How the Fully Unified Model (FUM) Works: A Technical Specification

Justin Lietz
Independent Researcher / FUM Project

March 30, 2025

Abstract

This document provides a detailed technical specification for the Fully Unified Model (FUM), a novel brain-inspired computational architecture. FUM aims to achieve autonomous, expert-level mastery across diverse domains using minimal training data by integrating sparse Spiking Neural Networks (SNNs), an emergent knowledge graph, biologically plausible learning rules (Spike Timing-Dependent Plasticity with excitatory and inhibitory dynamics modulated by a Self-Improvement Engine incorporating Temporal Difference learning and intrinsic motivation metrics), and adaptive structural plasticity. The architecture emphasizes computational efficiency, data efficiency, and emergent self-organization, contrasting with traditional deep learning paradigms. Key mechanisms, algorithms, practical implementation considerations, stability controls, scaling strategies, and the underlying rationale are described in detail.

Contents

1 High-Level Concept: Brain-Inspired Efficient Superintelligence

1.1 Goal

Achieve autonomous, expert-level mastery across diverse domains (e.g., Mathematics, Logic, Coding, Language, Visual Perception, Introspection) using **minimal training data** (target: 80-300 inputs). The aim is to outperform large-scale models (like 700B parameter LLMs) in accuracy and speed, while operating **efficiently on constrained hardware**.

Hardware Context (Development & Validation): The specific hardware configurations mentioned throughout this document (Linux workstation with AMD Threadripper PRO 5955WX, MI100 32GB VRAM, 7900 XTX 24GB VRAM, 512GB RAM, 6TB SSD) represent the author's (Justin Lietz) test environment. These are **not rigid requirements** for FUM deployment but serve as the platform where the model's theoretical foundations are validated. Notably, the predecessor model, AMN (Adaptive Modular Network), has already been successfully validated up to a 10-unit model size on this hardware, demonstrating the feasibility of the core concepts.

Why Minimal Data? Unlike LLMs requiring terabytes of data and vast pre-training, FUM aims for human-like learning efficiency, inferring complex patterns from sparse examples. This reduces reliance on massive datasets and computational resources, making advanced AI potentially achievable within the constraints of the development hardware. The design philosophy balances a minimal seeded structure during initialization with knowledge purely learned from these minimal examples (see Section 6.2 for details).

1.2 Core Philosophy

Mimic the efficiency (human brain $\sim$20W) and adaptability of biological brains by employing a **hybrid architecture**. This contrasts with monolithic architectures like Transformers used in most LLMs.

1. **Sparse Spiking Neural Networks (SNNs):** Chosen for inherent **temporal processing** (information encoded in spike timing, not just rate), potential for massive **energy efficiency** (neurons only compute when they spike, targeting ¿1M-fold savings vs. LLMs), and **biological plausibility**. High sparsity (target: 95%) drastically reduces the number of active connections, further saving computation and memory compared to dense ANNs/Transformers. Includes both excitatory and inhibitory neurons (typically 80:20 ratio) for stability and balanced dynamics.

2. **Emergent Knowledge Graph:** A dynamic graph structure replaces fixed layers or a predefined coordinator network. **Why?** This allows relationships between concepts and domains to emerge organically from neuron interactions and learning feedback, fostering adaptability and cross-domain knowledge transfer without manual design. This differs significantly from the fixed, layered structure of most deep learning models.

3. **Tensor-based Computation:** Leverages frameworks like PyTorch for efficient batch processing of certain operations (e.g., graph analysis, SIE calculations, clustering) and seamless integration with GPU acceleration (ROCm), complementing the SNN's event-driven nature via a carefully managed hybrid interface.

