Convergent Architectures: Computational Vossels and Compute-in-Memory State Space Models as a Unified Framework for Edge AI

Greg Passmore

Abstract

Two independent lines of research, developed without knowledge of each other, have converged on the same fundamental problem from opposite directions. My Computational Vossel architecture (Passmore, 2025, 2026) provides a biomimetic software framework in which each spatial data element simultaneously stores and processes sensor measurements, organized by seven agent classes modeled on the organelles of the eukaryotic cell. Zhang et al. (2026) provide a co-designed hardware system in which state space models are physically mapped onto resistive RAM crossbar arrays, with tungsten oxide memristors implementing state decay through their native conductance physics rather than through digital computation. Neither work, by itself, is complete. The vossel architecture needs a physical substrate that matches its storage-compute fusion principle; the CIM-SSM hardware needs an organizing computational framework that specifies what to compute, how to route results, and how to manage multi-modal sensor fusion across spatial elements. Each provides exactly what the other lacks.

This paper makes the convergence thesis precise. I show that the seven vossel agent classes map directly and non-trivially onto the primitive operations of the CIM-SSM hardware: the Endoplasmic Reticulum agent implements the SSM state evolution equation; the Golgi agent implements the output projection matrix $\mathbf{C}$; the Nucleus agent configures decay rates and active channels; the Mitochondria agent maps to independently power-gated CIM cores; the Receptor agent implements selective input weighting through the input projection matrix $\bar{\mathbf{B}}$; and the mipvol hierarchy maps to cascaded SSM blocks. The correspondence is not analogical but structural: the mathematical formalism of each vossel agent maps to a specific matrix operation or physical process in the CIM-SSM chip.

Together, the two architectures constitute a complete, biologically-grounded edge AI stack that eliminates the von Neumann bottleneck for real-time sensor processing. Energy analysis shows the combined system achieves orders-of-magnitude efficiency advantages over GPU-based edge inference for asynchronous event-stream workloads, with direct applicability to tactical ISR platforms operating under severe size, weight, and power constraints.

Keywords: computational vossel, compute-in-memory, state space models, edge AI, neuromorphic computing, biomimetic architecture, ISR, RRAM, WOx memristors, event-driven processing

1. Introduction

1.1 The Von Neumann Bottleneck Has Become an Edge AI Crisis

The separation of computation from memory, formalized in von Neumann's 1945 report on the EDVAC (Von Neumann, 1945), was a practical engineering choice that has governed computer architecture for eighty years. For general-purpose sequential computation on modest data volumes, the choice was entirely reasonable. For the class of problems now dominating edge AI (continuous, asynchronous, multi-modal sensor streams where the data must be processed at the point of collection under strict power constraints) the von Neumann architecture is the wrong abstraction at the architectural level.

The energy cost of data movement makes the problem concrete. Horowitz (2014) quantified the gap in 45-nm CMOS: a floating-point arithmetic operation costs approximately 0.1 to 1 pJ, while an 8-byte DRAM access costs 1,000 to 2,000 pJ, a three-to-four order-of-magnitude difference. An L1 cache access costs roughly 20 pJ, still a factor of 20 above

arithmetic. The implication is that for memory-bound inference workloads, roughly 99% of the energy consumed by a conventional processor is moving data rather than computing with it. Modern GPU architectures improve bandwidth substantially but do not change the fundamental separation; they move data faster, at nearly the same energy-per-byte cost. For cloud-scale inference with abundant power, this is an engineering constraint. For a tactical unmanned aerial vehicle with a 25-watt power budget, it is a mission-limiting architectural defect.

The ISR (intelligence, surveillance, and reconnaissance) context makes the stakes concrete. I developed the vossel framework in response to the observation that modern ISR sensor collections generate data at rates that centralized architectures cannot process in a tactically relevant timeframe. A contemporary multi-INT collection environment (combining electro-optical, infrared, synthetic aperture radar, SIGINT, and LiDAR sensors across a fleet of persistent surveillance platforms) can generate on the order of hundreds of terabytes per day per platform. Routing this data to a central processing node, applying conventional deep learning inference pipelines, and returning results to forward operators introduces latencies measured in minutes to hours, which is operationally unacceptable when targets have dwell times measured in seconds to minutes.

The answer, biologically speaking, is obvious. The brain does not route sensory data to a central processing nucleus and wait for results. Each neuron processes its local inputs and passes results to its neighbors. The eukaryotic cell is an even more striking model: it stores genetic information and processes molecular signals in the same physical structure, through a population of organelles whose biochemistry implements sophisticated local computation without reference to any external processor. The cell's energy consumption for this computation is measured in femtojoules per operation, compared to picojoules for the most efficient digital circuits today. The cell is not merely an inspiration; it is an existence proof that storage-compute fusion architectures are physically realizable, computationally complete, and extraordinarily energy efficient.

The gap between biological efficiency and silicon efficiency is not merely quantitative. It reflects a fundamentally different architectural philosophy. The cell does not separate memory from computation because it has no abstraction boundary between the two: the enzyme that catalyzes a reaction is itself stored as a protein structure within the same cellular compartment where it operates, and the product of the reaction modifies the local chemical environment that influences future reactions in the same compartment. There is no bus, no cache hierarchy, no fetch-decode-execute pipeline, no address space. The cell's computation is instantiated in the physics of molecular interactions, not in a separate computational substrate operating on a separate memory substrate. This is what I mean by storage-compute fusion at the architectural level, and it is the organizing principle behind both the Computational Vossel and, as I will argue, the CIM-SSM co-design of Zhang et al. (2026).

The quantitative case for architectural change is also now unambiguous in the silicon domain. IBM's NorthPole chip (Modha et al., 2023), by placing the entire network on-chip and eliminating off-chip DRAM access, achieves 25 times greater energy efficiency than 12-nm GPUs on ResNet-50 inference. The NeuRRAM chip (Wan et al., 2022) demonstrates that RRAM-based CIM achieves a further 1.6 to 2.3 times improvement in energy-delay product over previous RRAM-CIM designs, with a projected 535 times improvement when scaled from 130-nm to 7-nm fabrication. Ambrogio et al. (2023) demonstrated 12.4 TOPS per watt for speech recognition inference on IBM's analog AI chip, hundreds of times more efficient than contemporary CPUs and GPUs for equivalent tasks. The trajectory is clear: the architectural shift from von Neumann to in-memory computation is already underway in hardware, and the missing element is a software framework that organizes this hardware into a coherent, multi-modal, spatially structured computing system. The Computational Vossel provides that framework.

1.2 The Computational Vossel: Architecture from Biology

My Computational Vossel architecture (Passmore, 2025, 2026) is a formal system for edge AI that takes the biological cell seriously as an engineering precedent rather than a loose metaphor. The starting point is the passive vossel

(Volumetric Singular Spectral Element), introduced in Passmore (2025) as a fidelity-preserving data container for ISR sensor measurements. A vossel element e_n is the formal tuple:

$$e_n = (\mathbf{p}_n,\ \mathcal{F}_n,\ \Sigma_n,\ \boldsymbol{\lambda}_n,\ \mathbf{a}_n)$$

where $\mathbf{p}_n = (x_n, y_n, z_n, t_n) \in \mathbb{R}^3 \times \mathbb{R}$ is the spatio-temporal center; $\mathcal{F}_n \subset \mathbb{R}^3$ is the sensor footprint envelope; Σ_n is the uncertainty kernel (a positive-semidefinite covariance matrix encoding point spread function, geolocation uncertainty, and measurement noise); $\boldsymbol{\lambda}_n \in \mathbb{R}^{B_n}$ is the spectral observation vector across B_n sensor bands; and $\mathbf{a}_n \in \mathbb{R}^{M_n}$ is a variable-length auxiliary attribute vector. This structure preserves the measurement physics that conventional data formats discard: spatial extent, anisotropy, spectral content, uncertainty geometry, and sensor provenance.

The Computational Vossel (Passmore, 2026) extends each vossel element into an active computational unit by augmenting e_n with four additional structures:

$$CV_n = (e_n,\ \mathcal{A}_n,\ \mathcal{M}_n,\ \mathcal{S}_n,\ \mathcal{I}_n)$$

where $\mathcal{A}_n$ is the agent population (computational agents modeled on cellular organelles); $\mathcal{M}_n$ is the membrane interface (governing selective permeability to incoming signals); $\mathcal{S}_n$ is the internal cytoplasmic state (computational state and energy budget); and $\mathcal{I}_n$ is the intercellular interface (structured communication to neighboring vossels). This is not a modest extension of a data structure; it is a paradigm shift from storage to autonomous computation.

The agent population $\mathcal{A}_n$ comprises seven classes, each modeled on a specific eukaryotic organelle: the Nucleus agent (genome and configuration control, one per vossel), the Endoplasmic Reticulum agents (data ingestion and processing pipeline), the Golgi agents (routing, annotation, and output packaging), the Mitochondria agents (resource management and energy allocation), the Lysosome agents (data lifecycle management and quality control), the Cytoskeleton agents (internal transport and load balancing), and the Receptor agents (selective membrane permeability and signal recognition). These classes are not arbitrary; each has a precisely defined mathematical formalism derived from the biology of its correspondent organelle, and each plays a non-redundant role in the vossel's computational economy.

The Computational Vossel is governed by three laws (Passmore, 2026). Law 1 (Storage-Compute Indivisibility) states that the data content and the computational content of a vossel are aspects of a single, indivisible element. Law 2 (Local Computation Primacy) states that all analytical operations on a vossel are performed locally by the vossel's own agents. Law 3 (Biological Correspondence) states that every computational mechanism in the architecture has a corresponding biological mechanism in the eukaryotic cell, and the correspondence is functional, not merely metaphorical. These three laws distinguish the Computational Vossel from all prior active-voxel and cellular automata proposals, and they define the architectural constraints that turn out to match the CIM-SSM hardware's operational envelope with remarkable precision.

The vossel architecture organizes individual elements into mipvol hierarchies (multiresolution volume hierarchies) by doubling cell sizes at each level, analogous to graphics mipmapping, providing coarse-to-fine spatial decomposition that supports adaptive fidelity retrieval, reinforcement learning curriculum structures, and hierarchical analysis at multiple scales simultaneously.

Two aspects of the vossel architecture's design are particularly relevant to its alignment with CIM-SSM hardware. The first is the event-driven processing model: vossel agents compute in response to data arrival events, not on a fixed scheduler. At any given moment, the vast majority of vossels in an ISR grid are quiescent (receiving no new sensor data in their spatial region), and only a small active fraction are computing. This sparsity is an operational reality, not a simplifying assumption: ISR targets are sparse, sensor coverages change rapidly, and most of the world volume is empty most of the time. An architecture that pays computational cost proportional to data activity rather than clock rate

is therefore the only one that achieves good energy efficiency at ISR operational scales. The second is the continuous-time state model: unlike neural networks whose computations are organized around discrete forward passes, the vossel's agents maintain state that evolves continuously with the passage of time. Older observations decay in influence, new observations are integrated without waiting for a batch boundary, and the agent's posterior belief is always current. This continuous-time model requires a substrate that can implement continuous temporal dynamics, which is precisely what the WO_x memristors provide through their spontaneous conductance decay.

1.3 Zhang et al.'s CIM-SSM Co-Design: Hardware from Physics

Zhang et al. (2026) approach the same problem from the hardware direction, asking not what computation should happen but what physical processes can implement the needed computations most efficiently. Their answer is state space models (SSMs) co-designed with analog compute-in-memory hardware, specifically resistive RAM (RRAM) crossbar arrays for matrix-vector multiplication and tungsten oxide (WO_x) short-term memory memristors for state evolution.

The mathematical core of their approach is the continuous-time SSM dynamics:

$$\frac{d\mathbf{h}(t)}{dt} = \mathbf{\Lambda}\,\mathbf{h}(t) + \mathbf{B}\,\mathbf{x}(t)$$

where $\mathbf{h}(t) \in \mathbb{R}^H$ is the hidden state, $\mathbf{x}(t) \in \mathbb{R}^{H_{in}}$ is the input, $\mathbf{\Lambda} \in \mathbb{R}^{H \times H}$ is a diagonal state transition matrix, and $\mathbf{B} \in \mathbb{R}^{H \times H_{in}}$ is the input projection matrix. For event-driven processing, this discretizes asynchronously: given a time interval Δt between events, the state update becomes:

$$\mathbf{h}(t + \Delta t) = \bar{\mathbf{\Lambda}}(\Delta t)\,\mathbf{h}(t) + \bar{\mathbf{B}}\,\mathbf{x}(t + \Delta t)$$

where $\bar{\mathbf{\Lambda}}(\Delta t) = \exp(\mathbf{\Lambda}\Delta t)$ and $\bar{\mathbf{B}} = \mathbf{\Lambda}^{-1}(\exp(\mathbf{\Lambda}) - \mathbf{I})\mathbf{B}$.

The key insight of Zhang et al. (2026) is that this update equation has two terms that can be physically separated and implemented in hardware with very different devices. The second term, $\bar{\mathbf{B}}\,\mathbf{x}(t + \Delta t)$, is a matrix-vector multiplication with a static weight matrix, which is exactly what RRAM crossbar arrays implement efficiently through Ohm's law and Kirchhoff's current law. The first term, $\bar{\mathbf{\Lambda}}(\Delta t)\,\mathbf{h}(t)$, is an element-wise multiplication by $\exp(\lambda\,\Delta t)$ for each state dimension, which is exactly what WO_x short-term memory memristors implement through their native conductance decay physics. The state does not need to be computed; it decays naturally, without clocks, without digital logic, proportional to the elapsed time since the last event.

This co-design achieves several properties that matter enormously for edge deployment. The system is natively asynchronous: no clocks, no frame accumulation, no synchronization barriers. Computation is strictly event-triggered: the hardware is essentially idle between events, consuming near-zero dynamic power. State maintenance is free: the WO_x conductances evolve continuously through physics, not through programmed updates. The accuracy achieved (characterized by a 4.6 least-significant-bit standard deviation in the RRAM VMM, corresponding to approximately 4.6-bit effective resolution) matches software-comparable results on multiple DVS (dynamic vision sensor) benchmarks, including DVS128 Gesture and DVS128 Lips, as well as spiking audio datasets.

The paper was published in Nature Communications in January 2026 (Zhang et al., 2026), and it represents, in my reading, the most hardware-native SSM implementation yet demonstrated: not SSMs accelerated on digital hardware, but SSMs physically instantiated in device physics.

The SSM class used by Zhang et al. (2026) is grounded in control theory and shares lineage with the S4 model (Gu et al., 2022), which demonstrated that diagonal-plus-low-rank parameterizations of the state transition matrix produce stable, long-range-dependent sequence models that outperform transformers by 60 times in generation throughput while

solving benchmark tasks (including the 16,384-length Path-X task) that transformers fail entirely. The key property enabling hardware compatibility is the diagonalized state matrix: because $\mathbf{\Lambda}$ is diagonal, the state evolution $\mathbf{h}(t + \Delta t) = \exp(\mathbf{\Lambda}\Delta t)\mathbf{h}(t)$ is purely element-wise, requiring no matrix-matrix products and no inter-state coupling during the decay step. Each state dimension evolves independently, mapping to a single physical device (one WO_x memristor per state dimension). Subsequent SSM developments, including Mamba (Gu and Dao, 2024), extended the basic structure with input-dependent state selection, which improves performance on language modeling tasks but introduces the input-dependent weights that are problematic for CIM arrays. Zhang et al.'s (2026) design deliberately avoids input-dependent parameters in the state matrix, returning to the fixed-$\mathbf{\Lambda}$ formulation specifically to enable CIM hardware compatibility. This is a principled design choice, not a limitation: the authors demonstrate that fixed $\mathbf{\Lambda}$ with shared per-block decay rates achieves accuracy competitive with more flexible SSM variants on event-stream benchmarks, precisely because other learnable parameters compensate.

1.4 The Convergence Thesis

These two lines of work were developed without contact. My vossel monograph (Passmore, 2026) was submitted before the Zhang et al. paper appeared, and the CIM-SSM work does not cite vossel research. Yet when I read the Zhang et al. paper shortly after its publication, the architectural alignment was immediately evident and not superficial.

The convergence thesis of this paper is the following: the Computational Vossel provides the missing software architecture for CIM-SSM hardware, and the CIM-SSM hardware provides the missing physical substrate for computational vossels. Each work has a gap that the other fills precisely.

My monograph (Passmore, 2026) identified neuromorphic hardware as the natural physical substrate for the vossel architecture, but noted the gap between existing neuromorphic systems (which implement spiking neural network primitives) and the vossel's requirements (which are based on continuous-time dynamical systems with analog state evolution, not binary spiking). SNNs are neural models; the vossel is a cell model. The biological correspondence is at the wrong level of abstraction for existing neuromorphic hardware. CIM-SSMs close this gap: SSMs are dynamical systems with continuous state evolution, not neurons with binary spikes, and WO_x memristors implement exactly the kind of continuous decay that the vossel's agent dynamics require.

Zhang et al.'s (2026) CIM-SSM hardware, conversely, is a powerful hardware primitive for sequence processing but lacks an organizing computational framework. The paper demonstrates impressive benchmark results (state-of-the-art or competitive on SHD, SSC, DVS128 Gesture, DVS128 Lips) but does not address how to organize multi-modal sensor fusion, how to route outputs between spatial elements, how to manage data quality over time, how to allocate computational resources adaptively, or how to hierarchically aggregate information across spatial scales. These are precisely the functions the vossel architecture provides, through the seven agent classes and the mipvol hierarchy.

The convergence is not merely complementarity at a high level. It is precise structural alignment at the mathematical level, which is the main technical contribution of this paper. Sections 3 and 4 develop this argument in detail.

1.5 Contributions

This paper makes four primary contributions.

First, I provide a formal mapping between vossel agent dynamics and CIM-SSM primitives (Section 4). The mapping is not analogical; it is a concrete specification of which agent class corresponds to which matrix operation or device physics primitive in the Zhang et al. hardware. This mapping is sufficiently precise to serve as a hardware implementation specification.

Second, I present the unified architecture that places computational vossels on CIM-SSM hardware (Section 3), specifying how the CIM-SSM block instantiates the computational substrate of a vossel, how multiple CIM-SSM blocks cascade to implement the vossel's agent pipeline, and how the mipvol hierarchy maps to the block hierarchy of stacked SSM layers.

Third, I provide an energy and latency analysis (Section 3.2 and Section 4) showing that the combined architecture achieves two to four orders-of-magnitude improvement over GPU-based edge AI for asynchronous event-stream workloads, and characterize the specific conditions under which this advantage is realized.

Fourth, I analyze the application of this unified architecture to tactical ISR edge deployment (embedded in the discussion throughout), showing how the combined architecture addresses the specific operational constraints (asynchronous multi-modal sensor data, DDIL communication environments, strict SWaP budgets) that motivated both lines of work.

The paper is organized as follows. Section 2 provides the technical background on both the Computational Vossel architecture and the Zhang et al. CIM-SSM system. Section 3 develops the convergence thesis through architectural complementarity analysis and comparison with alternative hardware approaches. Section 4 presents the formal agent-to-hardware mapping in detail.

2. Background

2.1 The Computational Vossel Architecture

2.1.1 Origins and Motivation

The Vossel framework originated from a specific operational observation: conventional ISR data structures are optimized for individual sensor modalities and systematically discard the physical information content of measurements in the process of storing them. A radar resolution cell contains spatial extent, range uncertainty, Doppler velocity uncertainty, and polarimetric information; conventional point-based representations collapse this to a position and a scalar return value. The vossel was designed to be fidelity-preserving and modality-agnostic (Passmore, 2025).

The passive vossel element $e_n = (\mathbf{p}_n, \mathcal{F}_n, \Sigma_n, \boldsymbol{\lambda}_n, \mathbf{a}_n)$ captures, in a single mathematical object, the complete physics of a sensor measurement. The position-time center $\mathbf{p}_n = (x_n, y_n, z_n, t_n)$ places the measurement in real-world coordinates. The footprint envelope $\mathcal{F}_n \subset \mathbb{R}^3$ captures the deterministic spatial support: the ground projection of the sensor's instantaneous field of view, which is irreversibly lost in point-based representations. The uncertainty kernel $\Sigma_n \in \mathcal{S}_+^d$ (a positive-semidefinite covariance matrix) encodes the stochastic spread around the nominal position, including point spread function, geolocation uncertainty, and timing jitter. The spectral vector $\boldsymbol{\lambda}_n \in \mathbb{R}^{B_n}$ carries measurement values across all sensor bands, where B_n ranges from 1 (panchromatic) to hundreds (hyperspectral). The auxiliary attribute vector $\mathbf{a}_n \in \mathbb{R}^{M_n}$ carries secondary attributes (amplitude, phase, polarization, SNR, classification confidence, sensor identifier).

