28. Order and change in art: towards an active inference account of aesthetic experience, accessed February 19, 2026, https://royalsocietypublishing.org/rstb/article/379/1895/20220411/42813/Order-and-change-in-art-towards-an-active
29. Order and change in art: towards an active inference account of aesthetic experience - PMC, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC10725768/
30. Direct training high-performance deep spiking neural networks: a review of theories and methods - PMC, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC11322636/
31. Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems - PMC, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC7970006/
32. Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems - Frontiers, accessed February 19, 2026, https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2021.638474/full
33. omega-snn - crates.io: Rust Package Registry, accessed February 19, 2026, https://crates.io/crates/omega-snn
34. A mean-field approach to criticality in spiking neural networks for reservoir computing - PMC, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC12501264/
35. FORCE trained spiking networks do not benefit from faster learning while parameter matched rate networks do | bioRxiv, accessed February 19, 2026, https://www.biorxiv.org/content/10.1101/2024.08.16.608322v3.full-text
36. Comparison of FORCE trained spiking and rate neural networks shows spiking networks learn slowly with noisy, cross-trial firing rates | PLOS Computational Biology, accessed February 19, 2026, https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013224
37. A Mean-Field Approach to Criticality in Spiking Neural Networks for Reservoir Computing - bioRxiv, accessed February 19, 2026, https://www.biorxiv.org/content/10.1101/2025.02.05.636716.full.pdf
38. Spiffy: Efficient Implementation of CoLaNET for Raspberry Pi - arXiv, accessed February 19, 2026, https://arxiv.org/html/2506.18306v1
39. Htm vs spiking neural network - Numenta Theory, accessed February 19, 2026,

https://discourse.numenta.org/t/htm-vs-spiking-neural-network/5635
40. Burn, accessed February 19, 2026, https://burn.dev/
41. tracel-ai/burn: Burn is a next generation tensor library and Deep Learning Framework that doesn't compromise on flexibility, efficiency and portability. - GitHub, accessed February 19, 2026, https://github.com/tracel-ai/burn
42. burn - Rust - Docs.rs, accessed February 19, 2026, https://docs.rs/burn
43. burn - crates.io: Rust Package Registry, accessed February 19, 2026, https://crates.io/crates/burn
44. How Far Are We Really Learning? Emerging Neural Network Libraries in Rust - Reddit, accessed February 19, 2026, https://www.reddit.com/r/rust/comments/1i4ce5n/how_far_are_we_really_learning_emerging_neural/
45. Announcing Burn: New Deep Learning framework with CPU & GPU support using the newly stabilized GAT feature : r/rust - Reddit, accessed February 19, 2026, https://www.reddit.com/r/rust/comments/ynquym/announcing_burn_new_deep_learning_framework_with/
46. Compression and Inference of Spiking Neural Networks on Resource-Constrained Hardware - arXiv, accessed February 19, 2026, https://arxiv.org/html/2511.12136v1
47. Exploring Optimized Spiking Neural Network Architectures for Classification Tasks on Embedded Platforms - MDPI, accessed February 19, 2026, https://www.mdpi.com/1424-8220/21/9/3240
48. spiking_neural_networks - crates.io: Rust Package Registry, accessed February 19, 2026, https://crates.io/crates/spiking_neural_networks
49. micro_hnsw_wasm - Rust - Docs.rs, accessed February 19, 2026, https://docs.rs/micro-hnsw-wasm
50. MathisWellmann/reservoir_computer: Reservoir Computers on CPU - GitHub, accessed February 19, 2026, https://github.com/MathisWellmann/reservoir_computer
51. Burn 0.20 Released: Rust-Based Deep Learning With Speedy Perf Across CPUs & GPUs, accessed February 19, 2026, https://www.phoronix.com/news/Burn-0.20-Released
52. Neural networks - what crates to use? : r/rust - Reddit, accessed February 19, 2026, https://www.reddit.com/r/rust/comments/aofkg4/neural_networks_what_crates_to_use/
53. Self-Attention Based Semantic Decomposition in Vector Symbolic Architectures - arXiv, accessed February 19, 2026, https://arxiv.org/html/2403.13218v1
54. Cross-Layer Design of Vector-Symbolic Computing: Bridging Cognition and Brain-Inspired Hardware Acceleration - arXiv, accessed February 19, 2026, https://arxiv.org/html/2508.14245v1
55. Hyperdimensional Computing/Vector Symbolic Architectures, accessed February 19, 2026, https://www.hd-computing.com/
56. In-memory Vector Symbolic Architectures - ETH Zurich Research Collection, accessed February 19, 2026,