1.3 Key Differentiators vs. Broader Machine Learning Landscape

FUM's design choices distinguish it not only from LLMs but also from various other ML paradigms:

- **vs. Deep Learning (ANNs, CNNs, RNNs, Transformers):**
  - Neuron Model: Spiking (LIF) vs. Rate-based ANUs. Includes heterogeneity & intrinsic plasticity.
  - Learning Rule: Local STDP + SIE vs. Global backpropagation.
  - Architecture: Emergent graph + Structural plasticity vs. Fixed layers.
  - Data/Energy: Aims for higher efficiency.
  - Adaptability: Built-in vs. Retraining required.
- **vs. Traditional ML (SVMs, Decision Trees, k-NN, etc.):**
  - Representation: Distributed, dynamic vs. Explicit features/boundaries.
  - Learning: Online, continuous vs. Batch training.
  - Complexity Handling: Designed for high-dimensional, temporal data.
- **vs. Symbolic AI / Expert Systems:**
  - Knowledge Representation: Emergent weights vs. Explicit rules/symbols.
  - Learning: Data/feedback-driven vs. Pre-programmed knowledge.
  - Robustness: Aims for noise robustness vs. Potential brittleness.
- **vs. Standard Reinforcement Learning (Q-Learning, Policy Gradients):**
  - Core Mechanism: STDP modulated by SIE (with TD(0)) vs. Direct value/policy learning.
  - Representation: Integrated SNN/graph vs. Explicit tables/networks.
- **vs. Evolutionary Algorithms (Genetic Algorithms, Neuroevolution):**
  - Learning Timescale: Intra-lifetime vs. Generational.
  - Mechanism: Synaptic plasticity + RL vs. Population selection + Genetic operators.

2 Core Architecture Components

2.1 Spiking Neurons: Leaky Integrate-and-Fire (LIF) with Heterogeneity and Intrinsic Plasticity

2.1.1 Model & Rationale

Employs the standard Leaky Integrate-and-Fire (LIF) model. **Why LIF?** It offers a good balance between biological realism and computational tractability, capturing essential integrate-and-fire dynamics without the complexity of models like Hodgkin-Huxley. This efficiency is crucial for large-scale simulation.

2.1.2 Contrast with ANNs

Unlike Artificial Neuron Units (ANUs) in standard ANNs (like ReLUs, Sigmoids) which compute a static output based on summed weighted inputs in one pass, LIF neurons integrate inputs *over time* and communicate via discrete *spikes* (events), enabling richer temporal coding.

2.1.3 Equation & Simulation Timestep

The membrane potential V of a neuron i at discrete time t (where time progresses in steps of dt) is updated based on the previous potential $V_i(t-1)$, the input current $I_i(t)$ (sum of weighted spikes from connected neurons plus external input), and a leak term determined by the neuron's specific membrane time constant τ_i:

$$V_i(t) = V_i(t-1) + I_i(t) - \frac{V_i(t-1)}{\tau_i} \cdot dt \tag{1}$$

The input current $I_i(t)$ typically represents the sum of post-synaptic potentials resulting from pre-synaptic spikes:

$$I_i(t) = \sum_j w_{ji} s_j(t - \delta_{ji}) + I_{ext,i}(t) \tag{2}$$

where w_{ji} is the synaptic weight from neuron j to i, $s_j(t-\delta_{ji})$ represents pre-synaptic spikes from neuron j arriving at time t (potentially with delay δ_{ji}, although delays are often simplified or ignored in discrete simulations), and $I_{ext,i}(t)$ is any external input current. In the FUM implementation, $I_i(t)$ incorporates the term `w @ spikes(t-1)` and external encoded input.

Simulation Timestep (dt): Fixed at $dt = 1$ms. **Rationale:** Balances simulation fidelity (adequate for STDP dynamics with $\tau_\pm \approx 20$ms) and computational cost (avoids 100x cost of 0.01ms step), enabling reasonable training times on development hardware.

2.1.4 Firing Mechanism & Reset

A neuron i generates an output spike $s_i(t) = 1$ when its membrane potential $V_i(t)$ crosses its specific threshold $v_{th,i}$:

$$\text{If } V_i(t) \geq v_{th,i} \text{ then } s_i(t) = 1 \tag{3}$$

After firing, the potential is reset:

$$\text{If } s_i(t) = 1 \text{ then } V_i(t) \leftarrow v_{reset} \tag{4}$$

where v_{reset} is the resting potential (-70mV).