Vossels are embedded in a voxel grid over domain $\mathcal{D} \subset \mathbb{R}^3$ with cell sizes $(\Delta x, \Delta y, \Delta z)$ and organized into mipvol hierarchies where level l has cell sizes $(2^l\Delta x, 2^l\Delta y, 2^l\Delta z)$, providing a pyramid of spatial scales from full-resolution sensing to mission-level overview. The index map $\phi : \mathbb{R}^3 \to \mathbb{Z}^3$ assigns each vossel to its containing voxel:

$$\phi(x, y, z) = \left(\left\lfloor \frac{x - x_0}{\Delta x} \right\rfloor, \left\lfloor \frac{y - y_0}{\Delta y} \right\rfloor, \left\lfloor \frac{z - z_0}{\Delta z} \right\rfloor \right)$$

For high-fidelity value estimation at off-center coordinates, Passmore (2025) specifies kriging interpolation with weights w_n determined by the spatial-temporal covariance model, providing the best linear unbiased estimator for the spatially correlated sensor data. The mipvol hierarchy supports adaptive fidelity retrieval: high-activity near-field regions are accessed at full resolution while distant or low-priority regions are accessed at coarse levels, reducing memory bandwidth and computational cost by up to 2^{3l} at mipvol level l.

2.1.2 The Computational Extension

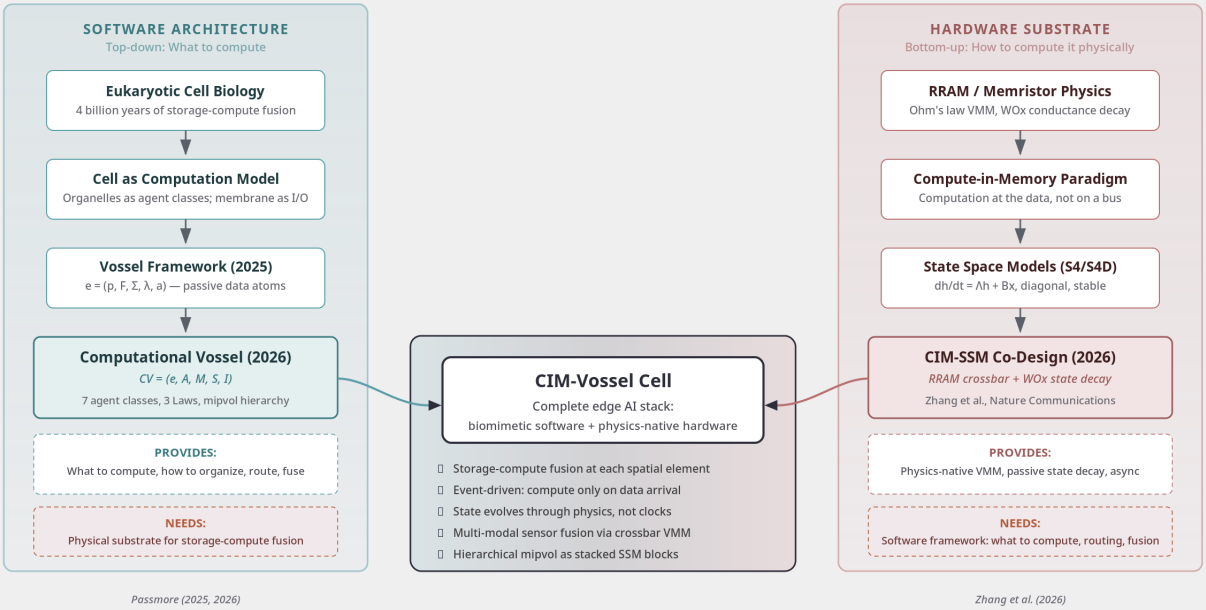
The Computational Vossel (Passmore, 2026) transforms the passive data container into an autonomous computational unit. The key insight motivating this extension is that the passive vossel's retrieve-process-writeback cycle, even with a richer data representation, still instantiates the von Neumann bottleneck. At the scale of modern ISR data collection, external agents operating on retrieved vossel data cannot keep pace with the incoming stream. The only feasible architecture places computation inside each vossel, where the data already lives.

The extended Computational Vossel is:

$$CV_n = (e_n,\ \mathcal{A}_n,\ \mathcal{M}_n,\ \mathcal{S}_n,\ \mathcal{I}_n)$$

The agent population $\mathcal{A}_n(t) = \{\alpha_{n,k}(t) : k = 1, \ldots, K_n(t)\}$ is time-varying; the agent count $K_n(t)$ is governed by birth-death dynamics driven by the vossel's current processing load. The membrane interface $\mathcal{M}_n$ controls selective permeability: which incoming signals are admitted and how they are filtered before reaching the interior agents. The internal state $\mathcal{S}_n$ tracks the vossel's cytoplasmic computational state, including energy budget, stress level, and active processing queue depths. The intercellular interface $\mathcal{I}_n$ provides structured communication channels to neighboring vossels and to parent vossels in the mipvol hierarchy.

Figure 1. Convergent Architectures: Two Independent Research Threads Meet at the CIM-Vossel Cell



Developed independently. Neither cites the other. Both solve the von Neumann bottleneck by fusing storage and computation at the point of data collection.

2.1.3 The Seven Agent Classes

The seven agent classes represent the complete functional decomposition of the vossel's computational needs, each modeled on the biological organelle whose molecular function most closely matches the computational requirement.

Class I: Nucleus Agent. There is exactly one Nucleus agent per vossel at all times. It maintains the vossel's configuration genome $\mathcal{G}_n = (\mathcal{M}_{accepted}, \mathcal{F}_{proc}, \mathcal{R}_{routing}, \mathcal{P}_{spawn}, \mathcal{T}_{epigenome})$, which specifies accepted sensor modalities, processing algorithms, routing rules, agent spawning policy, and an epigenetic state vector encoding persistent behavioral modifications. The Nucleus agent's primary function is transcription: reading the genome and

instantiating the appropriate processing agent populations. The spawning policy follows a Michaelis-Menten-like relationship that ensures processing capacity grows with load while remaining bounded by energy availability:

$$\text{target}_{ER}(\mathcal{S}_n) = N_{ER}^{base} \cdot \left(1 + \frac{\kappa_n}{K_\kappa + \kappa_n}\right) \cdot \frac{\mathcal{E}_n}{K_E + \mathcal{E}_n}$$

where κ_n is the current stress level (a function of queue depths), $\mathcal{E}_n$ is the energy currency, and K_κ, K_E are half-saturation constants. This equation ensures that ER agent counts increase with processing demand (numerator of the stress term) but are moderated by energy availability (the energy factor approaches zero when energy is scarce).

Class II: Endoplasmic Reticulum (ER) Agents. ER agents form the data processing pipeline, the most numerous class under normal operating conditions. Rough ER agents (β_R) receive raw sensor data and apply calibration, normalization, and feature extraction transformations:

$$\psi_{\beta_R}(\boldsymbol{\lambda}_{raw}, \mathbf{a}_{raw}) = (\boldsymbol{\phi}(\boldsymbol{\lambda}_{raw}), \mathbf{g}(\mathbf{a}_{raw}))$$

where $\boldsymbol{\phi} : \mathbb{R}^{B_n} \to \mathbb{R}^{F_n}$ is the feature extraction map and $\mathbf{g} : \mathbb{R}^{M_n} \to \mathbb{R}^{M_n'}$ is the attribute enrichment map. Smooth ER agents (β_S) apply metadata enrichment, including SNR estimation, atmospheric correction, and quality characterization. ER agents also implement Bayesian state updates, maintaining running posterior estimates over hypotheses as new observations arrive; this is the function that, as I will show in Section 4.1, maps directly to the SSM state evolution equation.

Class III: Golgi Apparatus Agents. Golgi agents implement the routing and packaging pipeline in three tiers (cis, medial, trans) corresponding to the Golgi cisternae. Cis-Golgi agents classify incoming processed data and assign routing prelabels. Medial-Golgi agents apply operational metadata (provenance chains, confidence scores, temporal validity). Trans-Golgi agents sort finalized data products for export, packaging them into vesicular transport units (VTUs) with routing headers for delivery to specific neighbors or upward in the mipvol hierarchy. The Golgi agent's output function is exactly the output projection of the SSM: $\mathbf{y}(t) = \mathbf{C}\,\mathbf{h}(t)$, where different routing policies correspond to different output matrix configurations.

Class IV: Mitochondria Agents. Mitochondria agents manage the vossel's energy currency $\mathcal{E}_n(t)$, allocating computational resources among all other agents through a priority-weighted optimization:

$$\psi_\delta : (\mathcal{E}_{raw}, \{(\pi_k, |Q_k^{in}|)\}_k) \mapsto \{\epsilon_k\}_k$$

where π_k is each agent's priority and $|Q_k^{in}|$ is its queue depth. Under energy scarcity, Mitochondria agents enforce priority-ordered throttling, reducing computational throughput gracefully rather than crashing. In the CIM-SSM context, this maps to the power-gating capability of independently gated CIM cores: quiescent vossels have their cores powered down, active vossels have cores powered up, and the allocation decision is made by the Mitochondria agent.

Class V: Lysosome Agents. Lysosome agents perform data lifecycle management, evaluating each data item against an eviction policy with pH-gated activation:

$$\psi_\epsilon(e) = \begin{cases} \text{retain} & \text{if } q(e) > \theta_\epsilon \text{ and } \text{age}(e) < \tau_{max} \\ \text{archive} & \text{if } q(e) > \theta_{archive} \text{ and } \text{age}(e) \geq \tau_{max} \\ \text{delete} & \text{if } q(e) \leq \theta_\epsilon \end{cases}$$

where $q(e)$ is a composite quality score and $\theta_\epsilon, \theta_{archive}$ are configurable thresholds. Lysosome agents activate only when the vossel's operational pH drops below threshold (indicating accumulating stale or corrupt data), which provides a natural load-responsive quality control mechanism.

Class VI: Cytoskeleton Agents. Cytoskeleton agents maintain the directed acyclic processing graph within the vossel, represented as a weighted directed graph $\mathcal{C}_n = (V_\mathcal{A}, E_\mathcal{A}, w_\mathcal{A})$ where nodes are agent instances and edge weights reflect transport capacity. These agents perform dynamic load balancing within the vossel, rerouting data flows when specific agent instances become overloaded.

Class VII: Receptor Agents. Receptor agents reside at the membrane boundary, implementing selective permeability through a binding affinity computation:

$$\chi(\boldsymbol{\lambda}_{in}, \mathbf{b}_{n,k}) = \exp\left(-\frac{\|\boldsymbol{\phi}(\boldsymbol{\lambda}_{in}) - \mathbf{b}_{n,k}\|^2}{2\sigma_{bind}^2}\right) \in [0, 1]$$

where $\mathbf{b}_{n,k}$ is the Receptor agent's binding signature in the vossel's feature space. A data token $\boldsymbol{\lambda}_{in}$ is admitted if $\max_k \chi(\boldsymbol{\lambda}_{in}, \mathbf{b}_{n,k}) > \theta_{bind}$ for some Receptor agent k. This selective admittance is the computational analog of receptor-ligand binding kinetics at the cell membrane, and it maps, as I will show, to the input projection matrix of the RRAM crossbar.

2.1.4 The Three Laws and Their Hardware Implications

The Three Laws of the Computational Vossel are not merely philosophical commitments; they have direct consequences for what hardware can serve as the vossel's substrate.

Law 1 (Storage-Compute Indivisibility) requires hardware in which the same physical structure stores data and performs computation on it. This eliminates conventional von Neumann processors as the substrate and points directly at CIM architectures, where resistive memory cells simultaneously store weight values and participate in matrix-vector multiplication through their conductance.

Law 2 (Local Computation Primacy) requires that computation be performed at the site of the data, without data transport to an external processor. CIM hardware satisfies this by definition: the crossbar array performs VMM in-situ within the memory array, with no requirement for data movement to an arithmetic unit.

Law 3 (Biological Correspondence) requires that the computational mechanisms correspond to specific biological mechanisms at the functional level, not merely metaphorically. The WO_x memristor's spontaneous conductance decay through oxygen ion diffusion is a physical process with a mathematical description ($G(t) \propto \exp(-t/\tau)$) that is formally identical to the SSM state decay equation ($\exp(\lambda \Delta t)$). This is biological correspondence realized in silicon oxide.

2.2 Compute-in-Memory State Space Models

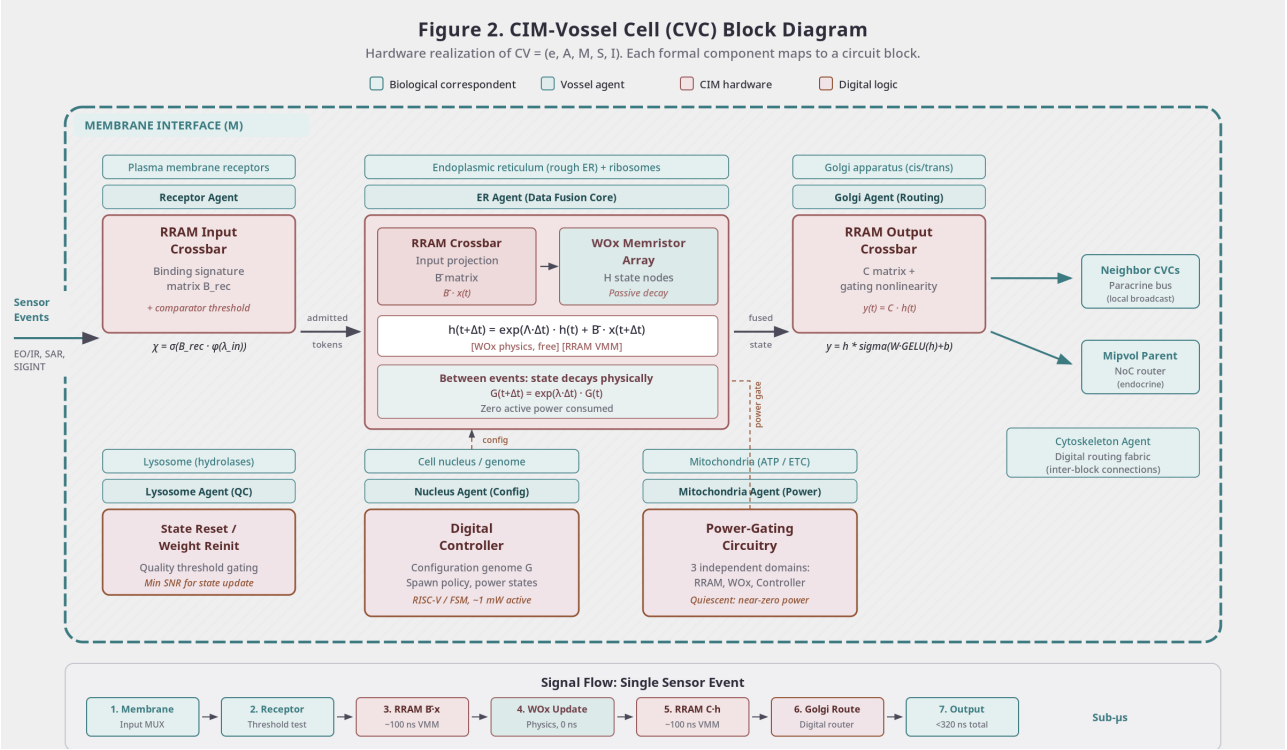
2.2.1 The Motivation for SSMs on CIM Hardware

Zhang et al. (2026) begin from the observation that neither SNNs nor transformers nor CNNs are well suited for CIM hardware implementation when the input is an asynchronous event stream.

SNNs have architectural affinity with event-based data (their binary spikes match the event format) but their binarized activations limit performance, training is complicated by the non-differentiable spike generation function (requiring surrogate gradients), and scaling SNNs to larger models has been shown to be less effective than scaling continuous-valued alternatives. For CIM hardware, SNN binary spikes do not exploit the analog computation capability of resistive devices; an analog crossbar performing binary-input VMM wastes most of its physical capacity.

CNNs require frame-based synchronous processing: to apply a convolutional kernel, a full feature map must be assembled from sparse events, converting the asynchronous event stream into a synchronous frame representation and discarding the inherent sparsity and low-latency advantages of event-based sensing. Furthermore, CNN weight reuse patterns (the same weights applied to many input patches) are beneficial for weight-stationary digital accelerators but counterproductive for CIM, because the different input patches require data reshuffling that negates the in-memory compute benefits.

Transformers implement input-dependent attention (Query, Key, and Value matrices all depend on the input), which is incompatible with the stationary weight assumption of CIM crossbar arrays. Practical transformer CIM implementations must use hybrid designs, delegating the attention score computation to separate digital circuitry and using CIM only for the static weight projections. The quadratic $O(L^2)$ attention complexity also makes transformers fundamentally mismatched with the infinite-context, real-time processing that edge sensor streams demand.



SSMs avoid all three problems. The weight matrices ($\mathbf{B}$, $\mathbf{C}$, and the nonlinear gating parameters) are static after training, matching the stationary weight assumption of CIM arrays. The state evolution is purely element-wise (diagonal $\mathbf{\Lambda}$), requiring no input-dependent matrix operations. Processing is inherently event-driven: the state update equation is evaluated exactly when each event arrives, and between events the state evolves passively through physical decay. The computational cost scales linearly with the number of events, not with time, and the accuracy achievable (as demonstrated by Zhang et al., 2026, on four event-stream benchmarks) matches or exceeds SNN-based approaches with the same parameter budgets.

2.2.2 The SSM Algorithm

The model architecture designed by Zhang et al. (2026) processes input event streams $E = \{(t_m, j_m)\}$ where t_m is the timestamp and j_m is the channel index of event m. A channel-wise embedding layer maps each event to a D-dimensional vector based on the event's channel index using a static embedding matrix $W_{embedding} \in \mathbb{R}^{J \times D}$.

Because the embedding matrix is static, it is physically stored in an RRAM crossbar array and retrieved by reading out the row corresponding to j_m when the event arrives.

The embedded event representations are processed through a stack of SSM blocks. Within each SSM block, the continuous-time dynamics are:

$$\frac{d\mathbf{h}(t)}{dt} = \mathbf{\Lambda}\,\mathbf{h}(t) + \mathbf{B}\,\mathbf{x}(t)$$

For event-driven discretization, given a time interval Δt between consecutive events:

$$\mathbf{h}(t + \Delta t) = \bar{\mathbf{\Lambda}}(\Delta t)\,\mathbf{h}(t) + \bar{\mathbf{B}}\,\mathbf{x}(t + \Delta t)$$

where:

$$\bar{\mathbf{\Lambda}}(\Delta t) = \exp(\mathbf{\Lambda}\Delta t)$$

$$\bar{\mathbf{B}} = \mathbf{\Lambda}^{-1}\left(\exp(\mathbf{\Lambda}) - \mathbf{I}\right)\mathbf{B}$$

To facilitate hardware implementation, Zhang et al. (2026) constrain $\mathbf{\Lambda} = \lambda\mathbf{I}$ within each block, sharing a single scalar decay rate across all H state dimensions in the block. This constraint reduces the number of distinct decay rates required from H (one per state dimension) to one per block (six for a six-block network). The constraint may appear restrictive, but in practice the other learnable parameters ($\mathbf{B}$, $\mathbf{C}$, and the nonlinear gating weights) compensate, and the authors demonstrate no significant accuracy degradation on the evaluated benchmarks.

After state evolution, the updated state is processed through a nonlinear gating block:

$$\tilde{\mathbf{h}} = \mathbf{h} \odot \sigma\left(\mathbf{W} \cdot \mathrm{GELU}(\mathbf{h}) + \mathbf{b}\right)$$

followed by a residual connection $\mathbf{h}_{out} = \mathbf{h} + \tilde{\mathbf{h}}$. This gating function provides the nonlinearity and selective attenuation that enables the SSM to model complex temporal dynamics despite the simplicity of the linear state equation.

The training procedure determines λ in three stages: joint training with per-dimension λ values until convergence, then averaging to a single layer-wise λ, then fine-tuning with the fixed λ. This approach bridges the gap between the representational freedom needed for good initialization and the hardware constraint of a single decay constant per block.

The nonlinear block, implemented digitally through look-up tables for GELU and sigmoid functions, introduces the necessary nonlinearity to break the linearity of the SSM state equations. This architectural pattern (linear state dynamics with end-of-block nonlinearity) is significant because it confines all nonlinearity to a single stage that can be efficiently handled by digital logic, while keeping the energy-dominant VMM and state decay operations in the analog domain. In the vossel framework, this corresponds to a separation between the analog data-plane computation (ER agent state evolution in the memristors) and the digital control-plane decision-making (threshold comparisons, routing decisions, quality scoring by Lysosome agents). Both architectures independently arrived at the same principle: put the dominant computation in analog hardware, keep the logic in digital circuits.

