SPARSITRON

A Compute-Governed Sparse Learning Substrate for Continual and Bounded Intelligence

Technical Report v1.0
Date: January 2026

Author:
Vaibhav Chiruguri
Intellicortex Technologies Private Limited
Bengaluru, India
vaibhav@intellicortex.in

Patent Pending (India)

Abstract

Modern artificial intelligence systems scale capability by increasing parameter count and unconstrained computation, resulting in escalating energy cost, unpredictable inference latency, and limited support for continual learning. In contrast, biological cognition operates under strict resource constraints, relying on sparse activity, local state dynamics, and gradual consolidation of experience into stable cognitive structure.

This technical report presents SPARSITRON, a compute-governed sparse learning substrate in which computation is enforced as a hard invariant rather than an optimization objective. The architecture employs adaptive inverse-fan-out routing to bound total computation per layer, integer-based node state machines for deterministic execution, and generator-defined stable connectivity in place of stored adjacency matrices. Learning occurs through structural plasticity and consolidation, whereby persistent experiential patterns update connectivity generators rather than accumulating unbounded state.

We describe the system architecture, execution model, and a public benchmark harness demonstrating compute invariance, continual adaptation under task drift, memory stabilization through consolidation proxies, and hybrid control of dense computation. The results indicate a viable pathway toward predictable, energy-efficient, and continually learning intelligence systems operating under fixed computational budgets.

---

1. Introduction

The dominant trajectory in artificial intelligence has been to increase model capacity and computational expenditure in pursuit of higher performance. Large neural networks, particularly transformer-based architectures, have achieved impressive results across language, vision, and multimodal tasks. However, this progress has come with significant costs: rising energy consumption, infrastructure complexity, unpredictable inference latency, and limited ability to adapt after deployment.

From a systems perspective, these limitations arise because computation is treated as a soft constraint. Training and inference procedures optimize accuracy subject to available hardware, but the architecture itself does not enforce hard bounds on resource usage. As a result, deployment in constrained, real-time, or long-lived environments remains challenging.

Biological intelligence presents a contrasting model. Neural systems operate under fixed energy budgets, sparse and event-driven activity, and gradual consolidation of experience into stable cognitive structure. Adaptation occurs continuously without global retraining, and computation remains bounded regardless of environmental complexity.

This report explores an alternative architectural design point inspired by these observations: compute-governed intelligence, where learning and reasoning must occur within explicit and enforced computational limits. SPARSITRON is introduced as a sparse learning substrate designed to embody this principle.

---

2. Design Goals and Non-Goals

2.1 Design Goals

SPARSITRON is designed with the following goals:

1. Hard Compute Bounds: Enforce strict upper limits on routing and computation per layer per timestep.
2. Sparse, Event-Driven Execution: Activate only a small subset of nodes at any time.
3. Deterministic and Hardware-Efficient Operation: Use integer arithmetic and bounded loops suitable for GPU execution.
4. Continual Learning: Support adaptation over time without catastrophic forgetting or retraining.
5. Bounded Memory Growth: Prevent unbounded accumulation of learned state.
6. Compatibility with Existing Models: Enable hybrid integration with dense neural networks.

2.2 Non-Goals

The present work explicitly does not aim to:

- Achieve state-of-the-art accuracy on standard machine learning benchmarks.
- Provide an end-to-end language model or task-specific system.
- Introduce new gradient-based optimization methods.
- Claim biological realism or cognitive completeness.

SPARSITRON is intended as a learning substrate, not a complete application-level model.

---

3. Architecture Overview

The SPARSITRON architecture consists of multiple layers of computational nodes. Each node maintains a small local integer state and may emit routing events to downstream nodes when its state exceeds a threshold. At any timestep, only a subset of nodes in each layer is active.

The system enforces a routing budget B, which defines the maximum number of routing events permitted per layer per timestep. This budget is a first-class architectural parameter and is not exceeded by design.

Connectivity within the system is divided into two fields:

- Stable Prior Connectivity, represented by deterministic generators.
- Adaptive Memory Connectivity, formed and pruned during operation.

A periodic consolidation process updates generator parameters based on persistent patterns observed in adaptive memory.

---

4. Compute Governance via Adaptive Inverse Fan-Out

A central mechanism in SPARSITRON is the enforcement of compute governance through adaptive inverse fan-out.

Let n_{active} denote the number of active nodes in a layer at a given timestep. The fan-out f of each active node is determined as an inverse function of n_{active}, for example:

$$f = \max\left(1, \left\lfloor \frac{B}{n_{\text{active}}} \right\rfloor \right)$$

or alternatively,

$$f = \max(1, k - n_{\text{active}})$$

where B and k are predefined constants.