2.1.5 Heterogeneity

Neuron parameters are drawn from distributions at initialization:

- $\tau_i \sim \mathcal{N}(\mu = 20\text{ms}, \sigma^2 = (2\text{ms})^2)$
- $v_{th,i} \sim \mathcal{N}(\mu = -55\text{mV}, \sigma^2 = (2\text{mV})^2)$
- $v_{reset} = -70\text{mV}$ (fixed)

Rationale: Enhances network dynamics and robustness.

2.1.6 Intrinsic Plasticity (Adaptivity)

Parameters adapt based on firing rate $rate_i$ (calculated over 50 timesteps) to maintain a target rate (0.1–0.5 Hz). Let $rate_{target,min} = 0.1$ Hz, $rate_{target,max} = 0.5$ Hz. The update rules (applied every 50 timesteps) are:

$$\text{If } rate_i > rate_{target,max}: \quad v_{th,i} \leftarrow \min(v_{th,i} + 0.1\text{mV}, -50\text{mV}) \tag{5}$$

$$\tau_i \leftarrow \max(\tau_i - 0.1\text{ms}, 15\text{ms}) \tag{6}$$

$$\text{If } rate_i < rate_{target,min}: \quad v_{th,i} \leftarrow \max(v_{th,i} - 0.1\text{mV}, -60\text{mV}) \tag{7}$$

$$\tau_i \leftarrow \min(\tau_i + 0.1\text{ms}, 25\text{ms}) \tag{8}$$

2.1.7 Implementation (Kernel Scope & Responsibility)

Core LIF update (Eq. 1, implicitly Eq. 2), spike detection (Eq. 3), and reset (Eq. 4) executed in ROCm HIP kernel (`pulse_kernel`) on 7900XTX. Kernel records spike times in `spike_history`. STDP calculations and eligibility trace updates occur outside the kernel in PyTorch on MI100 (see Sec 2.2.6, 2.5.3).

2.2 Neural Plasticity: Spike Timing-Dependent Plasticity (STDP) with Inhibition

2.2.1 Purpose & Contrast with Backpropagation

2.2.2 Excitatory STDP Rule

For connections from an excitatory presynaptic neuron i to postsynaptic neuron j, let $\Delta t = t_j - t_i$ be the time difference between postsynaptic spike time t_j and presynaptic spike time t_i. The weight change Δw_{ij} is:

$$\Delta w_{ij}(\Delta t) = \begin{cases} A_+ \exp(-\Delta t/\tau_+) & \text{if } \Delta t > 0 \text{ (Potentiation)} \\ -A_- \exp(\Delta t/\tau_-) & \text{if } \Delta t < 0 \text{ (Depression)} \\ 0 & \text{if } \Delta t = 0 \end{cases} \tag{9}$$

where A_+, A_- are amplitudes and τ_+, τ_- are time constants.

2.2.3 Inhibitory STDP Rule & Neuron Types

Network contains inhibitory neurons (typically 20%). For connections from an inhibitory presynaptic neuron i to postsynaptic neuron j, the rule promotes stability:

$$\Delta w_{ij}(\Delta t) = \begin{cases} -A_+ \exp(-\Delta t/\tau_+) & \text{if } \Delta t > 0 \text{ (Weakening Inhibition)} \\ A_- \exp(\Delta t/\tau_-) & \text{if } \Delta t < 0 \text{ (Strengthening Inhibition)} \\ 0 & \text{if } \Delta t = 0 \end{cases} \tag{10}$$

Note the sign changes compared to Eq. 9.

2.2.4 Parameters & Weight Range

Parameters: $A_+ = 0.1$, $A_- = 0.12$, $\tau_+ = 20\text{ms}$, $\tau_- = 20\text{ms}$. Weights w_{ij} are clamped: $w_{ij} \in [-1, 1]$.