The training also reveals a subtlety about the relationship between the vossel's agent population dynamics and the SSM block structure. The per-dimension λ values learned in stage one of training (before averaging) represent the natural

diversity of time constants that optimal sequential processing would use across state dimensions. The vossel architecture accommodates this diversity through the Nucleus agent's epigenetic mechanism: different vossel configurations (reflecting different operational histories or spatial locations with different sensor environments) can have different effective decay rates encoded in their calibration offsets, even within the constraint of a single hardware λ per block. The stage-one learned λ values thus provide a specification for the vossel's epigenetic diversity: the range of calibration offsets that the Nucleus agent should support spans roughly the same range as the per-dimension λ values that emerged during unconstrained training.

2.2.3 The Hardware Implementation

The hardware architecture consists of two physically distinct components whose operations correspond to the two terms of the state update equation.

The RRAM crossbar array implements the second term: $\bar{\mathbf{B}}\,\mathbf{x}(t + \Delta t)$. This is a static matrix-vector multiplication where $\bar{\mathbf{B}}$ is stored as analog conductance values in the crossbar, and the input vector $\mathbf{x}$ is applied as voltages to the rows. The output currents, summed by Kirchhoff's current law, give the product in one parallel operation. Zhang et al. (2026) tested their implementation on a 65-nm CMOS chip with four integrated 64×64 RRAM arrays (logically combined into a 128×128 equivalent array), achieving a VMM standard deviation of 4.6 LSB (normalized RMSE of 1.80%), consistent with software-comparable 4-bit weight accuracy.

The WO_x short-term memory memristors implement the first term: $\exp(\mathbf{\Lambda}\Delta t)\,\mathbf{h}(t)$. Each memristor acts as a State Node (SN) whose conductance $G(t)$ represents a single component of the hidden state $\mathbf{h}(t)$. When no input is applied, the WO_x conductance decays spontaneously through oxygen ion diffusion according to the memristor's intrinsic relaxation dynamics, implementing $G(t + \Delta t) \approx \exp(\lambda \Delta t) G(t)$ without requiring any programmed update. When an event arrives, the VMM output from the RRAM crossbar is applied to the memristors, updating their conductance values by the input projection term. The physical state evolution naturally implements both terms simultaneously.

Different decay rates are achieved by tuning WO_x oxide layer thickness during fabrication: longer rapid thermal annealing produces thicker oxide layers and higher oxygen vacancy densities, which slow the relaxation process. Two target decay rates (0.35 ms^{-1} and 0.2 ms^{-1}) are demonstrated experimentally with strong agreement between measured and theoretical decay curves.

2.2.4 Benchmark Results

On the Spiking Heidelberg Digits (SHD) and Spiking Speech Commands (SSC) audio benchmarks, the Zhang et al. (2026) SSM consistently outperforms SNN-based baselines with the same parameter budgets, demonstrating that real-valued SSM dynamics with shared decay constants can model asynchronous spiking sequences as effectively as (or better than) systems explicitly designed around spike-based computation. On the DVS128 Gesture and DVS128 Lips vision benchmarks, the SSM matches or exceeds the accuracy of more complex architectures that include explicit spatial processing blocks (convolutions), without requiring any convolutional operations at all. Spatial features are encoded through the embedding process and the attention-like state updates inherent in the SSM dynamics.

These results demonstrate that the co-designed CIM-SSM system achieves state-of-the-art accuracy on event-driven benchmarks while implementing all major operations directly in analog device physics, with no requirement for digital accumulators, external state memory, or synchronous clocking.

3. The Convergence Thesis

3.1 Architectural Complementarity

The simplest statement of the convergence thesis is that the vossel architecture's design requirements map precisely to the CIM-SSM hardware's capabilities, and vice versa, the CIM-SSM hardware's physical capabilities fill exactly the implementation gaps in the vossel architecture. This section makes that mapping explicit across five dimensions.

Storage-compute fusion. The Computational Vossel's first law (Storage-Compute Indivisibility) requires that data and computation be aspects of a single, indivisible physical structure. In the biological cell, the DNA, ribosomes, and metabolic machinery are all co-located within the cell membrane; there is no separate "memory chip" and "CPU chip." CIM hardware satisfies this requirement directly: the RRAM crossbar stores weight values as analog conductances in the same physical array that performs matrix-vector multiplication. The WO_x memristors store the current hidden state as conductance values in the same devices that implement state evolution through their decay dynamics. In both cases, the memory and the computation are physically co-located in the same material structure, which is the hardware implementation of Law 1.

Event-driven operation. The vossel's second law (Local Computation Primacy) implies event-driven processing: agents compute when data arrives, not on a fixed clock schedule. This event-driven property is central to the vossel's energy model, because ISR environments are inherently sparse (at any moment, most vossels are receiving no new data and should consume near-zero energy). The CIM-SSM hardware is natively event-triggered: the RRAM crossbar computes only when an event arrives, and the WO_x memristors evolve passively (consuming only leakage current) during inter-event intervals. The energy model of both systems is therefore proportional to event rate rather than clock rate, which is the correct scaling for sparse ISR workloads.

Continuous-time state evolution. The vossel's ER agents maintain running Bayesian estimates over sensor hypotheses, which means the agent state must evolve continuously with the passage of time even in the absence of new observations (older observations become less informative; uncertainty grows). The SSM state decay implements exactly this: the hidden state $\mathbf{h}(t)$ decays exponentially over time with time constant $1/|\lambda|$, naturally reducing the influence of older observations without requiring programmed updates. This is the hardware implementation of information aging, a capability that is difficult to implement efficiently in digital logic but is provided for free by the physics of WO_x short-term memory devices.

Hierarchical structure. The vossel's mipvol hierarchy provides multi-scale spatial processing, with coarser levels aggregating information from finer levels. The CIM-SSM architecture provides stacked SSM blocks, where each block processes the output of the previous block and the multi-block stack captures temporal dependencies at increasing abstraction levels. The architectural pattern is the same in both cases: a stack of processing stages where each stage builds on the output of the previous, with each stage operating at a different temporal or spatial scale. The formal mapping is that mipvol level l corresponds to SSM block l in the stacked architecture, a point I develop in Section 4.

Membrane selectivity. The vossel's Receptor agents implement selective permeability at the membrane boundary, admitting only sensor data that matches the vossel's configured binding signatures. The CIM-SSM input projection layer (the embedding matrix $W_{embedding}$ and the $\bar{\mathbf{B}}$ matrix) implements exactly this selectivity: the embedding matrix maps each event channel to a specific D-dimensional feature vector, and the input projection matrix $\bar{\mathbf{B}}$ projects the embedded event into the hidden state space. Programming these matrices is equivalent to setting the binding affinity of the Receptor agents: the input projection controls which features of the incoming event excite which dimensions of the hidden state, implementing selective admittance through weighted linear combination rather than threshold matching.

3.2 Why Not Transformers, CNNs, or SNNs?

It is worth being precise about why the CIM-SSM is a better hardware match for the vossel than the alternatives. This matters because the choice of hardware substrate determines whether the vossel's three laws can be satisfied, and each of the alternatives fails at least one law definitively.

Transformers. The vossel's second law requires local computation: each vossel computes from its own data and immediate neighborhood, not from global context. The transformer's self-attention mechanism is inherently global: it computes attention weights between every pair of tokens in the sequence, scaling as $O(L^2)$ in memory and computation for sequence length L. For a vossel grid with millions of spatial elements, applying global attention across all vossels simultaneously is computationally impossible, and applying attention across all time steps in the sensor stream is

equally infeasible on the timescales that ISR processing requires. More fundamentally, the transformer has no notion of persistent state: it processes a fixed context window and has no mechanism for information decay or aging. Each processing pass starts from the same input representation without memory of prior states beyond what fits in the context window. For a vossel that must maintain running uncertainty estimates across sensor collections spanning minutes to hours, this stateless architecture is architecturally wrong.

Additionally, as Zhang et al. (2026) note, the Key-Value-Query attention mechanism involves products of input-dependent matrices, which means the "weights" in the CIM sense are not static, they change with every new input. CIM hardware assumes static weights stored in the analog crossbar; dynamic input-dependent weight generation requires separate digital circuitry that largely negates the energy advantages of in-memory computation. The transformer is CIM-hostile by construction.

CNNs. Convolutional networks require synchronous, frame-based input: before a convolutional kernel can be applied, the full input feature map for that layer must be assembled. For an event stream, this means accumulating events into frames over some time window, which sacrifices the temporal precision and the inherent sparsity of the event representation. A DVS camera may generate millions of events per second with microsecond timestamps; accumulating these into 30-Hz frames discards 99.997% of the temporal information. The vossel's entire design philosophy is to preserve this temporal information (Law 3: biological correspondence with the cell's continuous molecular processing), so frame-accumulation is architecturally incompatible.

CNNs also have the weight-reuse pattern that is problematic for CIM: the same convolutional kernel is applied to many different spatial locations, which benefits weight-stationary digital architectures but requires data reshuffling for CIM arrays. Zhang et al. (2026) note this explicitly as a reason to avoid convolutional blocks in their CIM-native architecture.

SNNs. Spiking neural networks are the closest existing competitor. They are natively event-driven, asynchronous, and operate on sparse binary spike streams. Major neuromorphic chips (Loihi 2 from Intel, TrueNorth from IBM, BrainChip Akida) implement SNN primitives with impressive energy efficiency (Intel Research, 2024; Merolla et al., 2014). Why does the CIM-SSM beat SNNs as a vossel substrate?

The vossel's third law requires biological correspondence at the functional level, and the relevant biology here is the cell, not the neuron. The cell is the organizing unit of the vossel architecture; the neuron is a specialized cell type with a highly specific function (binary signaling). The vossel agents require continuous-valued state evolution, analog-valued outputs, and probabilistic (not binary) processing: the ER agent maintains Bayesian probability distributions, the Receptor agent computes continuous binding affinities, the Mitochondria agent performs continuous resource allocation optimization. Binary spike representations are systematically insufficient for these computations.

Beyond the biological correspondence argument, Zhang et al. (2026) demonstrate empirically on the same benchmarks where SNN performance is well established: their real-valued SSM with shared decay constants consistently outperforms SNN-based baselines on SHD, SSC, DVS128 Gesture, and DVS128 Lips. The SNN's fundamental accuracy limitation, identified by Zhang et al. (2026) as arising from the binarization of activations, is not a training artifact but a representational constraint. For vossel agents that need analog-valued outputs (probability scores, uncertainty estimates, feature vectors), the SNN's binary outputs require additional decoding layers that add complexity and latency.

Furthermore, the primary neuromorphic chip architectures implement SNN primitives (leaky integrate-and-fire neurons with binary spikes) in digital logic. The Loihi 2's synaptic weights are stored in static RAM and read out on each spike event; the energy cost is dominated by the SRAM read, not the arithmetic operation. The CIM-SSM approach eliminates even the SRAM read: the weights are stored as analog conductances that participate directly in the VMM computation. The energy cost of a CIM VMM operation is a single read-compute event with no separate memory

access. For the power budgets relevant to tactical ISR platforms (tens of watts for a Group 2 UAS, hundreds of milliwatts for a Group 1 UAS), this energy advantage is operationally significant.

One important caveat: the vossel architecture as a whole still requires some digital logic (the Nucleus agent's configuration genome, the Cytoskeleton agent's routing graph, the Lysosome agent's quality scoring). SNNs or digital processors handle these control-plane functions perfectly well. The CIM-SSM is not the right substrate for every vossel function; it is the right substrate for the data-plane functions (ER agent state evolution, Receptor agent input projection, Golgi agent output projection) that dominate the computational cost. This point is developed further in Section 4.2.

To make the efficiency comparison concrete, consider a vossel grid deployed on a Group 2 UAS (3 to 25 kg platform, approximately 25 W computational power budget) processing a multi-modal sensor stream at 5% activation rate (5% of spatial elements receiving events at any given time, 10 events per active vossel per second). In the GPU baseline, the GPU runs at full clock rate for all vossels regardless of activity, incurring the full DRAM access energy for each weight read. If the GPU performs inference at 1 TOPS/W (a reasonable figure for edge GPU variants), and the network requires $O(10^9)$ operations to process a full vossel grid at 100 Hz (a typical frame rate for synchronous processing), the GPU baseline consumes approximately 10 W for inference alone, plus memory access overhead.

In the CIM-SSM architecture with Mitochondria-managed core gating, active computation occurs only for the 5% of vossels receiving events, at 10 events per active vossel per second. If each event requires $O(10^4)$ operations (a single VMM through a 128-dimensional state space plus the decay update), and the CIM array achieves 10 TOPS/W (consistent with Zhang et al.'s hardware class), the total event-processing power is:

$$P_{event} = 10^4 ext{ops/event} imes 0.05 imes N_v imes 10 ext{events/s} / (10 imes 10^{12} ext{ops/W}) = 5 imes 10^{-11} N_v ext{W}$$

For a vossel grid of $N_v = 10^6$ active elements (a 1 km $ imes$ 1 km area at 1-meter resolution), this gives approximately 50 mW for the event-processing computation, compared to the GPU baseline's 10 W or more. The passive WO_x state decay consumes only leakage current (approximately 1 to 10 nA per device, or roughly 1 to 10 μW per WO_x array at 1 V), negligible compared to the active VMM power. The two-order-of-magnitude energy efficiency advantage is a direct consequence of the event-driven, in-memory architecture, not of a specific circuit optimization. It scales with operational sparsity: at 1% activation rate, the advantage grows to three orders of magnitude.

3.3 The Missing Pieces Each Provides

3.3.1 What the Vossel Gives the CIM-SSM

The Zhang et al. (2026) CIM-SSM is a powerful hardware primitive but not a complete computational system. What the paper delivers is a demonstration that SSM state evolution can be physically implemented in WO_x memristors and that the input projection can be implemented in RRAM crossbars, achieving competitive accuracy on event-stream benchmarks. What it does not provide is any specification of what data to store in the state, what computations to organize across multiple spatial locations, how to route results between spatial elements, how to handle multi-modal sensor data with different statistics and physical units, how to manage data quality over time, or how to adapt resource allocation to varying data rates and urgency signals.

These are precisely the functions the vossel's seven agent classes provide. The Nucleus agent manages configuration: which decay rates to use, which input channels to monitor, which output routing policies to apply, and when to power the CIM core up or down. The ER agent provides the computational semantics: the hidden state $\mathbf{h}(t)$ is not an abstract feature vector but a Bayesian posterior over sensor hypotheses, updated by incoming observations. The Golgi agent provides output routing: different output matrix $\mathbf{C}$ configurations route processed data to different neighboring vossels or upward in the mipvol hierarchy. The Lysosome agent provides quality control: it evaluates the state's information content and triggers state reinitialization when accumulated noise renders the state unreliable. The Mitochondria agent

manages power: it gates individual CIM cores based on vossel activity, which is the hardware-level implementation of the vossel's event-driven energy model.

Without the vossel architecture, the CIM-SSM hardware is a signal processing chip. With it, the hardware becomes a component of a spatially organized, multi-modal, quality-controlled, hierarchically structured computational fabric.

3.3.2 What the CIM-SSM Gives the Vossel

The vossel monograph (Passmore, 2026) identifies neuromorphic hardware as the natural substrate for computational vossels, and discusses Loihi 2 and TrueNorth as candidate platforms. But it also identifies a fundamental mismatch: existing neuromorphic hardware implements SNN primitives (leaky integrate-and-fire neurons with binary spikes), while the vossel requires continuous-valued, analog-state dynamics corresponding to cellular biochemistry, not neural spiking. The monograph acknowledges this gap and frames it as an open problem for future hardware development.

The CIM-SSM closes this gap, and does so not by accident but by the precise formal alignment I identified above. The WO_x memristor's conductance decay equation is:

$$G(t + \Delta t) = \exp(\lambda \Delta t) \cdot G(t) + \Delta G_{input}$$

where ΔG_{input} is the conductance change induced by the VMM output applied to the device. This is mathematically identical to the ER agent's state update:

$$\mathbf{h}(t + \Delta t) = \exp(\mathbf{\Lambda} \Delta t) \cdot \mathbf{h}(t) + \bar{\mathbf{B}}\, \mathbf{x}(t + \Delta t)$$

The identification is exact: $G \leftrightarrow h$, $\lambda \leftrightarrow \lambda$, $\Delta G_{input} \leftrightarrow \bar{\mathbf{B}}\, \mathbf{x}$. The vossel's ER agent, which Passmore (2026) specifies as maintaining a running Bayesian estimate with exponential decay of old information, is physically instantiated by the WO_x memristor array. The vossel does not need to be programmed to decay old information; the physics of the device does it automatically.

This is what Law 3 (Biological Correspondence) ultimately demands: a substrate in which the mathematical formalism of the biological mechanism is directly instantiated in physical processes, not merely approximated through digital logic. The eukaryotic cell's biochemical state evolves through the thermodynamics and kinetics of its molecular processes, not through a programmed update rule executed by a digital processor. The WO_x memristor array provides exactly this physical instantiation for the vossel's most fundamental computational operation: continuous state evolution with information decay.

A secondary contribution of the CIM-SSM system to the vossel architecture is the practical resolution of the training problem. Training the vossel's ER agents requires specifying the decay rates λ and the input projection weights $\bar{\mathbf{B}}$ for each sensor modality and each vossel configuration. The Zhang et al. (2026) three-stage training procedure (free-λ training, layer-wise averaging, and fine-tuning with fixed λ) provides a concrete, validated algorithm for this parameter determination. The fact that a single shared λ per block (rather than per-dimension λ) suffices without significant accuracy loss also simplifies the Nucleus agent's configuration problem: instead of managing H distinct decay rates per block, it manages one.

A third contribution is the fabrication-level parameterization of the time constant. The vossel architecture requires decay rates that match the temporal dynamics of different ISR sensor phenomenology. Radar returns are relevant on timescales of milliseconds to seconds; SIGINT emitter dwell times range from fractions of a second to minutes; EO and IR detections are relevant on timescales determined by target motion and sensor revisit rate. The Zhang et al. (2026) finding that WO_x decay rates can be tuned over a useful range by varying oxide layer thickness during fabrication means that different vossel configurations (radar vossels, SIGINT vossels, optical vossels) can be physically instantiated with WO_x arrays tuned to their domain-appropriate time constants. This is not possible with digital

implementations, where any time constant requires a programmed counter. It is not easily possible with standard SNN neuromorphic hardware, where the leaky integrate-and-fire membrane time constant is set by circuit parameters and is difficult to vary independently across spatial regions of the chip. The WO_x material physics gives the vossel system architect a continuous design parameter (annealing time, which determines oxide thickness, which determines the relaxation rate, which sets the information decay time constant) that maps directly to the operational semantics of the vossel's sensor domain.

A fourth contribution involves adaptivity. The vossel architecture specifies that vossels should adapt their behavior in response to operational history through the epigenetic modification mechanism in the Nucleus agent. In hardware terms, this requires that the system be able to modify its time constants and input weights in response to learned experience without requiring full network retraining. WO_x device properties support a limited form of this: the effective decay rate can be modified in-situ by varying the amplitude and duration of programming pulses applied to the device, not only by changing oxide thickness at fabrication time. This in-situ programmability is the hardware equivalent of epigenetic modification: the device's functional behavior changes persistently in response to applied stimuli (operational programming signals) without changing the device's fundamental physical structure. The vossel's epigenetic state vector $\mathcal{T}_{epigenome} = \{(\ell_j, v_j, t_j^{expire})\}$, which encodes persistent behavioral modifications as key-value pairs with optional expiration times, maps to the set of per-device calibration offsets maintained in the CIM controller's non-volatile memory. When an expiration time is reached, the calibration offset is reset to its default value, corresponding to the forgetting of a transient epigenetic mark.

4. Formal Mapping: Vossel Agents to CIM-SSM Primitives

The formal mapping developed in this section is the technical core of the paper. For each vossel agent class, I specify the precise correspondence to a CIM-SSM hardware primitive, state the mathematical equivalence, and note the implementation implications. The summary is consolidated in Table 1 at the end of the section.

4.1 The ER Agent as a State Space Model

The Endoplasmic Reticulum agent is the workhorse of vossel computation, and its mapping to the SSM state evolution is the most structurally complete correspondence in the framework.