As activity increases, individual node fan-out decreases, ensuring that the total number of routing events remains bounded. This mechanism prevents computational explosion under adversarial or highly stimulating inputs and provides predictable execution cost.

---

5. Integer Node State Dynamics

Each node operates as a local integer state machine comprising:

- an input counter,
- a decay parameter, and
- a firing threshold.

Incoming routing events increment the counter, which decays at each timestep. When the counter exceeds the threshold, the node fires, emits routing events, and resets or partially resets its state.

This design eliminates floating-point activation functions and enables deterministic, low-overhead execution suitable for parallel hardware.

---

6. Generator-Defined Stable Connectivity

Stable connectivity in SPARSITRON is represented not by stored adjacency matrices but by deterministic connectivity generators.

When a node fires, destination nodes are generated as a function of:

- the node identifier,
- a context value (salt),
- a motif identifier, and
- one or more codebook entries.

Formally,

$$D = G(\text{node\_id}, \text{context}, \text{motif}, \text{codebook})$$

Generators may employ deterministic hashing, integer mixing, or rule-based selection to produce a bounded set of destinations. This approach allows stable cognitive structure to be encoded compactly and modified without explicit edge storage.

---

7. Routing Motifs and Codebooks

Motifs define reusable classes of routing behavior, such as:

- competition,
- pooling or abstraction,
- inhibition,
- novelty detection,

- sequential propagation.

Codebooks specify destination regions using compact parameters (e.g., base index, span, stride). Together, motifs and codebooks define structural patterns that guide information flow through the network.

---

8. Adaptive Memory and Structural Plasticity

In addition to generator-defined routing, SPARSITRON maintains an adaptive memory connectivity store. Connections in this store are formed, strengthened, weakened, and pruned based on local activity patterns and optional utility signals.

Structural plasticity operates without global gradient propagation and enables rapid adaptation to changing environments.

---

9. Dual Connectivity Fields

Propagation within SPARSITRON occurs through the union of stable prior connectivity and adaptive memory connectivity. These two fields evolve on different timescales and under different rules:

- Prior connectivity provides stable cognitive structure.
- Adaptive memory captures recent experience.

This separation allows continual learning without overwriting established structure.

---

10. Consolidation into Updated Generators

A consolidation process executes periodically at a lower frequency than adaptive memory updates. Consolidation identifies persistent or high-utility patterns within adaptive memory and updates parameters of the connectivity generator, such as:

- context values,
- motif selection rules,
- codebook spans or indices.

Rather than copying adaptive connections into a stored stable adjacency, consolidation modifies the generator rules themselves, embedding learned structure while maintaining bounded state.

---

11. Execution Model and GPU Implementation

SPARSITRON is designed for efficient execution on GPUs and similar parallel hardware. The execution model consists of:

1. Integer node dynamics kernels.
2. Active node counting.
3. Budgeted routing kernels enforcing the routing budget.

All loops are bounded, and all operations use integer arithmetic, enabling predictable latency and energy usage.

---

12. Benchmark Harness

A public benchmark harness accompanies this report. The benchmarks are designed to validate architectural invariants rather than task-specific accuracy.

12.1 Compute Invariance

Stress tests demonstrate that routing events per layer never exceed the specified budget under adversarial activation.

12.2 Continual Learning (Synthetic)

Sequential task shifts test the system's ability to adapt structurally without retraining.

12.3 Memory Stabilization

Memory proxies show growth during exposure followed by stabilization or reduction during consolidation phases.

12.4 Ablation Studies

Disabling adaptive inverse fan-out results in degraded stability, demonstrating the necessity of compute governance.

12.5 Hybrid Control of Dense Computation

A toy demonstration shows how SPARSITRON signals can gate dense computation, reducing operations under a fixed budget.

---

13. Related Work

SPARSITRON is related to, but distinct from, several prior approaches:

- Spiking Neural Networks: event-driven but lacking hard compute governance.
- Dynamic Sparse Training: weight-centric and gradient-based.
- Mixture-of-Experts Models: soft routing with expensive experts.
- Graph Neural Networks: reliance on stored adjacency structures.

SPARSITRON focuses on compute invariants and generator-defined structure rather than optimization-centric learning.

---

14. Limitations and Future Work

Current limitations include:

- reliance on synthetic benchmarks,
- absence of end-to-end task evaluations,
- evolving consolidation heuristics,
- early-stage hybrid integration.

Future work will explore real-world datasets, refined consolidation strategies, and deeper integration with dense models.

---

15. Conclusion

This report introduced SPARSITRON, a compute-governed sparse learning substrate that enforces bounded computation, supports continual adaptation, and stabilizes learned structure through consolidation. By treating computation as a first-class architectural invariant, SPARSITRON offers an alternative pathway toward predictable, energy-efficient, and long-lived intelligence systems.