2.2.5 Eligibility Traces for Temporal Credit Assignment

To link STDP events to delayed rewards, each synapse (i, j) has an eligibility trace e_{ij} updated as:

$$e_{ij}(t) = \gamma e_{ij}(t-1) + \sum_{\text{pairs at } t} \Delta w_{ij}(\Delta t) \tag{11}$$

where $\gamma = 0.95$ is the decay factor and the sum is over STDP events occurring at timestep t. This trace integrates past STDP events with exponential decay:

$$e_{ij}(t) \approx \sum_{k \leq t} \gamma^{t-k} \left(\sum_{\text{pairs at } k} \Delta w_{ij}(\Delta t_k) \right) \tag{12}$$

Stored as sparse FP16 tensor on MI100, initialized to 0.

2.2.6 STDP Calculation Location & Final Weight Update

STDP change $\Delta w_{ij}(t)$ is calculated per 50ms window in PyTorch on MI100 based on `spike_history` from 7900 XTX. Traces e_{ij} are updated on MI100. The final weight update occurs on 7900 XTX after receiving e_{ij} and R_{total} from MI100:

$$w_{ij}(T+1) = \text{clip}\left(w_{ij}(T) + \eta_{eff}(T) \cdot R_{total}(T) \cdot e_{ij}(T), -1, 1\right) \tag{13}$$

where T denotes the end of the simulation window (e.g., 50 timesteps), and η_{eff} is the modulated learning rate (see Sec 2.3.7).

2.2.7 Role & Stability Mechanisms (Incl. Synaptic Scaling)

STDP enables associative learning. Stability is crucial and maintained by:

- Inhibitory neurons and inhibitory STDP.
- Global inhibition proportional to network average rate.
- Intrinsic Plasticity (Sec 2.1.6).
- Synaptic Scaling: Applied every 1000 timesteps *after* STDP/SIE updates. For each neuron j, calculate total excitatory input $S_{exc,j} = \sum_{i \in \text{Exc}} \max(0, w_{ij})$. If $S_{exc,j} > 1$, calculate scale_factor $= 1/S_{exc,j}$. Then update incoming excitatory weights w_{ij} where $w_{ij} < 0.8$ by multiplying by scale_factor. This preserves strong connections ($w_{ij} \geq 0.8$).

2.3 Continuous Reinforcement Learning: Self-Improvement Engine (SIE) with TD Learning

2.3.1 Purpose & Contrast with Supervised Learning

2.3.2 Reward Signal (`total_reward`) & Component Calculation

Calculated on MI100 after each 50-timestep window:

$$R_{total}(T) = \delta_{TD}(T) + \nu(T) - h(T) + \beta_{SB}(T) \tag{14}$$

2.3.3 TD Learning Specifics (TD(0), Value Function)

Uses TD(0) algorithm. Value function $V(s)$ predicts expected future cumulative reward for state s. States s correspond to cluster IDs from adaptive clustering (Sec 4.4).

$$\delta_{TD}(T) = r(T) + \gamma V(s_{T+1}) - V(s_T) \tag{15}$$

$$V(s_T) \leftarrow V(s_T) + \alpha \delta_{TD}(T) \tag{16}$$

where $r(T)$ is external reward $(+1, -1, 0)$, $\gamma = 0.9$ (discount factor), $\alpha = 0.1$ (learning rate). `V_states` tensor (FP16 on MI100) stores values, initialized to 0.

2.3.4 Novelty Calculation

Based on cosine similarity between current input encoding $I_{enc}(T)$ and recent history I_{hist}:

$$\nu(T) = 1 - \max_{I' \in I_{hist}} \left(\frac{I_{enc}(T) \cdot I'}{\|I_{enc}(T)\| \|I'\|} \right) \tag{17}$$

I_{hist} stored in buffer on MI100. $\nu \in [0, 1]$.

2.3.5 Habituation Calculation

Uses counter $C_h[i]$ for each pattern i in history. If max(similarity) > 0.9 for pattern i: $C_h[i] \leftarrow \min(C_h[i] + 0.1, 1)$. Counter decays: $C_h \leftarrow C_h \times 0.95$ periodically.

$$h(T) = C_h[\text{matched index}] \tag{18}$$

$h \in [0, 1]$. Stored/updated on MI100.