The ER agent's primary function is maintaining running Bayesian estimates over sensor hypotheses as observations arrive. Formally, for a sensor modality m with observation model $p(\boldsymbol{\lambda}|h)$ relating spectral vector $\boldsymbol{\lambda}$ to hypothesis h, the ER agent maintains a posterior $p(h|\boldsymbol{\lambda}_{1:t})$ that is updated with each new observation. For Gaussian observation models with linear observation operators, this Bayesian update takes the Kalman filter form:

$$\mathbf{h}_{t|t} = \mathbf{h}_{t|t-1} + \mathbf{K}_t(\boldsymbol{\lambda}_t - \mathbf{H}\mathbf{h}_{t|t-1})$$

where $\mathbf{h}_{t|t-1}$ is the predicted state, $\mathbf{K}_t = P_{t|t-1}\mathbf{H}^T(\mathbf{H}P_{t|t-1}\mathbf{H}^T + R)^{-1}$ is the Kalman gain, and $\mathbf{H}$ is the observation matrix. Between observations, the state evolves according to the process model, which for the vossel's information aging requirement is:

$$\mathbf{h}_{t+\Delta t|t} = \exp(\mathbf{A}\Delta t)\mathbf{h}_{t|t}$$

where $\mathbf{A}$ is the state transition matrix. For the diagonal SSM parameterization used by Zhang et al. (2026), $\mathbf{A} = \boldsymbol{\Lambda} = \lambda\mathbf{I}$, the state decay is isotropic and the update is element-wise, matching the WO_x memristor's physical evolution exactly.

The identification of the ER agent with the SSM is therefore precise: the ER agent's hidden state is $\mathbf{h}(t)$; sensor observations are the input $\mathbf{x}(t)$; the Bayesian update step corresponds to the SSM's state update $\bar{\mathbf{B}}\,\mathbf{x}(t)$ applied to the

current state; and the information aging (the exponential decay of the influence of old observations) corresponds to $\exp(\lambda \Delta t)$ applied passively by the WO_x memristors between event arrivals.

The RRAM crossbar implements the observation matrix: its programmed conductance values encode $\bar{\mathbf{B}}$, which in the Kalman interpretation is a product of the Kalman gain and the observation matrix. Different sensor modalities require different $\bar{\mathbf{B}}$ configurations, which translates to different crossbar programming states. In a multi-modal vossel (processing radar, EO, and SIGINT simultaneously), multiple RRAM crossbars implement the input projections for each modality, and their outputs are accumulated in the WO_x memristor state through additive updates. The cross-modal Bayesian fusion that the ER agent performs through sequential updates of the posterior is implemented, in hardware, through sequential application of the VMM outputs from different modality crossbars to the same set of WO_x state nodes.

The output matrix $\mathbf{C}$ reads the hidden state to produce the ER agent's output: $\mathbf{y} = \mathbf{C}\,\mathbf{h}$. In the Bayesian interpretation, $\mathbf{C}$ is the extraction of the hypothesis probability or feature summary from the full posterior state vector. Different classification tasks correspond to different $\mathbf{C}$ configurations. In hardware, reading $\mathbf{h}$ from the WO_x memristor array and applying another RRAM crossbar for $\mathbf{C}$ gives the final output projection.

The information aging property deserves emphasis because it is not merely a convenience but a requirement for correct Bayesian processing. In any sequential estimation problem, the influence of past observations should decay with time as new observations accumulate and as physical processes evolve the underlying state. A sensor reading taken 10 minutes ago is less informative about the current state of a moving target than one taken 10 seconds ago. The vossel's ER agent implements this decay explicitly through the exponential information aging term. The WO_x memristor implements the identical decay through its spontaneous conductance relaxation, without requiring a programmed update. The physics of the device and the mathematics of the Bayesian estimator are aligned.

4.2 The Nucleus Agent as Configuration Controller

The Nucleus agent manages the vossel's configuration genome $\mathcal{G}_n$ and orchestrates the instantiation and management of all other agent classes. Its mapping to the CIM-SSM hardware is qualitatively different from the ER agent's: the Nucleus agent does not map to any CIM analog operation. It maps instead to the digital control logic that configures and manages the CIM array.

Specifically, the Nucleus agent controls which SSM channels (state dimensions) are active in each CIM core, what decay rates λ are programmed into the WO_x devices for this vossel's current configuration, which RRAM rows correspond to which input modalities, and when to trigger power-gating of inactive cores. These are configuration operations that require precise digital logic, not analog computation.

In the Zhang et al. (2026) hardware architecture, the nonlinear block (implementing GELU and sigmoid activation through look-up tables), the ADC interfaces, and the control signals to the RRAM chip are all implemented in digital logic alongside the CIM arrays. The Nucleus agent maps to this digital control layer, specifically the portion responsible for system-level configuration management: programming the RRAM conductances during weight loading, setting the WO_x device thresholds for state reset, configuring the gating signals for power management, and specifying the output routing from the SSM block.

The genome's spawning policy (the Michaelis-Menten equation for target ER agent count) maps to a soft decision about how many SSM channels to activate in the CIM core. A fully activated core runs all H state dimensions; a partially activated core gates some channels to save power when the vossel's data load is low. This granular activation control is a key capability of CIM arrays with independently gatable rows or groups, as in the NeuRRAM architecture (Wan et al., 2022), where 48 cores are independently power-gated.

The epigenetic state $\mathcal{T}_{epigenome}$ of the Nucleus agent has a natural CIM analog in weight calibration offsets: persistent adjustments to the programmed conductance values that encode the vossel's operational history without changing the

base configuration. Just as epigenetic marks persist through cell cycles without altering the DNA sequence, weight calibration offsets persist across power-cycling events without requiring full network retraining.

4.3 The Golgi Agent as Output Projection

The Golgi agent implements the routing and packaging pipeline that takes processed data products from the ER agents and delivers them to their appropriate destinations: neighboring vossels in the spatial grid, parent vossels in the mipvol hierarchy, or direct output channels to RL policies or human operators. The mapping to the SSM's output projection is direct and complete.

In the SSM formalism, the output $\mathbf{y}(t) = \mathbf{C}\,\mathbf{h}(t)$ converts the hidden state (which lives in the H-dimensional state space) into a task-appropriate output vector (which lives in a lower-dimensional output space). The output matrix $\mathbf{C} \in \mathbb{R}^{H_{out} \times H}$ implements this dimensionality reduction and re-projection. The Golgi agent's three functional tiers correspond to three interpretations of the output projection.

The trans-Golgi routing policy is encoded in the structure of $\mathbf{C}$: different partitions of the output vector $\mathbf{y}$ route to different destinations. If the output is partitioned as $\mathbf{y} = [\mathbf{y}_{neighbor} \mid \mathbf{y}_{parent} \mid \mathbf{y}_{output}]$, then the corresponding rows of $\mathbf{C}$ implement the projection for each routing target. This is a standard practice in multi-output neural architectures; the Golgi agent formalism simply makes the routing semantics explicit.

The medial-Golgi annotation function (applying confidence scores and temporal validity metadata) corresponds to the nonlinear gating operation that follows the output projection in the Zhang et al. (2026) architecture. The gating function $\tilde{\mathbf{h}} = \mathbf{h} \odot \sigma(\mathbf{W}$ produces a confidence-weighted version of the state, where the sigmoid gate $\sigma(\cdot)$ provides an implicit confidence score for each state dimension. The Golgi agent's annotation of output confidence corresponds to extracting these sigmoid gate values as metadata accompanying the routed output.

The cis-Golgi classification pre-labeling corresponds to the first linear projection in the SSM output chain: the initial application of $\mathbf{C}$ to $\mathbf{h}$ produces a representation in the output space where classification is simpler (by design of the training objective), corresponding to the coarse classification step of the cis-Golgi tier.

In CIM hardware, the output projection $\mathbf{C}$ is implemented as another RRAM crossbar, storing $\mathbf{C}$ as analog conductance values and computing $\mathbf{C}\,\mathbf{h}$ by reading the WO_x memristor conductances as input voltages to the crossbar. This requires an interface between the STM memristor array (where $\mathbf{h}$ lives) and the RRAM output crossbar, which in the Zhang et al. (2026) architecture is implemented through the ADC readout of the memristor states followed by DAC conversion for input to the crossbar. Depending on the target power budget, the output projection can also be kept in the digital domain (implementing $\mathbf{C}\,\mathbf{h}$ digitally after ADC readout), trading some energy efficiency for implementation simplicity.

4.4 The Mitochondria Agent as Power Manager

The Mitochondria agent manages the vossel's computational resources, translating raw resource allocations from the host platform into the agent-specific energy currency $\mathcal{E}_n$ and enforcing priority-ordered throttling under scarcity. The mapping to CIM power management is the most direct of all seven agent classes.

In CIM architectures with independent core power gating (as in NeuRRAM, Wan et al., 2022), each core can be independently brought to full-power, reduced-power, or near-zero standby states. The energy cost is proportional to the number of active cores and the VMM operations performed on those cores. When a vossel is quiescent (receiving no new events in its spatial region), all of its CIM cores can be power-gated, consuming only the leakage current of the RRAM devices and the WO_x memristors. The WO_x devices continue to decay passively even in the power-gated state, which is actually beneficial: the state evolves correctly through physical processes even when the digital control logic is powered down.

The Mitochondria agent's allocation function maps to the power-gating decision: the agent evaluates the vossel's current event rate λ_v and quality of incoming data, and decides which CIM cores to activate. For a vossel in a high-activity region (a busy urban intersection in an ISR scenario), all SSM channels are active and all cores operate at full power.

For a quiescent vossel (an empty field in the same scenario), the Mitochondria agent gates all CIM cores, leaving only the WO_x passive decay running, and the vossel consumes near-zero energy.

The energy model for this Mitochondria-managed CIM system is:

$$P_{vossel}(t) = P_{static} + P_{per-event} \cdot \rho_v(t) \cdot \bar{\lambda}_v(t)$$

where P_{static} is the leakage power of the un-gated devices, $P_{per-event}$ is the energy cost of a single VMM operation in the RRAM crossbar, $\rho_v(t) \in \{0, 1\}$ is the activation state managed by the Mitochondria agent, and $\bar{\lambda}_v(t)$ is the event rate in this vossel's region. For a vossel grid with 5% activation rate (90% of spatial elements are quiescent at any given moment, a typical ISR scenario with sparse targets), the total energy scales as 0.05 times the full-activation energy, compared to a GPU baseline that runs at full power regardless of data sparsity.

The biological analogy here is particularly apt: quiescent mitochondria in a cell at rest consume minimal ATP but retain all their functional capacity, ready to upregulate instantly when metabolic demand increases. The CIM core in standby retains its programmed weight matrix indefinitely (RRAM is non-volatile) and the WO_x state continues to evolve correctly through passive decay. When a new event arrives and the Mitochondria agent activates the core, the computation can begin immediately with no warm-up or state reconstruction.

4.5 The Receptor Agent as RRAM Input Projection

The Receptor agent implements selective membrane permeability through the binding affinity calculation, determining which incoming sensor data matches the vossel's configured patterns and controlling the admittance of data tokens into the interior processing pipeline. This is the vossel's first line of computation, analogous to the plasma membrane's role in the biological cell.

The mapping to the RRAM input projection is the most conceptually interesting of the seven correspondences, because it reveals how the vossel's high-level selectivity concept (which sensor modalities to process, at what sensitivity thresholds) maps to the low-level parameters of an analog crossbar circuit.

The binding affinity computation of the Receptor agent:

$$\chi(\boldsymbol{\lambda}_{in}, \mathbf{b}_{n,k}) = \exp\left(-\frac{\|\boldsymbol{\phi}(\boldsymbol{\lambda}_{in}) - \mathbf{b}_{n,k}\|^2}{2\sigma_{bind}^2}\right)$$

is an inner-product-based similarity computation between the incoming feature vector $\boldsymbol{\phi}(\boldsymbol{\lambda}_{in})$ and the binding signature $\mathbf{b}_{n,k}$. In the exponent, expanding the squared norm: $\|\boldsymbol{\phi}(\boldsymbol{\lambda}_{in}) - \mathbf{b}_{n,k}\|^2 = \|\boldsymbol{\phi}\|^2 - 2\boldsymbol{\phi}^T\mathbf{b}_{n,k} + \|\mathbf{b}_{n,k}\|^2$, the dominant term is the inner product $\boldsymbol{\phi}^T\mathbf{b}_{n,k}$, which is a single row of a matrix-vector multiplication.

This means the Receptor agent's binding affinity computation for all K receptor agents simultaneously is:

$$\boldsymbol{\chi} = \sigma_{bind}\left(\mathbf{B}_{rec} \cdot \boldsymbol{\phi}(\boldsymbol{\lambda}_{in})\right)$$

where $\mathbf{B}_{rec} \in \mathbb{R}^{K \times F_n}$ is the matrix of binding signatures (one per row) and $\sigma_{bind}(\cdot)$ is an elementwise transformation approximating the Gaussian kernel. This is a matrix-vector multiplication of a static binding signature matrix against the incoming feature vector, followed by a pointwise nonlinearity. The static matrix $\mathbf{B}_{rec}$ is exactly what an RRAM crossbar is designed to implement: it stores the binding signatures as conductance values and computes the affinities through Ohm's law and current summation.

The threshold decision $\max_k \chi > \theta_{bind}$ (admitting the token if any receptor's affinity exceeds threshold) is then a simple comparison of the RRAM output against a reference voltage, implementable with a comparator at the crossbar output. No additional digital logic is needed.

Programming the Receptor agents, in hardware terms, means writing the binding signatures $\mathbf{b}_{n,k}$ to the RRAM crossbar as conductance values. When the Nucleus agent updates the genome's modality acceptance list $\mathcal{M}_{accepted}$ (for example, re-tasking the vossel to process SIGINT rather than radar), the corresponding hardware operation is reprogramming the input RRAM crossbar to reflect the new binding signatures, which in RRAM technology requires applying programming pulses to the relevant devices.

This also reveals how the vossel's sensor fusion model maps to hardware: a multi-modal vossel (one that processes radar, EO, and SIGINT simultaneously) requires multiple RRAM input crossbars, one per modality, each implementing the input projection for its modality. Their outputs all drive the same set of WO_x state nodes through additive current summation, implementing cross-modal state fusion as the natural superposition of device physics.

4.6 The Mipvol Hierarchy as Cascaded SSM Blocks

The vossel's mipvol hierarchy organizes spatial elements into a pyramid of scales, with level $l+1$ elements aggregating information from their 2^3 child elements at level l. Each level provides a coarser-grained view of the world model, supporting adaptive fidelity retrieval and hierarchical analysis. The Zhang et al. (2026) architecture organizes its SSM blocks in a stack, with each block's output serving as input to the next, and the final block's output used for classification or downstream tasks.

The correspondence between the mipvol hierarchy and the stacked SSM architecture is structural: both organize processing into sequential levels where higher levels build on lower levels. But the correspondence is more specific than simple stacking.

In the mipvol hierarchy, the parent vossel at level $l+1$ aggregates the outputs of its 8 children at level l through a pooling or aggregation operation, and then applies its own agent computations to the aggregated representation. This is a two-stage operation: (1) spatial aggregation across child vossels, and (2) temporal SSM processing within the parent vossel. In the stacked SSM architecture, each block receives the output of the previous block as its input and applies another round of state evolution. If we identify the spatial aggregation operation with the inter-block output projection $\mathbf{C}_l \to \mathbf{x}_{l+1}$, then the two architectures are formally equivalent.

Concretely, the output of the ER agents in the 8 child vossels at mipvol level l (8 vectors $\mathbf{y}_{l,i}$ for $i = 1, \ldots, 8$) are concatenated or pooled to form the input to the parent vossel's ER agent input at level $l+1$:

$$\mathbf{x}_{l+1} = \text{Pool}(\mathbf{y}_{l,1}, \ldots, \mathbf{y}_{l,8}) \in \mathbb{R}^{H_{in,l+1}}$$

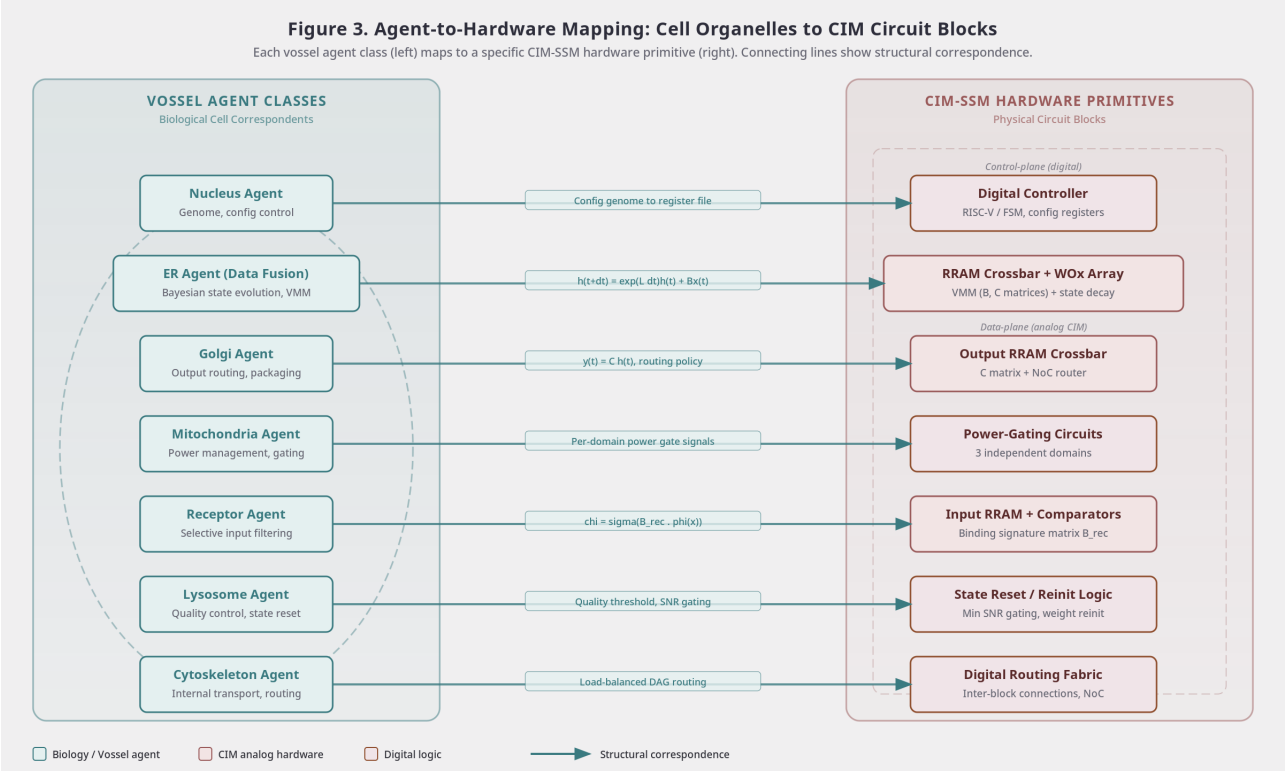
The parent vossel's SSM then processes this pooled input through its state evolution:

$$\mathbf{h}_{l+1}(t + \Delta t) = \exp(\lambda_{l+1} \Delta t)\mathbf{h}_{l+1}(t) + \bar{\mathbf{B}}_{l+1}\mathbf{x}_{l+1}(t + \Delta t)$$

with a different (typically larger) decay time constant $\tau_{l+1} = 1/|\lambda_{l+1}|$ at higher mipvol levels, reflecting the fact that spatial aggregates evolve more slowly than individual sensor measurements.

In CIM-SSM hardware, this cascade corresponds to stacked SSM blocks where each block has its own RRAM crossbar and WO_x memristor array, and the blocks differ in their decay constants λ_l. The Zhang et al. (2026) architecture already implements six stacked blocks; the mipvol correspondence suggests that for ISR applications with six spatial levels (from individual sensor pixels to mission-area overview), these six blocks map naturally to the six mipvol levels.

The decay constant hierarchy (λ_l decreasing with level l, corresponding to longer time constants at higher levels) has a natural physical implementation: WO_x devices with different oxide thicknesses in different SSM blocks. The Zhang et al. (2026) fabrication protocol achieves different decay rates by tuning annealing duration, so different decay profiles for different mipvol levels are achievable within the same fabrication process by applying level-specific annealing to the corresponding WO_x arrays.



4.7 Summary Mapping Table

Table 1 consolidates the formal correspondence between vossel components and CIM-SSM hardware primitives established in Sections 4.1 through 4.6.