https://www.research-collection.ethz.ch/bitstreams/5c0600c9-ebf1-4ad1-81a3-7928c604f61d/download
57. Vector Symbolic Architectures as a Computing Framework for Emerging Hardware - PMC, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC10588678/
58. amari-holographic - crates.io: Rust Package Registry, accessed February 19, 2026, https://crates.io/crates/amari-holographic
59. The best of both worlds: Deep learning meets vector-symbolic architectures, accessed February 19, 2026, https://communities.springernature.com/posts/the-best-of-both-worlds-deep-learning-meets-vector-symbolic-architectures
60. Industrial-Algebra/Minuet: Holographic Memory And Database Toolkit - GitHub, accessed February 19, 2026, https://github.com/industrial-algebra/minuet
61. Graceful Degradation — Why It Is Important And How To Achieve It | by Mario Bittencourt, accessed February 19, 2026, https://medium.com/@mbneto/graceful-degradation-why-it-is-important-and-how-to-achieve-it-07f6d4a5f7d5
62. Bio-Inspired Neural Computing / Ai Nervous-System · GitHub, accessed February 19, 2026, https://gist.github.com/ruvnet/5cfda0d67fcf6b3580c7d1ea7ba426bd
63. micro-hnsw-wasm — Rust implementation // Lib.rs, accessed February 19, 2026, https://lib.rs/crates/micro-hnsw-wasm
64. amari_holographic - Rust - Docs.rs, accessed February 19, 2026, https://docs.rs/amari-holographic
65. rishikanthc/hypervector: A crate that implements hyperdimensional vectors and VSAs. - GitHub, accessed February 19, 2026, https://github.com/rishikanthc/hypervector
66. Canonical microcircuits for predictive coding - PMC - NIH, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC3777738/
67. Cortical column - Wikipedia, accessed February 19, 2026, https://en.wikipedia.org/wiki/Cortical_column
68. Predictive coding - Wikipedia, accessed February 19, 2026, https://en.wikipedia.org/wiki/Predictive_coding
69. Understanding cortical computation through the lens of joint-embedding predictive architectures - bioRxiv, accessed February 19, 2026, https://www.biorxiv.org/content/biorxiv/early/2025/11/25/2025.11.25.690220.full.pdf
70. Predictive Coding as a Model of Response Properties in Cortical Area V1 - PMC - NIH, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC6634102/
71. Layer 6 corticocortical neurons are a major route for intra- and interhemispheric feedback, accessed February 19, 2026, https://elifesciences.org/articles/100478
72. Cortical networks with multiple interneuron types generate oscillatory patterns during predictive coding - Our journal portfolio - PLOS, accessed February 19, 2026, https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1013469

73. Computational components of visual predictive coding circuitry - Frontiers, accessed February 19, 2026, https://www.frontiersin.org/journals/neural-circuits/articles/10.3389/fncir.2023.1254009/full
74. Modelling Predictive Coding in the Primary Visual Cortex (V1): Layer 4 Receptive Field Properties in a Balanced Recurrent Spiking Neuronal Network | bioRxiv, accessed February 19, 2026, https://www.biorxiv.org/content/10.1101/2025.10.20.683584v1.full-text
75. Understanding cortical computation through the lens of joint-embedding predictive architectures - bioRxiv.org, accessed February 19, 2026, https://www.biorxiv.org/content/10.1101/2025.11.25.690220v1.full-text
76. Cortical column model of predictive coding. The cortical column model... | Download Scientific Diagram - ResearchGate, accessed February 19, 2026, https://www.researchgate.net/figure/Cortical-column-model-of-predictive-coding-The-cortical-column-model-of-predictive_fig1_385324840
77. Ubiquitous predictive processing in the spectral domain of sensory cortex - eLife, accessed February 19, 2026, https://elifesciences.org/reviewed-preprints/109053v1
78. Constrained Predictive Coding as a Biologically Plausible Model of the Cortical Hierarchy - OpenReview, accessed February 19, 2026, https://openreview.net/pdf?id=TVpZaWNczF6
79. The Computational Properties of a Simplified Cortical Column Model - PMC, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC5019422/
80. Cortical Representation of Touch in Silico - PMC - NIH, accessed February 19, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC9588483/
81. Pyramidal Predictive Network: A Model for Visual-Frame Prediction Based on Predictive Coding Theory - MDPI, accessed February 19, 2026, https://www.mdpi.com/2079-9292/11/18/2969
82. Toward biologically plausible predictive coding - UvA-DARE (Digital Academic Repository), accessed February 19, 2026, https://pure.uva.nl/ws/files/291076838/Thesis.pdf
83. Multi-Layer Cortical Learning Algorithms - University of Memphis Digital Commons, accessed February 19, 2026, https://digitalcommons.memphis.edu/cgi/viewcontent.cgi?article=1026&context=ccrg_papers
84. Predictive Coding Networks for Temporal Prediction | bioRxiv, accessed February 19, 2026, https://www.biorxiv.org/content/10.1101/2023.05.15.540906v2.full-text
85. Deep Gated Hebbian Predictive Coding Accounts for Emergence of Complex Neural Response Properties Along the Visual Cortical Hierarchy - Frontiers, accessed February 19, 2026, https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2021.666131/full