2.3.6 Self-Benefit Calculation (Complexity & Impact Metrics)

$$\beta_{SB}(T) = \text{complexity}(T) \times \text{impact}(T) \tag{19}$$

Complexity: Network-wide average spikes per neuron per timestep.

$$\text{complexity}(T) = \frac{\sum_{i, t \in [T - T_{win} + 1, T]} s_i(t)}{N \cdot T_{win}} \tag{20}$$

where N is number of neurons, $T_{win} = 50$. (Can be calculated per cluster). **Impact:** Reduction in firing rate variance σ^2. Let σ^2_{before} be avg variance over last 1k steps, σ^2_{after} be variance for current window, $\sigma^2_{baseline}$ be avg over 10k steps.

$$\text{impact}(T) = \text{clip}\left(\frac{\sigma^2_{before} - \sigma^2_{after}}{\max(\sigma^2_{baseline}, 0.01)}, -1, 1 \right) \tag{21}$$

Safeguard: Penalty reduced during exploration: $\text{impact}_{adj} = \text{impact} \times (1 - \nu(T))$.

2.3.7 Influence on Learning (Modulation)

Total reward modulates base learning rate $\eta = 0.01$.

$$\text{mod\_factor}(T) = 2\sigma(R_{total}(T)) - 1 \quad \text{where } \sigma(x) = \frac{1}{1 + e^{-x}} \tag{22}$$

$$\eta_{eff}(T) = \eta(1 + \text{mod\_factor}(T)) \tag{23}$$

Final weight update (Eq. 13) uses η_{eff} and R_{total}, resulting in quadratic scaling emphasis.

2.3.8 Goal

Maximize cumulative R_{total} over time via STDP and structural plasticity.

2.4 Unified Knowledge Graph (Emergent)

2.5 Tensor-Based Computation and Hybrid Interface

2.5.1 Hybrid Approach Rationale

2.5.2 Frameworks & Hardware Roles (Development Context)

2.5.3 Interface: Data Flow & Synchronization

3 Multimodal Input/Output Processing

3.1 Encoder Mechanism: From Raw Data to Spike Trains

3.1.1 Purpose & Contrast with LLM Input

3.1.2 Encoding Methods (Rate & Temporal)

3.1.3 Poisson Spike Generation Details

Spikes generated based on target frequency f and $dt = 1\text{ms}$. Probability $p = f \cdot dt$. Spike $s(t) = 1$ if rand() $< p$ and refractory period (5ms) allows. Max rate capped at 200 Hz.

3.1.4 Output & Extensibility

3.2 Decoder Mechanism: From Spike Trains to Structured Output

4 Emergent Behaviors and Self-Organization

4.1 Emergent Energy Landscape

Concept contrasts with predefined energy functions like Hopfield: $E = -\frac{1}{2}\sum_{i \neq j} w_{ij} s_i s_j$. FUM's stability emerges from dynamics guided by SIE. Stability measured by firing rate variance $\sigma^2 < (0.05\text{ Hz})^2$.

4.2 Knowledge Graph Evolution (Detailed)

Driven by STDP/SIE modulating w_{ij}.

4.3 Self-Modification (Structural Plasticity - Detailed Algorithms)

4.3.1 Rationale & Triggers

4.3.2 Growth Algorithm

4.3.3 Pruning Algorithm

4.3.4 Rewiring Algorithm & Limits

4.4 Adaptive Domain Clustering (Dynamic k and Edge Cases)

4.4.1 Purpose & Mechanism

4.4.2 Determining Number of Clusters (k)

Dynamic k via Silhouette Score $S = (b - a)/\max(a, b)$, with $k = \max(\text{argmax}(S_k), k_{min})$.

4.4.3 Cluster Assignment & Reward Attribution (Domain Identification)

Reward attribution uses soft probabilities $p_i = \text{softmax}(\text{similarity}_i)$. Average reward avg_reward_c.

4.4.4 Edge Case Handling (Small k, Empty Clusters)

Override k if $k < k_{min}$. Set $avg\_reward_c = 0$ if $num\_inputs_c = 0$, triggers growth.