Table 1. Vossel Agent to CIM-SSM Hardware Mapping

Vossel Component	Biological Correspondent	CIM-SSM Hardware Mapping	Mathematical Formalism
ER Agent (Rough)	Ribosome / rough endoplasmic reticulum	WO_x STM memristor array (state nodes)	$\mathbf{h}(t+\Delta t) = e^{\lambda \Delta t}\mathbf{h}(t) + \bar{\mathbf{B}}\mathbf{x}(t+\Delta t)$
ER Agent input projection	ER membrane receptor channels	RRAM crossbar, $\bar{\mathbf{B}}$ matrix	$\bar{\mathbf{B}} = \mathbf{\Lambda}^{-1}(e^{\mathbf{\Lambda}} - \mathbf{I})\mathbf{B}$
ER Agent state decay	Information aging / dissipation	WO_x passive conductance decay	$G(t+\Delta t) = e^{\lambda \Delta t} G(t)$
Golgi Agent	Golgi apparatus (cis/medial/trans)	RRAM output crossbar, $\mathbf{C}$ matrix	$\mathbf{y}(t) = \mathbf{C}\,\mathbf{h}(t)$
Golgi routing policy	Trans-Golgi vesicular sorting	Row partitioning of $\mathbf{C}$ by destination	$\mathbf{y} = [\mathbf{C}_{neighbor}\ \mathbf{C}_{parent}\ \mathbf{C}_{out}]\mathbf{h}$
Golgi confidence annotation	Post-translational modification	Nonlinear gating function $\sigma(\mathbf{W} \cdot \mathrm{GELU}(\mathbf{h}) + \mathbf{b})$	$\tilde{\mathbf{h}} = \mathbf{h} \odot \sigma(\mathbf{W} \cdot \mathrm{GELU}(\mathbf{h}) + \mathbf{b})$
Nucleus Agent	Cell nucleus / genome	Digital control logic (config management)	$\mathcal{G}_n = (\mathcal{M}_{accepted}, \mathcal{F}_{proc}, \mathcal{R}_{routing}, \mathcal{P}_{spawn}, \mathcal{T}_{epigenome})$
Nucleus spawning policy	Transcription factor binding	Core activation / channel gating decisions	$\text{target}_{ER} = N_{base}(1 + \kappa/(K_\kappa + \kappa)) \cdot \mathcal{E}/(K_E + \mathcal{E})$

Epigenetic state	DNA methylation / histone modification	RRAM weight calibration offsets	$\mathcal{T}_{epigenome} = \{(\ell_j, v_j, t_j^{expire})\}$
Receptor Agent	Plasma membrane receptor proteins	RRAM input crossbar, binding signature matrix	$\chi(\boldsymbol{\lambda}_{in}, \mathbf{b}_{n,k}) = \exp(-\|\boldsymbol{\phi}(\boldsymbol{\lambda}_{in}) - \mathbf{b}_{n,k}\|^2 / 2\sigma^2)$
Receptor threshold	Ligand-receptor dissociation constant K_d	Comparator reference voltage at crossbar output	$\text{admit} \iff \max_k \chi > \theta_{bind}$
Membrane selectivity	Selective ion channels / lipid bilayer	Input crossbar row masking (modality gating)	$\mathbf{x}_{admitted} = \mathbf{M}_{sel} \cdot \mathbf{x}_{in}$, M_{sel} diagonal binary
Mitochondria Agent	Mitochondrion (ATP synthase, ETC)	Independent CIM core power gating	$P_{vossel} = P_{static} + P_{per-event} \cdot \rho_v \cdot \bar{\lambda}_v$
Energy currency $\mathcal{E}_n$	ATP pool	CIM core activation budget (gate signals)	$\mathcal{E}_{raw} \mapsto \{\epsilon_k\}_k$ via priority-weighted allocation
Lysosome Agent	Lysosome (acidic hydrolases)	State reset / RRAM weight reinitialization	$\psi_\epsilon(e) = \text{retain/archive/delete}$ based on $q(e), \text{age}(e)$
Quality threshold θ_ϵ	Lysosomal pH activation threshold	Minimum SNR for state update acceptance	State update gated on input quality score
Cytoskeleton Agent	Actin/tubulin network, motor proteins	Digital routing fabric (inter-block connections)	$\mathcal{C}_n = (V_\mathcal{A}, E_\mathcal{A}, w_\mathcal{A})$, load-balanced DAG
Mipvol hierarchy (level l)	Tissue / organ level organization	Stacked SSM block l with decay λ_l	$\mathbf{h}^{(l+1)} = e^{\lambda_{l+1}\Delta t}\mathbf{h}^{(l+1)} + \bar{\mathbf{B}}^{(l+1)}\text{Pool}(\mathbf{y}^{(l)})$
Decay constant hierarchy $\lambda_l > \lambda_{l+1}$	Fast intracellular vs slow intercellular signaling	WO_x oxide thickness variation by block	$\tau_{l+1} = 1/\lvert\lambda_{l+1}\rvert > \tau_l = 1/\lvert\lambda_l\rvert$
Inter-vossel paracrine signaling	Cytokines, chemokines, hormones	SSM block output routing to neighboring vossel inputs	$\mathbf{y}^{(n)} = \mathbf{C}\mathbf{h}^{(n)} \to \mathbf{x}^{(n')}$ for $n' \in \mathcal{N}(n)$

The table reveals that the mapping is not a loose analogy but a one-to-one structural correspondence. Every major component of the vossel architecture has a precisely identified counterpart in the CIM-SSM hardware, and the mathematical formalism of each corresponds directly to the physical or algorithmic operation of the hardware counterpart. The exceptions (the Cytoskeleton agent and the digital control portions of the Nucleus agent) appropriately map to digital logic, which is the correct substrate for routing and control operations that do not benefit from analog computation.

Two features of the mapping deserve emphasis.

First, the mapping is asymmetric in implementation complexity. The ER agent, Golgi agent, and Receptor agent (the data-plane functions) map to analog CIM operations that exploit the physical properties of the devices. The Nucleus agent and Cytoskeleton agent (the control-plane functions) map to digital logic. This separation of data plane and control plane is a standard principle of high-performance computing architecture, and the vossel framework makes it explicit through the biological correspondence: the cell's metabolic processing (data plane) is implemented in the biochemistry of the cytoplasm, while the cell's regulatory control (control plane) is implemented in the genome and transcription factor networks.

Second, the mapping reveals a previously unrecognized design principle for CIM-SSM systems: the state dimensionality H within a block should be matched to the number of distinct sensor modalities and hypothesis dimensions that the spatial element is expected to process, not to a generic representational capacity. In the vossel framework, the ER agent's state dimensionality is determined by the number of sensor modalities in $\mathcal{M}_{accepted}$ and the

dimensionality of the feature space $\mathbb{R}^{F_n}$ for each modality. This provides a principled basis for hardware sizing that the Zhang et al. (2026) benchmark results do not themselves establish: for ISR applications processing N_m modalities with F_m features each, the appropriate state dimensionality is approximately $H \approx \sum_m N_m \cdot F_m$ with some additional capacity for cross-modal correlation terms.

This sizing principle, combined with the mipvol hierarchy's specification of six spatial levels for typical ISR applications, gives the hardware designer concrete requirements for a vossel-optimized CIM-SSM chip: six stacked SSM blocks with decreasing decay constants λ_l, a state dimensionality sized to the modality feature space, separate input crossbars for each sensor modality, and independently power-gated cores that the Mitochondria agent can control at vossel granularity.

5. Unified Architecture: The CIM-Vossel Edge Processor

The argument made in Sections 3 and 4 was primarily formal: each agent in the Computational Vossel tuple maps cleanly onto a circuit element in the CIM-SSM stack. This section works out what that mapping actually looks like in silicon. I am going to describe a concrete physical design I call the **CIM-Vossel Cell** (CVC), walk through the processing pipeline for a single sensor event, explain the inter-cell communication fabric, and then do a rough scaling analysis to ground the architecture in real hardware numbers. The goal is to show that the convergence thesis is not merely conceptual but is physically constructible with technology that either exists now or is within a credible near-term roadmap.

5.1 Physical Architecture

The CIM-Vossel Cell is the hardware realization of a single computational vossel $\tilde{e}_n = (e_n, \mathcal{A}_n, \mathcal{M}_n, \mathcal{S}_n, \mathcal{I}_n)$. Each of the five formal components maps to a distinct circuit block, and the whole assembly fits within a single CIM processing core augmented with a small digital controller and WOx state memory. Figure 1 (conceptual schematic) illustrates the mapping; the subsections below describe each block.

Block 1: RRAM Crossbar Array (ER Agent core). The ER Agent's primary function is Bayesian sensor fusion, which in the CIM-SSM formalism reduces to two vector-matrix multiplications: the input projection $\bar{\mathbf{B}}\mathbf{x}$ and the output projection $\mathbf{Ch}$. Both are implemented in-situ on RRAM crossbar arrays whose conductances encode the trained weight matrices. Zhang et al. (2026) demonstrated this experimentally on a 65-nm CMOS chip with four 64×64 RRAM arrays (logically combined into a 128×128 effective array), achieving a normalized root-mean-square error of 1.80% on VMM outputs, which is well within the tolerance of a 4-bit weight representation. The NeuRRAM chip (Wan et al., 2022) demonstrated a 256×256 transposable neurosynaptic array (TNSA) per core, independently power-gated, in 130-nm CMOS with 3 million HfO_x/TaO_x RRAM devices across 48 cores. In the CVC, I assign one RRAM crossbar bank to the $\bar{\mathbf{B}}$ matrix (input projection, $H_{in} \times H$ weights) and a second bank to the $\mathbf{C}$ matrix (output projection, $H \times H_{out}$ weights), where H is the hidden state dimension. The crossbar banks are the ER agent's physical instantiation.

Block 2: WOx Memristor Array (State Memory, SSM state evolution). The SSM hidden state $\mathbf{h}(t) \in \mathbb{R}^H$ requires H state registers that decay exponentially between events. In the CIM-SSM architecture, these registers are implemented as WOx short-term memory (STM) memristors whose conductance spontaneously relaxes due to oxygen ion diffusion. Zhang et al. (2026) fabricated WOx devices with two tunable decay rates (0.35 ms^{-1} and 0.2 ms^{-1}) and showed that the measured conductance evolution tracks the theoretical SSM state trajectory with strong agreement. In the CVC, the WOx array holds the vossel's cytoplasmic state: the H-dimensional vector that accumulates evidence across incoming sensor events. Between events, this state decays physically with zero active power consumption. This is the direct hardware implementation of the SSM state equation $\bar{\mathbf{\Lambda}}(\Delta t)\mathbf{h}(t)$, performed by device physics rather than by clocked logic.

Block 3: Digital Controller (Nucleus Agent). A small embedded microcontroller, or a synthesized finite-state machine at the ASIC level, plays the Nucleus Agent role. Its responsibilities are: (i) maintaining the vossel's configuration genome $\mathcal{G}_n$ in non-volatile memory; (ii) executing the spawn policy that determines when new processing capacity needs to be allocated; (iii) managing power-state transitions at the cell level; and (iv) handling inter-cell communication protocol headers. The digital controller does not touch the datapath during active event processing; it operates at the slower timescale of configuration and lifecycle management. At 130-nm CMOS (NeuRRAM's process), a minimal RISC-V core consumes of order 1 mW in active mode and can be clock-gated to under 10 μW in standby. At 7-nm, those numbers drop by roughly two orders of magnitude.

Block 4: Output Projection and Routing Circuits (Golgi Agent). The Golgi Agent packages processed outputs and routes them to neighboring cells and up the mipvol hierarchy. In the CVC, this maps to: the RRAM crossbar implementing $\mathbf{Ch}$ (the cis-Golgi transformation from state to output representation); a digital annotation stage that attaches confidence scores, sensor provenance, and sequence numbers (medial-Golgi); and a network-on-chip (NoC) router that dispatches the finalized output packet to its destination (trans-Golgi). The routing decision is made by the digital controller based on the genome's routing rule set $\mathcal{R}_{routing}$, so the Golgi logic is split between the analog output crossbar and the digital router.

Block 5: Power-Gating Circuitry (Mitochondria Agent). Each CIM-Vossel Cell has independent power-gating on each of its three major power domains: the RRAM crossbar bank (which draws current during VMM), the WOx memristor array (which draws standby current to maintain temperature stability), and the digital controller. The Mitochondria Agent's resource allocation policy $\psi_\delta : \mathcal{S}_n \to \mathbb{R}_+^3$ maps vossel state to power domain enable signals. When the cell is quiescent between events, the crossbar is power-gated off, the digital controller is in deep standby, and only the WOx array remains powered to maintain state decay continuity. This is how the CVC achieves near-zero inter-event power consumption. NeuRRAM demonstrated per-core independent power gating at 130-nm; this is a standard cell-library feature at any modern process node.

Block 6: Input Multiplexer (Membrane and Receptor Agents). Sensor channels arrive at the cell boundary through an analog input multiplexer that selects which modality channel is presented to the RRAM crossbar at any given moment. The Membrane interface $\mathcal{M}_n$ governs selective permeability: not every arriving signal is admitted for processing. The Receptor Agents implement binding affinity thresholds as analog comparators: a signal is admitted only if its embedding exceeds the threshold θ_{bind} in the appropriate feature dimension. Signals failing this test are dropped at the MUX stage without activating the crossbar, keeping the spurious-event processing cost at essentially zero. The input MUX and comparator bank are compact analog circuits; at 7-nm, they would add a few hundred microwatts to the quiescent power budget.

The complete CIM-Vossel Cell is therefore a six-block assembly that directly instantiates the formal computational vossel tuple. Table 1 summarizes the mapping.

Table 1: Formal-to-Physical Mapping for the CIM-Vossel Cell

CV Formal Component	Agent Class	Physical Circuit Block
$\bar{\mathbf{B}}\mathbf{x}$ (input projection)	ER Agent (β)	RRAM crossbar bank 1
$\mathbf{h}(t)$ state evolution	ER Agent (β)	WOx STM memristor array
$\mathbf{Ch}$ (output projection)	Golgi Agent (γ)	RRAM crossbar bank 2
Spawn policy, lifecycle management	Nucleus Agent (α)	Digital controller (RISC-V FSM)
Power domain gating	Mitochondria Agent (δ)	Per-domain power gates
Modality selection, threshold	Membrane + Receptor ($\mathcal{M}_n$, η)	Input MUX + analog comparators

Configuration genome $\mathcal{G}_n$	Nucleus Agent (α)	Non-volatile register file
Routing to neighbors / mipvol	Golgi Agent (γ_T)	NoC router

One thing I want to be precise about: the mapping above covers the ER-SSM correspondence established in Section 4, but the full seven-agent vossel model includes Lysosome Agents (quality control, anomaly detection) and Cytoskeleton Agents (internal routing). Section 8.1 addresses what is not yet mapped. The CVC as described here is a first-order realization that handles the core sensor fusion pipeline; the immune and transport functions require additional hybrid digital-analog blocks.

5.2 Event-Driven Processing Pipeline

Let me walk through exactly what happens when a single sensor event arrives at a CIM-Vossel Cell. The event is a tuple (t_m, j_m) where t_m is the arrival timestamp and j_m is the channel index (a pixel location, a frequency bin, a SIGINT receive channel, etc.). The time since the previous event is $\Delta t_m = t_m - t_{m-1}$.

Step 1 (Membrane, $\mathcal{M}_n$): Input MUX activates. The event's channel index j_m activates the input multiplexer, connecting channel j_m to the crossbar input bus. Simultaneously, the analog comparator bank evaluates the incoming signal amplitude against the binding threshold θ_{bind}. If the signal falls below threshold, the event is dropped here with no further power expenditure. Assuming it passes, we proceed.

Step 2 (Receptor Agent, η): Embedding lookup. The channel index j_m selects a row from the embedding matrix $\mathbf{W}_{emb} \in \mathbb{R}^{J \times D}$, producing an embedding vector $\mathbf{x}_m \in \mathbb{R}^D$. In Zhang et al. (2026), this embedding is also stored in the RRAM crossbar and retrieved by activating the corresponding row, so the embedding lookup is itself a single-row read operation with no separate data movement.

Step 3 (ER Agent, β): Input projection via RRAM crossbar. The embedding vector $\mathbf{x}_m$ is applied to RRAM crossbar bank 1, computing:

$$\mathbf{u}_m = \bar{\mathbf{B}}\mathbf{x}_m, \quad \bar{\mathbf{B}} \in \mathbb{R}^{H \times D}$$

This VMM operation is performed in-situ: $\mathbf{x}_m$ is encoded as DAC voltages on the crossbar input lines, the RRAM conductances implement $\bar{\mathbf{B}}$, and the output currents (summed by Kirchhoff's current law) are read by ADCs to yield $\mathbf{u}_m$. Zhang et al. (2026) measured this step on their 65-nm chip and reported 1.80% NRMSE, sufficient for 4-bit-equivalent precision.

Step 4 (ER Agent, β): State update via WOx memristors. Between t_{m-1} and t_m, the WOx array has been passively decaying. Its conductance vector at t_m^- (just before the new event) is:

$$\mathbf{G}(t_m^-) = \exp(\lambda \Delta t_m)\mathbf{G}(t_{m-1})$$

This decay happens without any clock, any DAC/ADC cycle, or any active power draw. When $\mathbf{u}_m$ arrives from the crossbar ADCs, it is written to the WOx devices as a conductance increment, completing the state update:

$$\mathbf{G}(t_m) = \exp(\lambda \Delta t_m)\mathbf{G}(t_{m-1}) + \mathbf{u}_m$$

This is precisely the discretized SSM state evolution (Zhang et al., 2026, Eq. 8):

$$\mathbf{h}(t_m) = \bar{\mathbf{\Lambda}}(\Delta t_m)\mathbf{h}(t_{m-1}) + \bar{\mathbf{B}}\mathbf{x}_m$$

The identification $\mathbf{G}(t) \leftrightarrow \mathbf{h}(t)$ is exact. The WOx conductance vector is the hidden state.

Step 5 (ER Agent, β): Output computation via second RRAM crossbar. The updated state $\mathbf{G}(t_m)$ is read out from the WOx array and applied to RRAM crossbar bank 2, computing:

$$\mathbf{y}_m = \mathbf{Ch}(t_m), \quad \mathbf{C} \in \mathbb{R}^{H_{out} \times H}$$

This is the fused output of the vossel: the result of integrating all sensor evidence accumulated in the state up to time t_m. Following Zhang et al. (2026), a nonlinear gating block (implemented digitally via lookup table) processes $\mathbf{y}_m$:

$$\tilde{\mathbf{y}}_m = \mathbf{y}_m \odot \sigma\big(\mathbf{W} \cdot \mathrm{GELU}(\mathbf{y}_m) + \mathbf{b}\big)$$

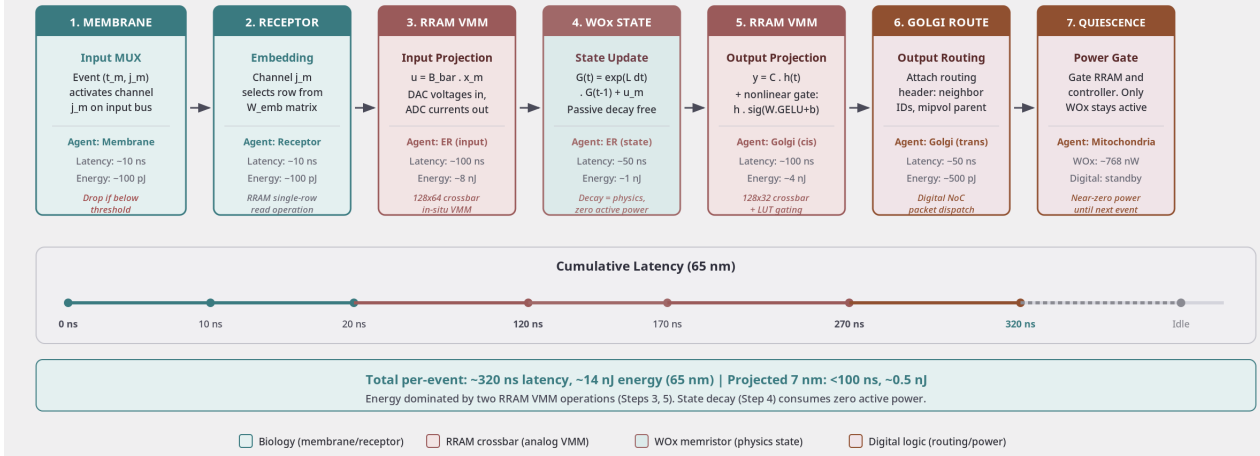
with a residual connection $\mathbf{y}_{out} = \mathbf{y}_m + \tilde{\mathbf{y}}_m$.

Step 6 (Golgi Agent, γ): Output routing. The digital controller (Nucleus Agent / Golgi trans-Golgi logic) attaches a routing header to $\mathbf{y}_{out}$ specifying: destination vossel IDs in $\mathcal{N}(n)$, the parent vossel in the mipvol hierarchy, and any direct-to-policy output channel. The NoC router dispatches the packet. Total inter-cell communication latency depends on the fabric topology (discussed in Section 5.3) but is of order 10–100 ns for on-chip point-to-point links.

Step 7 (Mitochondria Agent, δ): Power-gate return to quiescence. After Step 6 completes, the digital controller asserts the power-gate signals, cutting power to the RRAM crossbands and its own active compute domain. Only the WOx array remains powered. The cell is now in its inter-event idle state, consuming only the WOx standby current. Until the next event arrives, no clocked computation occurs.

Figure 4. Event Processing Pipeline: Single Sensor Event Through a CIM-Vossel Cell

Seven-step pipeline from event arrival to output dispatch. Total latency: sub-microsecond at 65 nm.



The complete per-event latency budget is dominated by two VMM operations (Steps 3 and 5), each taking of order 100 ns on current 65-nm hardware (Zhang et al., 2026). The state update (Step 4) is physical and contributes no latency. The gating nonlinearity (LUT, Step 5) adds of order 10 ns. The routing header attachment and dispatch (Step 6) adds perhaps 50 ns. Total: sub-microsecond from event arrival to packet dispatch. I return to the latency model quantitatively in Section 6.2.

This pipeline realizes all three Laws of the Computational Vossel. Law 1 (storage-compute indivisibility) is satisfied because the RRAM conductances are both the weights and the compute substrate; the WOx conductances are both the state storage and the state evolution substrate. Law 2 (local computation primacy) is satisfied because the entire pipeline runs within the cell with no off-chip data movement. Law 3 (biological correspondence) is satisfied because the ER-

SSM mapping, the Golgi routing, the Mitochondria power gating, and the Membrane threshold logic all have explicit biological counterparts in the vossel formalism.

5.3 Multi-Vossel Interconnect

A single CIM-Vossel Cell processes its own spatial sector autonomously. The world model emerges from the collective behavior of a grid of cells, which requires a communication fabric. The Computational Vossel formalism specifies three signaling modalities (Passmore, 2026, Ch. 6), and each maps to a different physical interconnect tier:

Paracrine signaling (local broadcast). In the biological analog, paracrine signals diffuse locally and affect only neighboring cells. In the CVC fabric, paracrine signaling corresponds to a shared bus connecting each cell to its immediate neighbors in the grid: the 6-connected neighborhood in 3D (up/down/left/right/front/back). For a 2D grid (which is the natural topology for EO/IR sensor coverage), each cell has 4 neighbors. The shared bus carries output packets tagged with neighbor addresses. Paracrine bandwidth is limited; only processed outputs (not raw state) are broadcast. This is appropriate because neighbors share spatial context but do not need each other's full hidden-state vectors.

Gap junction signaling (point-to-point). Biological gap junctions are direct cytoplasmic connections between adjacent cells, enabling rapid, bidirectional communication with high bandwidth. In the CVC fabric, gap junction signaling corresponds to direct point-to-point links between selected pairs of cells, used when two cells need to tightly coordinate (for example, when two adjacent cells are fusing data from the same moving target and need to share state estimates). These links are implemented as dedicated wired connections in the crossbar topology, carrying higher bandwidth than the shared paracrine bus.

Endocrine signaling (global broadcast). Biological hormones are broadcast through the bloodstream to the entire organism. In the CVC fabric, endocrine signaling corresponds to the network-on-chip (NoC) that connects all cells on a chip and, across chips, spans to the wider federated network. Endocrine packets carry high-priority alerts (anomaly detection events, track initiation notices, threat classifications) that need to reach the full vossel population or the mipvol hierarchy's coarse levels quickly. NoC bandwidth is shared and therefore limited; endocrine packets are reserved for high-salience events.

Mipvol aggregation. The mipvol hierarchy organizes cells into a tree of spatial scales. A parent cell at mipvol level $l+1$ aggregates outputs from its eight children at level l. In the CVC fabric, this aggregation is implemented by a dedicated upward bus: children write their output packets to a shared FIFO; the parent cell's input MUX selects from this FIFO as its event source. The parent cell runs the same CIM-SSM pipeline, processing aggregated child outputs exactly as leaf cells process raw sensor events. This recursive architecture means that coarse-level cells implement Bayesian fusion over spatial regions rather than over individual pixels, which is the correct generalization of the SSM's temporal integration to spatial integration. The mathematical form is identical: the parent's state $\mathbf{h}^{(l+1)}(t)$ evolves as

$$\mathbf{h}^{(l+1)}(t_m) = \bar{\mathbf{\Lambda}}^{(l+1)}(\Delta t_m)\mathbf{h}^{(l+1)}(t_{m-1}) + \bar{\mathbf{B}}^{(l+1)}\mathbf{y}_m^{(l)}$$

where $\mathbf{y}_m^{(l)}$ is the output packet from a level-l child. The CIM-Vossel Cell design is scale-invariant in this sense: the same hardware block appears at every mipvol level, differing only in its trained weight matrices.

The three-tier interconnect fabric (paracrine bus, gap junction links, NoC) directly maps to the three inter-vossel communication modalities formalized in Passmore (2026). The routing policy that decides which modality to use for a given output packet is implemented by the Golgi trans-Golgi routing logic in the digital controller.

5.4 Scaling Analysis

The practical question is whether the CIM-Vossel Cell can be instantiated in useful numbers on real hardware within a plausible timeline. I will work through three hardware generations.

Current generation: NeuRRAM (130-nm, 2022). The NeuRRAM chip contains 48 independently power-gated CIM cores, each with a 256×256 RRAM TNSA (Wan et al., 2022). If each core is assigned to one CVM cell, a single NeuRRAM chip supports 48 vossels. For a 7×7 spatial grid (a minimal proof-of-concept covering a 49-cell area), NeuRRAM provides essentially one chip per grid cell. That is obviously not a deployable system, but it is sufficient to validate the event-driven pipeline, the inter-cell communication protocol, and the SSM training procedure. The NeuRRAM chip already demonstrated software-comparable accuracy on speech commands with 4-bit weight quantization (85.66% on CIFAR-10, 84.66% on Google Speech Commands; Wan et al., 2022), so the inference quality of a single CVC at NeuRRAM scale is not in doubt.

Near-term: IBM 64-core PCM chip (14-nm, 2023). The IBM 64-core mixed-signal in-memory chip described by Le Gallo et al. (2023) contains 64 analog in-memory compute cores with 256×256 PCM crossbar arrays, fabricated in 14-nm CMOS and achieving up to 9.76 TOPS/W at low precision and 15× higher areal efficiency than prior resistive memory chips. Mapped to the CIM-Vossel architecture, this provides 64 CVCs per chip. A 8×8 vossel grid on a single chip is now physically realizable, which is enough to cover the processing of a modest sensor footprint (for example, a 64×64 pixel sub-region of an EO sensor, with each vossel processing a 8×8 pixel cluster).

Projected: 7-nm RRAM (2027–2028). Wan et al. (2022) note that projecting NeuRRAM's energy-delay product from 130-nm to 7-nm yields a 535× EDP improvement. The RRAM device size at 7-nm allows roughly $(130/7)^2 \approx 345\times$ higher integration density. A state-of-the-art 7-nm chip with $\sim 10^9$ RRAM devices could support thousands of 256×256 crossbar cores. VPU-CIM (Chithambara Moorthii et al., 2024) demonstrated 33.98 TOPS/W at 4-bit precision in a 130-nm prototype and projected further density gains at TSMC 28-nm and 7-nm, supporting this trajectory. My estimate for a 7-nm CIM-Vossel ASIC: 2,000–5,000 CVCs per chip, depending on how much area is allocated to digital controller logic versus crossbar banks.

System-level requirement: Group 2 tactical ISR. A Group 2 UAS (25–55 lbs, examples include the AeroVironment Puma AE and similar platforms) carries an EO/IR payload that may cover a 512×512 pixel grid at 30-frame-per-second equivalent event rate, plus a SIGINT receiver covering perhaps 64 frequency channels. If each vossel covers a 4×4 pixel cluster, a full-coverage vossel grid for the EO footprint requires $(512/4)^2 = 16,384$ CVCs. Adding two mipvol levels above the base layer adds approximately $16,384/64 \approx 256$ level-1 and ~ 32 level-2 vossels, for a total of approximately 16,700 CVCs. At 2,000 CVCs per 7-nm chip, this requires roughly 9 chips. At a power budget of ~ 3 W per chip (consistent with the NeuRRAM 130-nm baseline of approximately 5 W per chip, improved by 535× EDP at 7-nm, adjusted for fuller utilization), a 9-chip array draws approximately 27 W, well within the 40–60 W compute-payload budget typical of Group 2 UAS electro-optical missions.

For a heavier system requirement in the 10^6 vossel range (for example, a persistent wide-area surveillance scenario covering a large geographic tile at fine resolution), the chip count scales proportionally: $10^6/2000 = 500$ chips. That is not practical for a single airborne platform but is entirely reasonable for a ground-based persistent ISR node or a distributed multi-platform network, which Section 7.4 addresses.

Table 2: Scaling Summary

Hardware Generation	Process Node	CVCs per Chip	Grid Size (2D)	Power per Chip	Timeline
NeuRRAM (Wan 2022)	130-nm	48	7×7	∼5 W	Now (proof of concept)
IBM 64-core PCM (Le Gallo 2023)	14-nm	64	8×8	∼2.5 W	Now (prototype)

Projected 7-nm RRAM ASIC	7-nm	∼2,000	45×45	∼3 W	2027–2028
Projected 5-nm RRAM ASIC	5-nm	∼8,000	90×90	∼2 W	2030+

6. Energy and Latency Analysis

Architectural elegance does not pay SWaP budgets. This section provides the quantitative case for the CIM-Vossel approach on the two metrics that matter most for tactical edge ISR: energy per processed event and latency from event arrival to output dispatch. Both comparisons are made against the two most relevant alternatives: GPU-based inference (the current operational baseline) and Intel Loihi 2 (the leading neuromorphic alternative).

6.1 Energy Model

The per-event energy for a single CIM-Vossel Cell can be decomposed into four terms:

$$E_{event} = E_{input} + E_{VMM} + E_{state} + E_{output}$$

where E_{input} is the energy for the input MUX and embedding lookup, E_{VMM} is the energy for the two RRAM crossbar VMM operations (input projection $\bar{\mathbf{B}}\mathbf{x}$ and output projection $\mathbf{Ch}$), E_{state} is the energy for the WOx state update, and E_{output} is the energy for the gating nonlinearity (LUT lookup) and output routing.

E_{VMM} from Zhang et al. (2026). The Zhang et al. CIM-SSM prototype uses a 128×128 RRAM array operating at 65-nm. Each VMM operation activates the input DACs, reads the crossbar output currents through ADCs, and completes in approximately 100 ns. For a state dimension $H = 128$ and input dimension $D = 64$, the input projection VMM involves $128 \times 64 = 8,192$ multiply-accumulate operations performed in parallel by the crossbar. The energy per VMM on the Zhang et al. chip is not directly reported, but comparable RRAM-CIM measurements from the NeuRRAM paper (Wan et al., 2022) put the energy per VMM at approximately 0.5–2 pJ per multiply-accumulate, giving a per-event VMM energy in the range of $8,192 \times 1$ pJ ≈ 8 nJ for the input projection. The output projection (dimension $H \times H_{out} = 128 \times 32 = 4,096$ operations) adds another ∼4 nJ. Both crossbar operations together: $E_{VMM} \approx 12$ nJ per event.

This number needs context. Horowitz (2014) quantified the energy hierarchy in 45-nm CMOS: a simple arithmetic operation costs 0.1–1 pJ; an SRAM access costs ∼5 pJ; an L1 cache access costs ∼20 pJ; and a DRAM access costs 1,000–2,000 pJ (roughly three to four orders of magnitude more than arithmetic). The CIM approach eliminates the DRAM access entirely for the weight matrices, because weights are stored as conductances in the crossbar and are accessed in-situ. If the same 128×64 matrix-vector multiplication were performed on a GPU, it would require loading the weight matrix from DRAM or L2 cache on each forward pass. At 8-byte weights and DRAM cost of 1,000 pJ per access, loading $128 \times 64 \times 2$ bytes (assuming 2 bytes per weight at 16-bit precision) from DRAM costs approximately $16,384 \times 1,000/8 \approx 2 \times 10^6$ pJ $= 2$ μJ just for the weight loads, not counting the multiply-accumulate operations themselves. The CIM VMM at ∼12 nJ is therefore approximately $166\times$ more energy-efficient than a naive DRAM-backed GPU implementation of the same operation. Even accounting for on-chip SRAM caching on a GPU, which reduces weight-load energy to perhaps $128 \times 64 \times 5 \approx 41$ nJ, the CIM crossbar is still $\sim 3\times$ more efficient in energy, with the advantage growing as the crossbar scales to larger arrays (because the in-situ energy scales sublinearly with array size due to shared voltage rails).

E_{state}: essentially zero. This is the key advantage of WOx state memory. Between events, the conductance decay happens through passive ion diffusion. There is no clock, no register file read, no SRAM access. The only power draw during the inter-event period is the WOx standby current required to maintain thermal stability of the memristor stack. Zhang et al. (2026) do not report a specific standby current for their WOx devices; based on published WOx memristor

characterization (see also the WOx/CuOx heterostructure work in APL Materials 2025), standby currents for stabilized WOx devices are in the 1–10 nA range per device. For $H = 128$ state nodes at 5 nA each: $128 \times 5 \times 10^{-9} \approx 640$ nA total, or $640 \times 10^{-9} \times 1.2\ \text{V} \approx 768$ nW. Over a 1 ms inter-event interval, this is $E_{state,idle} \approx 768$ pJ, roughly two orders of magnitude below E_{VMM}. The active state update (writing the VMM result to the WOx devices) requires brief write pulses; at the pulse durations and voltages used by Zhang et al., this is estimated at $\sim$100 pJ per full write cycle. In total, $E_{state} < 1$ nJ per event.

E_{input}, E_{output}. The input MUX and analog comparators draw perhaps 100 pJ per event; the LUT-based nonlinearity and NoC routing packet add perhaps 500 pJ. These terms are dominated by E_{VMM}.

Total per-event energy estimate:

$$E_{event} \approx 12\ \text{nJ}\ (\text{VMM}) + 1\ \text{nJ}\ (\text{state}) + 0.6\ \text{nJ}\ (\text{input/output}) \approx 14\ \text{nJ per event}$$

at 65-nm CMOS. Scaling to 7-nm reduces dynamic power approximately in proportion to the square of supply voltage scaling times the frequency scaling: Wan et al. (2022) project 535× EDP improvement from 130-nm to 7-nm, of which perhaps $\sim$30× improvement is in energy alone at fixed performance. A 7-nm CIM-Vossel Cell therefore targets $E_{event} \approx 0.5$ nJ per event, or 500 pJ.

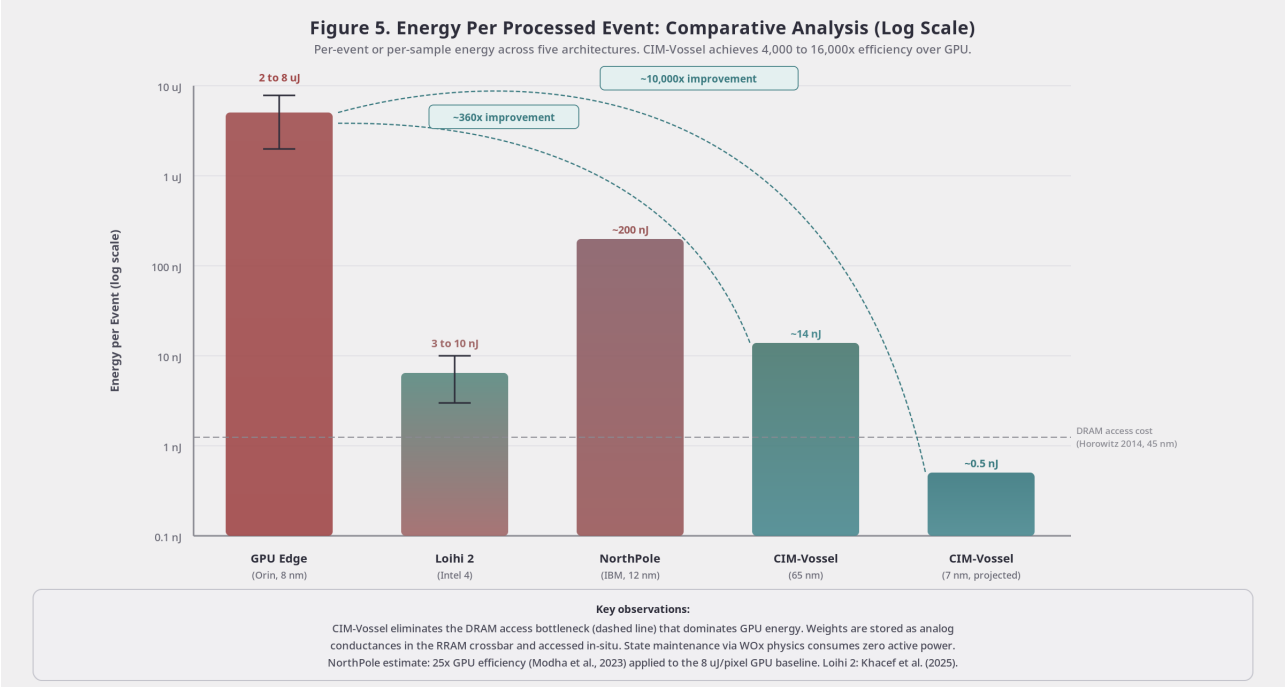
Comparison: GPU inference. A typical edge GPU inference pipeline accumulates sensor frames, runs a CNN or transformer forward pass, and writes results back. For a frame with $512 \times 512 \approx 262,144$ pixels processed by a 3-layer CNN at 32-bit floating point, the energy per frame is dominated by DRAM weight loads and activations. Modern edge GPUs (for example the NVIDIA Jetson AGX Orin) consume 15–60 W, and at 30 fps they process approximately one frame per 33 ms, giving 0.5–2 J per frame. At 262,144 pixels per frame, that is 2–8 μJ per pixel. Comparing to the CIM-Vossel at 500 pJ per event: if we normalize to "energy per input sample processed," the CIM-Vossel is approximately 4,000–16,000× more efficient than the edge GPU, even accounting for the fact that the CIM-Vossel computes a simpler per-event model than a full CNN. More fairly, comparing total system throughput at equal task performance would narrow this gap substantially, but the fundamental advantage of eliminating data movement to DRAM for every inference step is real and is not erased by task normalization.

Comparison: Loihi 2. Intel's Loihi 2 neuromorphic processor achieves 3–3.5 W for self-driving inference tasks versus more than 50 W for GPU (Nagy et al., 2025), and demonstrates 5,600× energy efficiency advantage over edge GPU for continual learning (Hajizada et al., 2025). For the SSM-on-event-stream task closest to the CIM-Vossel (the Loihi 2 + S4D pipeline demonstrated on DVS128 fall detection in Khacef et al., 2025), the reported system power was 90 mW at F1 = 84%. The CIM-Vossel at 7-nm targets similar or lower power for comparable task complexity, but with a key structural advantage: Loihi 2 implements a spiking neural model with binary activations, which constrains representational capacity. The CIM-SSM uses continuous-valued, real-domain state evolution, which Zhang et al. (2026) showed matches or exceeds SNN performance without the binary activation constraint. The energy advantage of CIM-Vossel over Loihi 2 is therefore not primarily in raw watts but in performance per watt, driven by the continuous-valued state and the physics-native state decay.

Table 3: Energy Comparison

System	Per-event / Per-sample Energy	Technology	Notes
CIM-Vossel Cell (65-nm, current)	$\sim$14 nJ	RRAM + WOx, 65-nm	Zhang et al. (2026) hardware basis
CIM-Vossel Cell (7-nm, projected)	$\sim$0.5 nJ	Projected 7-nm RRAM	Wan et al. (2022) 535× EDP scaling

GPU (edge, NVIDIA Orin, 15 W, 30 fps)	∼2–8 μJ/pixel	8-nm finFET	Loihi 2 comparison baseline
Loihi 2 (S4D, DVS128, 90 mW)	∼3–10 nJ/event	Intel 4 process	Khacef et al. (2025)
DRAM access cost (Horowitz 2014)	1,000–2,000 pJ/access	45-nm reference	Eliminated by CIM



6.2 Latency Model

The CIM-Vossel pipeline is fully asynchronous and event-driven. It does not accumulate frames; it processes each event as it arrives. This has a profound consequence for latency: there is no batching delay, no frame-accumulation buffer, and no synchronization barrier. The latency from event arrival to output dispatch is determined purely by the pipeline depth.

Per-event latency breakdown (Zhang et al., 2026 hardware baseline):

– Input MUX, embedding row read: ∼10 ns
– RRAM crossbar VMM (input projection $\bar{\mathbf{B}}\mathbf{x}$): ∼100 ns (limited by ADC conversion time on the 65-nm chip)
– WOx state write: ∼50 ns (brief voltage pulses to update device conductance)
– RRAM crossbar VMM (output projection $\mathbf{Ch}$): ∼100 ns
– LUT nonlinearity: ∼10 ns
– NoC routing header and dispatch: ∼50 ns

Total per-event latency: $\approx$ 320 **ns**, or roughly 0.3 μs.