4.4.5 Adaptation

5 Training and Scaling: Detailed Implementation Strategy

5.1 Phase 1: Random Seed Sprinkling (Foundation Building)

5.1.1 Objective

5.1.2 Cellular Components & Mechanisms (Incl. Initialization Strategy & Dynamic States)

5.1.3 Physics of Initial State Formation

Energy Minimization (LIF leak), Stochastic Initialization ($w \sim U$), Phase Space Dynamics.

5.1.4 Expected Outcome

5.2 Phase 2: Tandem Complexity Scaling (Refinement and Competence)

5.2.1 Objective

5.2.2 Cellular Components & Mechanisms

5.2.3 Mathematical Formulations

Includes STDP (Eq. 9, 10), Eligibility Trace (Eq. 11), SIE Modulation (Eq. 23), TD Learning (Eq. 15, 16), Silhouette Score.

5.2.4 Expected Outcome

5.3 Phase 3: Continuous Self-Learning (Autonomy and Mastery)

5.3.1 Objective

5.3.2 Cellular Components & Mechanisms

5.3.3 Emergent Physics Principles

Self-Organized Criticality (SOC): Maintained near critical point by learning rules, balancing stability/adaptability.

5.3.4 Expected Outcome

5.4 Scaling Strategy: Implementation Details

5.4.1 Distributed Computation (Graph Sharding)

5.4.2 Asynchronous Updates & Synchronization Details

5.4.3 Memory Management (Incl. Parameter Server & Caching)

5.4.4 Hardware Optimization (Development Context)

5.5 Practical Considerations: Tuning, Debugging, Stability, and Robustness

5.5.1 Hyperparameter Sensitivity & Tuning Strategy

Sensitivity: High for η, γ, SIE weights; Low for τ, v_{th}, k. Tuning: Bayesian Optimization (`gp_minimize`) maximizing avg SIE reward periodically.

5.5.2 Debuggability and Interpretability

Logging (rates, w, rewards, clusters). Anomaly detection ($\sigma^2 > 0.1$ Hz^2, extreme R_{total}). Visualization (‘networkx‘, ‘matplotlib‘). Diagnostic process defined.

5.5.3 Computational Cost of Overhead Components

Estimates: Core SNN 0.33ms/k-steps, Initial Overhead 0.96ms/k-steps (74%). Optimized Overhead 0.17ms/k-steps (12%). Mitigation maintains net efficiency.

5.5.4 Long-Term Stability and Potential Drift

Mechanisms: Inhibition, Scaling ($w < 0.8$ threshold), IP, Plasticity limits. Forgetting via decay ($w* = 0.99$, prune $|w| < 0.01$). Consolidation via ‘persistent‘ tag exempting from decay.

5.5.5 Robustness to Input Noise/Anomalies

Mechanisms: Input freq filtering, SNN tolerance (leak, inhibition), Anomaly flag (high avg rate), SIE smoothing/capping.

5.5.6 Justification for Specific Algorithmic Choices

TD(0): Efficiency, stability, suits sparse rewards vs TD(λ)/Q-learning. K-Means: Efficiency, scalability, interpretability vs DBSCAN/Hierarchical/Spectral.

6 Feasibility and Rationale Summary

6.1 Why is FUM considered feasible despite its ambitious goals?

6.2 Strategic Foundation: Balancing Initialization and Learning

Balance: 10-15

References

[1] Lietz, J. (2025). *How the Fully Unified Model (FUM) Works.* [Unpublished technical specification / Design document]. (Details the specific FUM architecture and potentially novel contributions: Overall Integrated System, specific SIE formulation, Integrated Structural Plasticity System, Adaptive Domain Clustering Integration, Multi-Phase Training Strategy, Emergent Knowledge Graph & Routing Mechanism, Emergent Energy Landscape Concept, Specific Hybrid Computation Model, Specific Temporal Encoding Schemes, Minimal Data Philosophy).

[2] Burkitt, A. N. (2006). A review of the leaky integrate-and-fire neuron model. *Biological Cybernetics*, *95*(1), 1-19.