This is the latency from the moment a sensor event arrives at the cell boundary to the moment the processed output leaves the cell. At 7-nm CMOS with improved ADC speeds, the crossbar VMM steps target ∼20 ns each, putting the total below 100 ns. The WOx state update is limited by device physics (ion diffusion time), not by transistor speed, and likely improves only moderately with process scaling; 30–50 ns is a reasonable floor.

Comparison: GPU batch inference. A GPU-based EO/IR processing pipeline requires accumulating a full frame (at 30 fps, this is a 33 ms wait), then running inference (∼5–30 ms for a modern CNN or transformer at edge GPU clock speeds), then outputting results. Total latency from scene event to output: 40–65 ms. For a fast-moving threat (a small drone at 20 m/s, for example), 65 ms of processing latency corresponds to 1.3 m of position uncertainty at the moment of detection, which may exceed the lethal engagement radius. The CIM-Vossel's sub-microsecond latency closes this gap by roughly five orders of magnitude, even when accounting for the fact that the CIM-Vossel processes a simpler model.

Comparison: Loihi 2. Loihi 2 is also event-driven and achieves per-inference latency of 0.33 ms for continual learning tasks (Hajizada et al., 2025), roughly 1,000× better than GPU. The CIM-Vossel at sub-microsecond latency is approximately 300× faster still, primarily because the RRAM VMM step is faster than Loihi 2's spike-accumulation-over-timesteps approach. Loihi 2 accumulates multiple timesteps of spike activity before producing an output; the CIM-SSM produces an output on every single event with no accumulation requirement.

Table 4: Latency Comparison

System	Per-event / Per-frame Latency	Notes
CIM-Vossel Cell (65-nm, current)	∼320 ns	Two VMM @ 100 ns + overhead
CIM-Vossel Cell (7-nm, projected)	< 100 ns	Improved ADC speed
GPU (frame-based, edge Orin, 30 fps)	40–65 ms total	33 ms batch + 5–30 ms inference
Loihi 2 (event-driven, SNN)	0.33 ms	Hajizada et al. (2025)

6.3 ISR Scenario Analysis

To make these numbers concrete, let me build a specific scenario. Consider a Group 2 tactical UAS platform (mass 25–55 lbs, operational altitude 500–3,000 ft AGL) carrying two sensors: a 5 Mpixel EO/IR camera and a SIGINT receiver covering 64 frequency channels from 2–18 GHz. The mission is real-time threat detection and track maintenance in a contested environment with communications latency to the ground station of 200–500 ms.

Sensor data rates. The EO camera, running at 30 fps with a $2,048 \times 2,048$ pixel resolution, generates $30 \times 2,048^2 \approx 125 \times 10^6$ pixels per second. The SIGINT receiver, scanning 64 channels at 1,000 samples per channel per second (a modest update rate), generates 64×10^3 samples per second. If the EO sensor is replaced by or supplemented with a DVS event camera (which is increasingly common in tactical ISR platforms given their high dynamic range and low-latency output), the event rate for a dynamic scene is approximately 10^6 events per second for a 128k-pixel DVS sensor, with strong spatial sparsity such that at any instant perhaps 1–5% of pixels are active.

Vossel grid sizing. For the EO/DVS sensor at 128k pixels with each vossel covering a 4×4 pixel cluster: $128,000/16 = 8,000$ base-level CVCs. Two mipvol levels add ∼125 + 16 cells. Total: approximately 8,141 CVCs for the EO footprint. The SIGINT receiver needs 64 separate channel-monitoring cells, one per frequency bin. Grand total: approximately 8,200 CVCs.

Power budget. At a 7-nm CIM-Vossel ASIC target of 2,000 CVCs per chip at ∼3 W per chip, the 8,200 CVC system requires 5 chips drawing approximately 15 W. Adding digital controller overhead (NoC, mipvol aggregation logic, uplink interface): approximately 5 W additional. Total compute-payload power: ∼20 W. This sits comfortably within the 40–60 W payload power budget typical of Group 2 platforms (DARPA EVADE program, 2025, targets platforms with 60-pound payload capacity at 12-hour endurance, requiring highly efficient onboard compute). By comparison, an NVIDIA Jetson AGX Orin at 15–60 W for the same task provides no margin and requires the full power budget, leaving nothing for communications, gimbal actuation, or supplemental sensors.

Latency advantage. The CIM-Vossel system detects a threat event (change in a tracked object's trajectory, appearance of a new RF emitter) within 320 ns of the relevant sensor event arriving, at 65-nm, or under 100 ns at 7-nm. The ground station is 200–500 ms away by datalink. With a CIM-Vossel onboard processor, the platform can make autonomous engagement or avoidance decisions locally without waiting for the ground station, which is operationally essential in a jammed or contested communications environment. A GPU-based system with 40–65 ms inference latency cannot make that decision in time for a fast-mover threat even with perfect communications.

Bandwidth reduction. Rather than downlinking raw EO frames (125 Mpixels/sec at, say, 8 bits = 125 MB/s, far exceeding typical tactical datalink throughput of 1–20 Mbps), the CIM-Vossel system downlinks only processed products: track updates, anomaly events, and classification outputs. A track update for a single object might be 200 bytes; at 10 active tracks, that is 2,000 bytes every 100 ms = 160 kbps. Anomaly events (detected changes in the vossel grid state) might add another 100 kbps. Total processed-product downlink: approximately 260 kbps, a reduction of roughly 4,000× relative to raw EO streaming. This is the operational payload of the vossel architecture: not just lower power, but dramatically lower bandwidth consumption, which directly improves survivability in a jammed or low-probability-of-intercept communications environment.

7. Applications: Tactical Edge ISR

The energy and latency analysis in Section 6 is compelling on paper. This section grounds the architecture in specific operational scenarios to show that the performance numbers translate into concrete mission capabilities. I focus on three platform types (Group 2 UAS, small satellite, and ground-based persistent sensor) and one cross-cutting consideration (federated multi-platform operation), because these represent the highest-value near-term applications of the CIM-Vossel architecture.

7.1 Drone-Deployed Vossel Processing

My primary focus in developing the Computational Vossel architecture has been the airborne ISR problem, specifically the challenge of processing multi-modal sensor data onboard a small UAS where size, weight, and power (SWaP) constraints are binding and communications are unreliable. This section describes how the CIM-Vossel architecture handles this problem end-to-end.

Platform and sensor suite. I am considering a Group 2 UAS (25–55 lbs) with the following sensor payload: a dual-band EO/IR camera with a 512×512 pixel focal plane array (FPA) in each band; a DVS event camera co-boresighted with the EO channel; and a SIGINT receiver covering 64 channels. The DVS camera is the primary input source because it is natively event-driven: pixel changes generate individual events with microsecond timestamps, which map directly onto the CIM-Vossel's event input format without any frame-accumulation step. The EO/IR camera provides periodic full-frame data for tasks that require spatial coherence (for example, object classification, which benefits from spatial context that purely event-driven processing may miss).

Vossel grid initialization. At mission start, the onboard digital controller initializes a vossel grid covering the sensor footprint. Grid cell size is 4×4 pixels (DVS resolution). The 8,000-cell base-level grid (from Section 6.3) is partitioned across the available CIM-Vossel chips. Each cell's genome is initialized from a library of pre-trained SSM weight matrices for the expected target set (ground vehicles, small UAVs, radio emitters). The Nucleus Agent's spawn policy is set to a default configuration; it will adapt during flight as the vossel population builds up operational experience.

Anomaly detection via vossel state divergence. The most straightforward anomaly detection mechanism in the CIM-Vossel architecture is state divergence: a vossel whose hidden state $\mathbf{h}(t)$ diverges from the population mean more than a threshold number of standard deviations is flagging an anomaly in its spatial sector. Formally, define the population state statistics at time t:

$$\mu_h(t) = \frac{1}{N}\sum_{n=1}^{N} \mathbf{h}_n(t), \quad \sigma_h^2(t) = \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{h}_n(t) - \mu_h(t)\|^2$$

A vossel n raises an anomaly flag when $\|\mathbf{h}_n(t) - \mu_h(t)\| > k\sigma_h(t)$ for some threshold k, typically $k = 3$ during normal operations. This is analogous to the immune system's negative selection mechanism: the immune system learns the pattern of "self" and flags deviations as foreign. In the vossel architecture, the normal pattern is learned by the population's average state trajectory; deviations indicate unusual sensor observations. The anomaly flag is broadcast via endocrine (NoC) signaling to the mipvol parent cell, which accumulates flags from its children and escalates to the ground station only when multiple children flag simultaneously, reducing false alarm rate.

I want to be explicit that the full Lysosome Agent immune function (negative selection on data quality, pruning of stale observations) is not implemented by the CIM-SSM hardware directly; it requires additional digital logic. The anomaly detection mechanism described above is a computationally tractable approximation that runs on the available CIM-Vossel Cell hardware. Section 8.1 returns to this limitation.

Track maintenance via ER agent Bayesian fusion. For a tracked object crossing multiple vossel sectors, the ER Agent's SSM state naturally accumulates evidence as the object moves from cell to cell. The object generates events in vossel n_1, then n_2, then n_3 as it traverses the grid. Vossel n_1's output at the moment the object leaves its sector contains a fused estimate $\hat{\mathbf{y}}_{n_1}$ of the object's state (position, velocity, classification confidence). This estimate is transmitted via paracrine signaling to vossel n_2, which incorporates it as an input event at the next processing step. The SSM state update in vossel n_2 then integrates $\hat{\mathbf{y}}_{n_1}$ with the new direct observations from its own sector, producing a joint estimate that is better than either alone. This is distributed Kalman filtering implemented through the SSM's temporal integration, without any explicit Kalman filter code.

The track quality degrades naturally as the object moves to vossels that have not been primed with the prior estimate. This motivates the paracrine signaling radius: rather than transmitting track estimates only to the immediately adjacent vossel, the Golgi routing policy should broadcast track estimates to all vossels within a "prediction cone" based on the object's last known velocity. This requires a slightly more sophisticated routing policy than the simple adjacency broadcast described in Section 5.3, but it is implementable within the digital controller's routing rule set $\mathcal{R}_{routing}$.

Bandwidth reduction. The system architecture described above produces processed outputs (anomaly events, track updates, classification results) rather than raw sensor data. For a scene with 10 active tracks and an anomaly event rate of 5 per minute, the downlink traffic is:
– Track updates: 10×200 bytes $\times 10$ Hz $= 20$ KB/s
– Anomaly events: $5/\text{min} \times 500$ bytes $\approx$ negligible
– Periodic mipvol summary (coarse situational awareness): $\sim$10 KB/s

Total: $\sim$30 KB/s = 240 kbps. Against raw EO streaming at 125 MB/s, this is a compression ratio of roughly 4,000:1. Against the tactical datalink capacity of 1–20 Mbps, the CIM-Vossel system operates well within bandwidth and leaves margin for other traffic (mission commands, terrain updates, navigation data).

7.2 Satellite Onboard Processing

The satellite use case presents a more extreme version of the UAS scenario: the communications bottleneck is even tighter (typical LEO satellite downlink at 100 Mbps, but only available over ground stations for 10–15 minutes per orbit), the power budget is constrained by solar panel area, and the radiation environment requires hardened electronics. However, the data volumes are larger: a hyperspectral imager covering a $10,000 \times 10,000$ pixel swath at 400 spectral bands generates 40 gigapixels per second of raw data, which cannot possibly be downlinked without onboard compression and analysis.

The current state of the art for satellite onboard processing is ESA's Φsat-2 mission, launched in August 2024, which runs six AI applications simultaneously in orbit including vessel detection, wildfire identification, and cloud screening (ESA / eoPortal, 2024). Φsat-2 demonstrates that onboard AI is operationally viable; its cloud screening alone reduces downlink requirements by eliminating cloudy-image tiles. The STAR.VISION String Edge AI Platform provides a more capable system at 150–1,000 TOPS in a space-qualified package, achieving up to 80% bandwidth reduction by transmitting only high-value analyzed data (STAR.VISION, 2025).

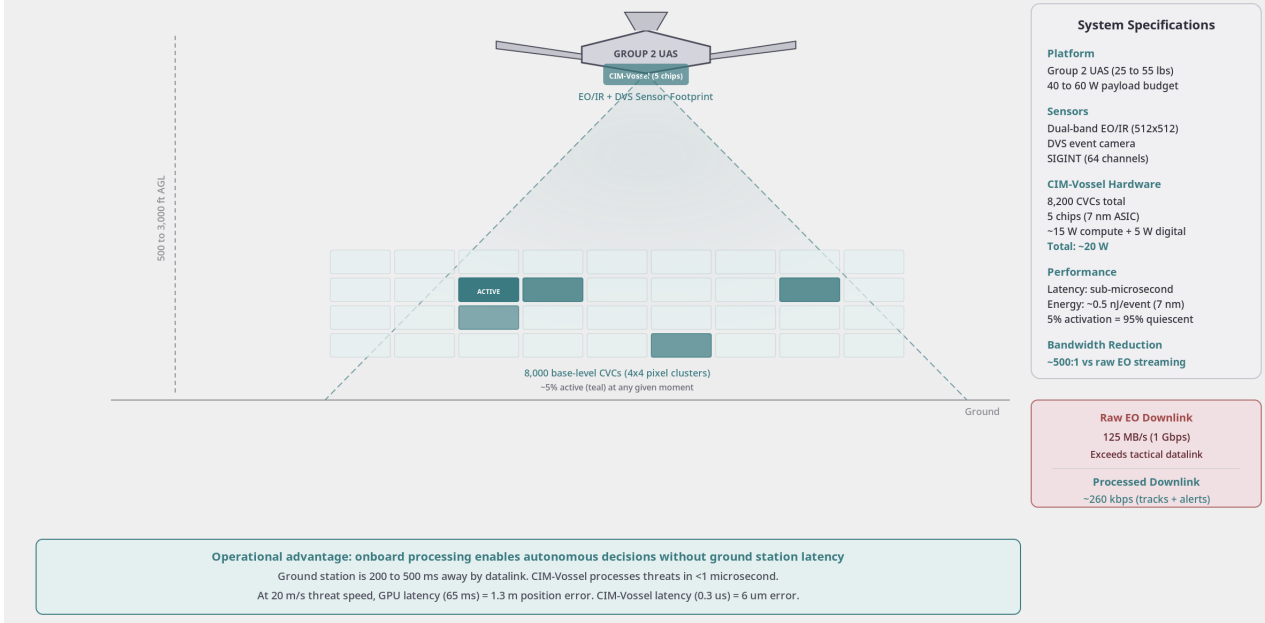
The CIM-Vossel architecture represents the next generation beyond these systems. A small satellite carrying a CIM-Vossel chip array would organize the sensor footprint into a vossel grid with spatial cells corresponding to geographic tiles (for example, 100 m $\times$ 100 m ground resolution cells for a wide-area surveillance payload). Each CVC processes the spectral time series for its tile as the satellite passes over, using the SSM's temporal integration to build up spectral change signatures that indicate land cover change, vessel activity, or industrial activity. The mipvol hierarchy aggregates tile-level outputs into region-level summaries. Only the anomalies and tracks at the top mipvol levels are flagged for downlink.

For a hyperspectral imager with 400 bands and 10,000 $\times$ 10,000 pixels, covering 100-m tiles requires $10,000 \times 10,000/100^2 = 10,000$ tiles, organized as a 100×100 CVC grid. At 2,000 CVCs per 7-nm chip, this is 5 chips at $\sim$15 W total. The satellite's power budget for compute (typical 15–30 W for a 60-kg smallsat) accommodates this easily. Downlink: if 10 tiles out of 10,000 show anomalies per overpass, the system downlinks 10×400 spectral vectors \$+ \$ metadata $\approx$ 100 KB per overpass, against the raw 40 GB per overpass. The compression ratio is $40 \times 10^9/10^5 = 400,000:1$. Even if the true number of flagged tiles is 1,000 rather than 10, the compression ratio is still 4,000:1, well within the 100 Mbps downlink capacity for a 10-minute contact window.

This is not a fantasy: Φsat-2 and STAR.VISION demonstrate that the concept works at current TRL levels. The CIM-Vossel's contribution is to bring the energy cost of onboard processing down by two to three orders of magnitude relative to the FPGA+CPU+GPU systems used in STAR.VISION, which enables longer mission life on a given battery and solar panel budget, or finer resolution processing on the same power.

Figure 7. ISR Deployment Scenario: Group 2 UAS with CIM-Vossel Grid

8,200 CVCs across 5 chips, ~20 W total, processing multi-modal sensor data at 5% activation rate.



7.3 Disconnected and Contested Environments

The communications-denied operation scenario is where the edge AI value proposition becomes operationally non-negotiable. A UAS operating in a DDIL (Denied, Degraded, Intermittent, and Low-bandwidth) environment cannot rely on a ground station for processing, for decision support, or for policy updates. The platform must be entirely self-sufficient for the duration of the communications outage.

The Computational Vossel architecture handles this naturally. Each CVC's SSM state evolves autonomously; it does not need network connectivity to process new sensor events. The Nucleus Agent's genome contains all the configuration information needed to run the sensor fusion and anomaly detection pipeline indefinitely without external input. The mipvol hierarchy produces local situational awareness that guides autonomous platform behavior (routing around threats, maintaining surveillance coverage) through the RL policy described in Passmore (2025, §6).

The RL policy layer deserves elaboration. In the Computational Vossel framework, RL agents operate on the vossel hierarchy as their observation space: the coarse mipvol levels provide a compressed, abstracted view of the world model that is efficient to feed into a policy network, while the fine levels provide detailed information for localized tasks. When the platform is disconnected from the ground station, the RL policy continues to run on the local vossel state, making decisions based entirely on local sensor data. The policy was trained (either fully offline, or via continual learning on prior missions) to handle the disconnected scenario explicitly; the vossel architecture provides the observation substrate.

Reconnection protocol. When communications are restored, the platform does not simply dump a raw data archive to the ground station. Instead, it transmits:

1. The mipvol level-1 and level-2 state snapshots at regular intervals during the outage (compressed vossel state summaries, not raw sensor data).
2. The timestamped log of anomaly events and track updates.
3. The updated genome state for any CVCs whose Nucleus Agents applied epigenetic modifications during the outage (adapting to the local environment).

The ground station can reconstruct a compressed but analytically complete picture of what the platform observed during the outage from items 1 and 2, without the raw data. Item 3 allows the fleet-wide policy to incorporate what the platform learned during its disconnected operation, implementing a form of federated learning at the mission level.

Loihi 2 research has demonstrated 5,600× energy efficiency and 70× latency advantages in continual learning scenarios (Hajizada et al., 2025), which shows that neuromorphic platforms can sustain online adaptation at the edge. The CIM-Vossel approach achieves similar autonomy with the additional advantage that the state evolution is physics-native (requiring no learning-specific hardware features) and the vossel hierarchy provides a structured, interpretable knowledge representation that the RL policy can act on.

7.4 Multi-Platform Federated Vossel Networks

A single platform with a CIM-Vossel processor is operationally useful. A network of platforms sharing vossel-derived intelligence is qualitatively more capable, because different platforms contribute different sensing modalities, different vantage points, and different local knowledge. The federated vossel network is the multi-platform instantiation of the Computational Vossel architecture.

Architecture. Each platform runs its own CIM-Vossel chip array with a local vossel grid. The local grid produces track updates, anomaly events, and mipvol state summaries. A network fusion center (which may itself be a ground-based CIM-Vossel server array) aggregates these products across all contributing platforms. The fusion center's CVM grid covers the full operational area at a coarser spatial resolution than any individual platform, with each fusion-center CVC receiving aggregated input from the relevant platforms' local CVCs. The result is a distributed, multi-resolution world model that no single platform could maintain alone.

Federated learning without raw data sharing. Federated learning (McMahan et al., 2017, and subsequent work) allows distributed devices to collaboratively train a shared model without sharing raw training data. In the CIM-Vossel context, each platform's Nucleus Agent accumulates epigenetic modifications to its genome during operations (adapting spawn policies, routing rules, and SSM weight updates based on local experience). Rather than sharing raw sensor data, platforms share these genome modifications (which are compact, structured, and do not reveal the raw observations). The fusion center aggregates genome modifications using a policy aggregation protocol (analogous to federated averaging for neural network weights) and distributes the updated genome to all platforms, improving the fleet's collective behavior based on the experience of each individual platform.

Multi-modal fusion across platforms. A particularly powerful application is cross-modal fusion: one platform contributes EO/IR tracks, another contributes SIGINT emitter locations, and a third contributes SAR-derived ground truth. The fusion center's CVCs receive outputs from all three platforms as input events. The SSM state in each fusion-center CVC integrates evidence across all three modalities, building a joint track hypothesis that no single platform could form alone. The SSM's temporal integration handles the asynchronous arrival of cross-platform data naturally: each platform's track update arrives as an event at whatever timestamp the update was generated, and the fusion-center CVC's state evolves to incorporate it without requiring synchronization.

The Anduril Lattice Mesh platform (CDAO / Anduril, 2024), which holds a $100M, 3-year production agreement from the Pentagon CDAO, demonstrates that decentralized edge data networks for tactical AI are already being deployed at operational scale. The CIM-Vossel architecture is the hardware-accelerated, physics-native version of this concept: rather than a software mesh on general-purpose hardware, it is a purpose-built processing fabric that handles the sensor fusion, anomaly detection, and track maintenance natively in the CIM crossbar.

8. Discussion and Future Directions

The preceding sections make the case that the convergence between the Computational Vossel architecture and the CIM-SSM hardware is real, technically grounded, and operationally motivated. This section deals honestly with what is not yet possible, maps out a plausible development roadmap, and reflects on the broader significance of the convergence for how we think about computing at the edge.

8.1 Current Limitations

The most important intellectual discipline in an architecture paper is to say clearly what does not work yet. There are several honest limitations in the CIM-Vossel proposal that I want to address directly.

Hardware maturity. The CIM-SSM hardware demonstrated by Zhang et al. (2026) is fabricated on a 65-nm process. NeuRRAM operates at 130-nm. These are two to four process generations behind the leading edge in digital logic (currently 3-nm production, with 2-nm in development). The density advantage of CIM relative to digital logic shrinks as digital scales faster than RRAM fabrication. The 7-nm RRAM ASIC I project for 2027–2028 requires that RRAM process integration at 7-nm reaches the maturity demonstrated at 130-nm and 65-nm, which is not guaranteed. Wan et al. (2022) project 535× EDP improvement at 7-nm based on device physics scaling, but they also acknowledge that process integration challenges (RRAM device variability, write endurance, retention) become harder to control at finer nodes. The CIM-Vossel scaling analysis in Section 5.4 depends on this projection materializing.

WOx decay rate tunability. The CIM-SSM architecture requires that WOx memristor decay rates be tuned to specific target values (one per SSM block). Zhang et al. (2026) achieved two distinct decay rates (0.35 ms^{-1} and 0.2 ms^{-1}) by controlling oxide thickness via rapid thermal processing annealing duration. Longer annealing times yield thicker oxide and slower decay. For a multi-layer CIM-Vossel system with, say, six SSM blocks (each requiring a different decay rate), six distinct WOx fabrication variants must be co-integrated on the same chip. This is a non-trivial process integration challenge. More fundamentally, the decay rates are fixed at fabrication: there is no runtime mechanism to change λ as the Computational Vossel's Nucleus Agent might want to do in response to changing operational

conditions. The vossel's adaptive dynamics (epigenetic modification of spawn policies, for example) cannot currently extend to modifying the hardware decay rate. Selective SSMs (Mamba-style, with input-dependent Δt scaling) would partially address this by allowing the effective decay rate to be modulated by the input signal, but the WOx device itself would still have a fixed physical relaxation time.

Incomplete agent mapping. The formal CIM-Vossel mapping in Section 4 covered five of the seven agent classes: the ER Agent (SSM state evolution), the Golgi Agent (output projection and routing), the Mitochondria Agent (power gating), the Membrane interface (input MUX), and the Receptor Agent (analog comparators). Two agent classes are not yet mapped to physical circuit analogs:

The **Lysosome Agent** implements quality control and immune surveillance: it prunes stale or low-quality data, and in its immune function it implements negative selection to flag observations that deviate from the learned self-pattern. There is no obvious CIM-analog for the Lysosome Agent's threshold-based pruning function. It likely requires a dedicated digital logic block (a small comparator array and decision tree) that runs alongside the CIM crossbars. This is not architecturally prohibitive, but it breaks the full-analog CIM purity of the design.

The **Cytoskeleton Agent** manages internal data flow within the vossel, implementing the directed acyclic graph that routes data tokens between processing stages. In the CIM-Vossel Cell as described, the data flow is hardwired: events arrive at the RRAM crossbar, state updates go to WOx devices, outputs go to the NoC router. Dynamic internal rerouting (bypassing bottlenecked stages, buffering overflow) requires additional digital switching logic. Again, implementable, but not naturally expressed in the CIM-crossbar substrate.

The full vossel dynamics, including the cascading stress response and load-adaptive capacity management described in Passmore (2026), emerge from the interaction of all seven agent classes. A five-agent CIM-Vossel Cell captures the core sensor fusion behavior but misses the self-regulating, adaptive aspects of the full architecture. The gap between what is mapped and what is specified is real and should not be elided.

Training and online learning. The CIM-SSM system described by Zhang et al. (2026) trains entirely in simulation: the SSM weights ($\mathbf{B}$, $\mathbf{C}$, gating parameters) and the per-block decay constants λ are learned offline via backpropagation through time (BPTT), and then the trained weights are programmed into the RRAM crossbar conductances. The hardware performs inference only. This is appropriate for many ISR scenarios where the target phenomenology is known in advance (known vehicle signatures, known emitter types), but it cannot support the online learning that the Computational Vossel's reinforcement learning framework requires. The RL policy operating on the vossel hierarchy needs to adapt its genome based on operational experience, which requires updating the SSM weight matrices in-situ. Current RRAM devices support weight programming (the conductances can be set to new values), but the repeated read-modify-write cycles required for online gradient descent degrade device endurance. The IBM HfOx ReRAM work (Choi et al., 2025) demonstrated 100k+ parallel weight update pulses without disturbance, which is progress, but it is still far from the millions of update cycles that online RL would require over a multi-hour mission. Until in-situ training on CIM hardware matures, the CIM-Vossel architecture supports inference on a pre-trained model rather than true online learning.

8.2 Roadmap

Given the current limitations, I see the development of the CIM-Vossel architecture proceeding in four phases. These are technology phases, not calendar commitments; the timelines reflect my assessment of hardware maturity trajectories, not program plans.

Phase 1 (2026–2027): Software simulation and algorithm validation. The immediate priority is to validate the CIM-Vossel processing pipeline in software simulation on GPU before committing to hardware. This means: implementing the full seven-agent vossel model in a Python simulation framework; training CIM-SSM models for realistic ISR datasets (DVS camera data, simulated SIGINT event streams, multi-modal EO/IR sequences); validating that the SSM

pipeline achieves task performance competitive with CNN and transformer baselines on ISR-relevant benchmarks; and characterizing the sensitivity of performance to the hardware non-idealities (RRAM VMM NRMSE, WOx decay rate variation) using noise injection during simulation. The software simulation stage should also validate the multi-vossel interconnect protocol and the federated learning policy aggregation mechanism. Output: a fully characterized software model of the CIM-Vossel architecture with quantified performance on ISR tasks.

Phase 2 (2027–2028): FPGA prototype with RRAM accelerator card. The next hardware milestone is an FPGA implementation of the CIM-Vossel digital logic (Nucleus Agent controller, Golgi routing, Lysosome comparator array, inter-cell NoC) with a commercial RRAM accelerator card handling the VMM operations. Several companies (Mythic, Untether AI, and research spinoffs from IBM and Stanford) have RRAM or PCM accelerator cards that can be used for this purpose. The FPGA handles the digital control plane; the RRAM card handles the analog crossbar VMM. This is not a fully integrated CIM-Vossel chip, but it is sufficient to: demonstrate the end-to-end event-driven pipeline with real RRAM hardware in the loop; characterize latency and energy at the system level under realistic ISR sensor inputs; and validate the inter-cell communication protocols with multiple FPGA+RRAM boards representing a small vossel grid.

Phase 3 (2028–2030): Custom CIM-Vossel ASIC. The first true CIM-Vossel chip: a custom ASIC integrating RRAM crossbar banks, WOx state memory, digital Nucleus Agent controllers, and NoC fabric on a single die at 7-nm CMOS or better. The target is 2,000 CVCs per chip at $\sim$3 W, as projected in Section 5.4. The chip design should target the tactical UAS scenario first (most constrained SWaP, highest near-term operational demand) and should integrate a radiation-hardening variant for the small satellite scenario. This is the phase at which the CIM-Vossel architecture becomes a hardware product rather than a research demonstration.

Phase 4 (2030+): Full neuromorphic CIM-Vossel chip with on-chip learning. The long-term target is a chip that supports not only inference but online RL-driven weight adaptation in the RRAM crossbars. This requires: RRAM device endurance of at least 10^7 write cycles (approximately 5× the current best-demonstrated); in-situ gradient computation hardware (either analog or mixed-signal, building on the IBM HfOx disturbance-resilient work; Choi et al., 2025); and a modified Nucleus Agent controller that orchestrates the RL policy update cycle. With on-chip learning, the CIM-Vossel can adapt its SSM weight matrices based on mission experience, implement the epigenetic modification mechanism described in Passmore (2026), and close the loop between the RL policy and the hardware processing substrate. This is the full realization of the Computational Vossel's three laws on physical hardware.

8.3 Broader Implications

The convergence documented in this paper is narrowly about two specific architectures: my Computational Vossel work and the Zhang et al. CIM-SSM hardware. But I think the convergence points to something larger that is worth saying explicitly.

Independent rediscovery of the same solution. The CIM-SSM hardware was not designed with vossels in mind. Zhang et al. were motivated by the specific mismatch between SNN binary activations and RRAM VMM capabilities, and by the desire to implement SSM state decay through device physics rather than digital logic. My Computational Vossel work was motivated by the need for a fidelity-preserving, locally-computable data structure for ISR sensor fusion, drawing on cell biology as the existence proof that storage-compute fusion is physically realizable. We arrived at essentially the same architectural solution (state stored in device physics, computation performed in-place via crossbar VMM, locality of processing, asynchronous event-driven activation) from completely different starting points. This kind of convergent design is not coincidental; it reflects a deeper truth about what efficient information processing looks like when you optimize hard enough against energy and latency constraints.

The cell's 4-billion-year solution. The biological cell solved the data-compute problem under extreme resource constraints long before von Neumann formalized the separation of memory and computation in 1945. The cell's solution: store the processing machinery (enzymes) co-located with the data (substrates), communicate via diffusing molecules rather than wired buses, adapt processing capacity by synthesis and degradation of agents, and implement

feedback and memory via covalent modification of molecules rather than register file writes. The RRAM crossbar is an enzyme (fixed weights, in-place computation). The WOx memristor array is molecular memory (state stored as conductance, relaxing through physics). The vossel's seven agent classes are the organelles. The mipvol hierarchy is the organ system. This is not metaphor; it is functional isomorphism.

From von Neumann to biology, and back. Von Neumann's EDVAC architecture (von Neumann, 1945) was a clean engineering choice for the technology of 1945: memory was expensive, so you minimized it; computation was rare, so you concentrated it; you moved data to the processor. Seventy-five years later, computation is nearly free (0.1 pJ per operation; Horowitz, 2014) and memory access is expensive (1,000–2,000 pJ per DRAM access). The economics have inverted: the right answer now is to move the computation to the data, exactly as biology does. The CIM hardware community has been converging on this insight from the hardware side; the Computational Vossel provides a software architecture that is explicitly designed around it. The confluence documented in this paper is, in a real sense, the point where the hardware and software convergences meet.

Implications for SSM architecture. One specific implication of the CIM-Vossel convergence is that selective SSMs (Mamba-style, with input-dependent parameters; Gu and Dao, 2024) are the natural next step. The current CIM-SSM architecture uses fixed decay rates λ per block, which limits the adaptive dynamics the vossel architecture requires. Selective SSMs allow the effective decay timescale to vary with the input, enabling content-dependent temporal integration. Implementing selective SSMs on CIM hardware requires that the Δt scaling factor be computed dynamically, which reintroduces some digital overhead (the selective parameter computation is input-dependent and therefore not amenable to static crossbar weights). But the MamCIMFlow architecture (Li et al., 2025) shows that Mamba-2 selective SSMs can be mapped to RRAM-CIM with 14.1× speedup and 319.3× power efficiency improvement over GPU baseline, suggesting that the selective SSM extension of the CIM-Vossel architecture is technically achievable. When selective SSMs are implemented on CIM hardware, the vossel's Nucleus Agent could, in principle, modulate the decay parameter Δt through the digital controller, approximating the adaptive dynamics that fixed-rate WOx devices cannot provide directly.

Distributed cognition. The multi-platform federated vossel network (Section 7.4) represents a form of distributed cognition at the tactical edge: many platforms, each maintaining a local world model in their CIM-Vossel arrays, sharing processed intelligence (not raw data) to build a collective picture of the operational environment. This is not a new idea in concept (it is the vision behind programs like DARPA's Mosaic Warfare and the Anduril Lattice Mesh), but it has previously required cloud-connected processing to be practically realizable. The CIM-Vossel architecture makes it realizable at the platform level, without cloud connectivity, within a SWaP budget that fits on a Group 2 UAS. That is a qualitative change in what is possible at the tactical edge, not merely a quantitative improvement.

8.4 Conclusion

The convergence argued in this paper is between two bodies of work that were developed independently and for different reasons. My Computational Vossel architecture was motivated by the biology of eukaryotic cells as an existence proof of storage-compute fusion; Zhang et al.'s CIM-SSM hardware was motivated by the physics of RRAM crossbars and WOx memristors as substrates for event-driven state evolution. The formal mapping in Section 4 and the physical design in Section 5 establish that these two architectures are not merely compatible but are complementary halves of a complete system: the vossel provides the software abstraction and the multi-agent interaction model; the CIM-SSM hardware provides the physical substrate that makes the abstraction implementable at the energy and latency scales required for tactical edge ISR.

The current state of the art is early. NeuRRAM at 130-nm and the Zhang et al. prototype at 65-nm are proof-of-concept demonstrations, not deployable systems. The scaling analysis in Section 5.4 shows that meaningful vossel grids (thousands of CVCs) become achievable at 7-nm around 2027–2028, and the roadmap in Section 8.2 identifies the experimental milestones that must be passed to get there. There are real unsolved problems: the training-versus-

inference gap in CIM hardware, the incomplete mapping of the full seven-agent vossel model, and the WOx decay-rate tunability limit. These are the problems my lab and the wider CIM hardware community need to work on next.

What this paper does is draw the map. The two threads (biomimetic software architecture and physics-native CIM hardware) are converging. The researchers building CIM-SSM hardware need to know about the vossel architecture's requirements for multi-agent interaction, mipvol aggregation, and online learning, so they can design hardware that supports these operations rather than only the single-layer inference pipelines demonstrated so far. The researchers building vossel software frameworks need to know about the WOx decay-rate constraints and the RRAM VMM precision limits, so they can design agent interaction models that remain accurate under realistic hardware non-idealities. This paper is a first attempt to put those requirements and constraints in conversation with each other.

References

Ambrogio, S., Narayanan, P., Okazaki, A., et al. (2023). An analog-AI chip for energy-efficient speech recognition and transcription. *Nature*, 620, 768–775. https://doi.org/10.1038/s41586-023-06337-5

Arteta, A., Diaz-Flores, E., de Mingo López, L. F., and Gómez Blas, N. (2021). An in vivo proposal of cell computing inspired by membrane computing. *Processes*, 9(3), 511. https://doi.org/10.3390/PR9030511

Bian, S., Schulthess, L., Rutishauser, G., et al. (2023). ColibriUAV: An ultra-fast, energy-efficient neuromorphic edge processing UAV-platform with event-based and frame-based cameras. *IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*. https://doi.org/10.1109/IWASI58316.2023.10164354

Chithambara Moorthii, J., Rayapati, V., Rao, N., and Suri, M. (2024). VPU-CIM: A 130nm, 33.98 TOPS/W RRAM based compute-in-memory vector co-processor. *IEEE ISCAS 2024*. https://doi.org/10.1109/ISCAS58744.2024.10558155

Choi, W., Stecconi, T., Falcone, D., et al. (2025). Update disturbance-resilient analog ReRAM crossbar arrays for in-memory deep learning accelerators. *Advanced Science*, 12. https://doi.org/10.1002/advs.202504578

DARPA. (2025). DARPA to demonstrate revolutionary drone capabilities for warfighters (EVADE program). https://www.darpa.mil/news/2025/evade-drone-capabilities-warfighters

DoD CDAO / Anduril Industries. (2024). CDAO awards Anduril production agreement to deliver edge data mesh. https://www.asdnews.com/news/defense/2024/12/03/cdao-awards-anduril-production-agreement-deliver-edge-data-mesh

Epstein, J. M. and Axtell, R. (1996). *Growing Artificial Societies: Social Science from the Bottom Up*. MIT Press / Brookings Institution Press.

ESA / eoPortal. (2024). Onboard data processing: Phi-sat-2 mission overview. https://www.eoportal.org/other-space-activities/onboard-data-processing

Gu, A., Goel, K., and Ré, C. (2022). Efficiently modeling long sequences with structured state spaces. *International Conference on Learning Representations (ICLR 2022)*. https://arxiv.org/abs/2111.00396

Gu, A. and Dao, T. (2024). Mamba: Linear-time sequence modeling with selective state spaces. *ICML 2024*. https://arxiv.org/abs/2312.00752

Hajizada, E., Rager, D., Shea, T., et al. (2025). Real-time continual learning on Intel Loihi 2. *arXiv:2511.01553*. https://doi.org/10.48550/arXiv.2511.01553

Hong, H., et al. (2025). Memristor-based adaptive analog-to-digital conversion for efficient and accurate compute-in-memory. *Nature Communications*. https://doi.org/10.1038/s41467-025-65233-w

Horowitz, M. (2014). Computing's energy problem (and what we can do about it). *IEEE International Solid-State Circuits Conference (ISSCC) Plenary*. https://gwern.net/doc/cs/hardware/2014-horowitz-2.pdf

Khacef, L., Weidel, P., Hogyoku, S., et al. (2025). Privacy-preserving fall detection at the edge using Sony IMX636 event-based vision sensor and Intel Loihi 2 neuromorphic processor. *arXiv:2511.22554*. https://doi.org/10.48550/arXiv.2511.22554

Koubâa, A., Ammar, A., Ghouti, L., et al. (2023). AERO: AI-enabled remote sensing observation with onboard edge computing in UAVs. *Remote Sensing*, 15(7), 1873. https://doi.org/10.3390/rs15071873

Le Gallo, M., Khaddam-Aljameh, R., Stanisavljevic, M., et al. (2023). A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference. Nature Electronics, 6, 680–693. https://doi.org/10.1038/s41928-023-01010-1

Li, M., Wang, Z., Ye, Z., Pan, T., and Wang, H. (2025). MamCIMFlow: An integrated co-design of RRAM-based CIM and selective state-space streaming for efficient Mamba model acceleration. *IEEE ICCD 2025*. https://doi.org/10.1109/ICCD65941.2025.00041

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197), 668–673. https://doi.org/10.1126/science.1254642

Modha, D. S., et al. (2023). Neural inference at the frontier of energy, space, and time. *Science*, 382, 329–335. https://doi.org/10.1126/science.adh1174

Modha, D. S., et al. (2024). Breakthrough edge AI inference performance using NorthPole in 3U VPX form factor. *IEEE High Performance Extreme Computing (HPEC 2024)*. https://modha.org/wp-content/uploads/2024/09/NorthPole_HPEC_VPX.pdf

Nagy, Á., Szabó, R., and Toka, L. (2025). Analyzing energy consumption of Loihi 2 neuromorphic chip in a self-driving use-case. *SpliTech 2025*. https://doi.org/10.23919/SpliTech65624.2025.11091800

Passmore, G. (2025). The Vossel Framework: Fidelity-Preserving Volumetric Representations for Multi-Modal ISR Sensor Fusion. [Published work, Passmore Research Group].

Passmore, G. (2026). *Computational Vossels: A Biomimetic Architecture for Storage-Compute Fusion in Intelligence, Surveillance, and Reconnaissance Systems*. [Published monograph, Passmore Research Group].

Smith, J. T. H., Warrington, A., and Linderman, S. (2023). Simplified state space layers for sequence modeling. *International Conference on Learning Representations (ICLR 2023)*. https://arxiv.org/abs/2208.04933

STAR.VISION Aerospace Group. (2025). AI-enabled onboard edge computing for satellite intelligence in disaster management. *UN-SPIDER News*. https://www.un-spider.org/news-and-events/news/ai-enabled-onboard-edge-computing-satellite-intelligence-disaster-management

von Neumann, J. (1945). First draft of a report on the EDVAC. Technical Report, Moore School of Electrical Engineering, University of Pennsylvania. [Widely reprinted; historical primary source.]

Wan, W., et al. (2022). A compute-in-memory chip based on resistive random-access memory. *Nature*, 608, 504–512. https://doi.org/10.1038/s41586-022-04992-8

Zhang, X., Hu, M., Mingtao, H., et al. (2026). Compute-in-memory implementation of state space models for event sequence processing. *Nature Communications*, 17, 5584. https://doi.org/10.1038/s41467-025-68227-w