[3] Destexhe, A., & Marder, E. (2004). Plasticity in single neuron and circuit computations. *Nature*, *431*(7010), 789-795.

[4] Gerstner, W., Kistler, W. M., Naud, R., & Paninski, L. (2014). *Neuronal dynamics: From single neurons to networks and models of cognition.* Cambridge University Press.

[5] Lapicque, L. (1907). Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation. *Journal de Physiologie et de Pathologie Générale*, *9*, 620-635.

[6] Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, *10*(9), 1659-1671.

[7] Marder, E., & Goaillard, J. M. (2006). Variability, compensation, and homeostasis in neuron and network function. *Nature Reviews Neuroscience*, *7*(7), 563-574.

[8] Pfeiffer, M., & Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Frontiers in Neuroscience*, *12*, 774. https://doi.org/10.3389/fnins.2018.00774

[9] Triesch, J. (2007). Synergies between intrinsic and synaptic plasticity mechanisms. *Neural Computation*, *19*(4), 885-909.

[10] Bi, G. Q., & Poo, M. M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, *18*(24), 10464-10472.

[11] Chklovskii, D. B., Mel, B. W., & Svoboda, K. (2004). Cortical rewiring and information storage. *Nature*, *431*(7010), 782-788.

[12] Helias, M., Tetzlaff, T., & Diesmann, M. (2014). The correlation structure of local neuronal networks intrinsically results from recurrent connectivity. *PLoS Computational Biology*, *10*(1), e1003458. https://doi.org/10.1371/journal.pcbi.1003458

[13] Holtmaat, A., & Svoboda, K. (2009). Experience-dependent structural synaptic plasticity in the mammalian brain. *Nature Reviews Neuroscience*, *10*(9), 647-658.

[14] Markram, H., Lübke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, *275*(5297), 213-215.

[15] Song, S., Miller, K. D., & Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, *3*(9), 919-926.

[16] Turrigiano, G. G., Leslie, K. R., Desai, N. S., Rutherford, L. C., & Nelson, S. B. (1998). Activity-dependent scaling of quantal amplitude in neocortical neurons. *Nature*, *391*(6670), 892-896.

[17] Vogels, T. P., Sprekeler, H., Zenke, F., Clopath, C., & Gerstner, W. (2011). Inhibitory plasticity balances excitation and inhibition in sensory pathways and memory networks. *Science*, *334*(6062), 1569-1573. https://doi.org/10.1126/science.1211095

[18] Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, *19*(6), 1468-1502.

[19] Frémaux, N., & Gerstner, W. (2016). Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, *9*, 85. https://doi.org/10.3389/fncir.2015.00085

[20] Izhikevich, E. M. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex*, *17*(10), 2443-2452. https://doi.org/10.1093/cercor/bhl147

[21] Marsland, S. (2014). *Machine learning: an algorithmic perspective.* CRC press.

[22] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*(1), 9-44.

[23] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

[24] Bak, P., Tang, C., & Wiesenfeld, K. (1987). Self-organized criticality: An explanation of the 1/f noise. *Physical Review Letters*, *59*(4), 381-384.

[25] Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A. C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. (2021). Knowledge graphs. *ACM Computing Surveys (CSUR)*, *54*(4), 1-37. https://doi.org/10.1145/3447772

[26] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*(8), 2554-2558.

[27] Mitchell, M. (2009). *Complexity: A guided tour.* Oxford University Press.

[28] Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, *20*(1), 359-392.

[29] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281-297). University of California Press.

[30] Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*, 53-65.

[31] Brette, R. (2015). Philosophy of the spike: rate-based vs. spike-based theories of computation. *Frontiers in Systems Neuroscience*, *9*, 151. https://doi.org/10.3389/fnsys.2015.00151

[32] Thorpe, S., Delorme, A., & Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Networks*, *14*(6-7), 715-725.

[33] AMD. (n.d.). *ROCm Documentation*. Retrieved March 30, 2025, from https://rocm.docs.amd.com/

[34] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32* (pp. 8026-8037). Curran Associates, Inc.

[35] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25* (pp. 2951-2959). Curran Associates, Inc.