Catalyst N2: Full Loihi 2 Feature Parity in an Open Neuromorphic Processor with Programmable Neuron Microcode and Cloud FPGA Validation

Henry Arthur Shulayev Barnes
School of Natural and Computing Sciences
University of Aberdeen
Aberdeen, AB24 3UE, United Kingdom
u13hs24@abdn.ac.uk

Abstract

I present **Catalyst N2**, the second generation of the Catalyst neuromorphic processor architecture, achieving full feature parity with Intel's Loihi 2. Building on the Catalyst N1 foundation of 128 cores with 131K neurons and Loihi 1 equivalence, N2 introduces a programmable neuron microcode engine that replaces the fixed CUBA integrate-and-fire model with user-defined neuron dynamics—enabling Izhikevich, adaptive LIF, sigma-delta, and resonate-and-fire models to execute natively on the same silicon. Four graded spike payload formats (binary through 24-bit), variable-precision weight packing (1–16 bit), convolutional synapse encoding, per-synapse-group plasticity control, and a hierarchical dendritic compartment tree with per-dendrite thresholds round out the architectural extensions. The accompanying SDK has grown from 14 modules and 168 tests (N1) to 88 modules and 3,091 tests across three backends: a cycle-accurate CPU simulator with hardware-accurate fixed-point defaults, a GPU simulator with stochastic rounding in the learning path for silicon-faithful weight evolution, and an FPGA hardware backend. Hardware-accurate defaults—24-bit fixed-point arithmetic, strict SRAM budget enforcement, and 7-bit learning trace registers—are enabled by default, ensuring that every simulation faithfully represents what the hardware would produce. I validate a 16-core instance on an AWS F2 Xilinx VU47P at 62.5 MHz, achieving 28/28 (100%) pass rate across seven test categories spanning single-neuron dynamics, synaptic connectivity, network topologies, multi-core routing, scale, and operational correctness. Feature coverage analysis identifies 155 Loihi 2 features, of which 152 are fully implemented and 3 are hardware-only (requiring physical multi-chip links). The Spiking Heidelberg Digits benchmark from N1 is preserved at 85.9% accuracy. Catalyst N2 is, to my knowledge, the first independently developed neuromorphic architecture to achieve comprehensive Loihi 2 parity and the first to be validated on cloud FPGA infrastructure.

1 Introduction

In the preceding paper [1], I introduced Catalyst N1: an open neuromorphic processor achieving architectural feature parity with Intel's Loihi 1 [2]. That work demonstrated 128 cores of 1,024 CUBA LIF neurons, 131K CSR synapses per core, a programmable synaptic microcode learning engine, and FPGA validation on AWS F2 with 98 RTL test scenarios at zero failures. The accompanying SDK provided three interchangeable backends and achieved 85.9% on the SHD spoken digit benchmark.

N1 was a necessary foundation. But Loihi 1 is no longer Intel's current architecture. Loihi 2 [3], fabricated in Intel 4, introduced programmable neuron dynamics through a microcode engine, variable-precision synapses, convolutional encoding, and a substantially richer set of plasticity and control features. Where Loihi 1 hardcoded the CUBA neuron model and offered a single synapse format, Loihi 2 treats the neuron itself as a programmable unit—a qualitative shift in architectural philosophy.

This paper presents Catalyst N2, which closes that gap. The core contribution is not merely additive feature accumulation but a systematic effort to match the *design philosophy* of Loihi 2: neurons as programmable compute units, synapses as configurable memory structures, and learning as a composable system of traces, eligibility, and modulation. The result is an architecture where five distinct neuron models coexist on the same core, where weight precision can be tuned per connection group from 1 to 16 bits, and where the learning engine's interaction with the neuron model is mediated through shared register files rather than hardcoded signal paths.

Relationship to N1. N2 is not a clean-sheet redesign. The RTL retains N1's core structure: the 35-state FSM pipeline, the 51-SRAM memory hierarchy, the barrier mesh and async NoC, the triple RV32IMF cluster, and the multi-chip link infrastructure. What changes is the programmability surface exposed to software. The neuron update phase, previously a fixed CUBA datapath, now dispatches to a per-neuron microcode program. The synapse delivery phase, previously

limited to three formats, gains convolutional encoding and variable-precision weight packing. The learning engine, previously constrained to 8-bit trace registers, now distinguishes 7-bit learning traces from 8-bit neuron microcode registers—matching the Loihi 2 hardware register file widths exactly.

Design philosophy. Three principles guided the N2 design:

1. **Programmability over fixed function.** Where N1 hardcoded behaviors (CUBA neuron model, 16-bit weights, single payload format), N2 parameterizes them. The cost is modest additional hardware (a program counter, a register file, multiplexers for format selection) relative to the flexibility gained.
2. **Hardware-accurate simulation by default.** N1's SDK defaulted to unbounded arithmetic for developer convenience. This created a gap between what the simulator predicted and what the hardware would do. N2 inverts this: the simulator defaults to hardware-accurate mode, and developers must explicitly opt out for debugging. This "hardware-first" approach means that any network that runs in simulation *will* run on hardware without surprises.
3. **Silicon-faithful learning.** Machine learning researchers care about weight evolution trajectories, not just final inference accuracy. If the GPU simulator uses float32 truncation while the CPU simulator uses integer stochastic rounding, trained models will diverge. N2 closes this gap by implementing stochastic rounding in the GPU learning path, ensuring that training on GPU produces the same weight distribution as training on the CPU reference.

Contributions. The specific contributions of this work are:

- A programmable neuron microcode engine with a dedicated instruction set, per-neuron program selection, and dendritic register access, enabling five neuron models (CUBA, Izhikevich, adaptive LIF, sigma-delta, resonate-and-fire) to execute natively without RTL modification.
- Four graded spike payload formats (binary, 8-bit, 16-bit, 24-bit) configurable per core, variable-precision weight packing (1/2/4/8/16-bit per synapse group), and convolutional synapse encoding with weight-shared kernels.
- Per-synapse-group plasticity enable bits, advanced multi-trace learning (five independent trace time constants), persistent reward traces with configurable decay, and homeostatic threshold plasticity with epoch-based rate targeting.
- An SDK of 88 Python modules with 3,091 tests, hardware-accurate fixed-point defaults, GPU learning fidelity (stochastic rounding, weight quantization matching the CPU reference), and comprehensive state monitoring and checkpointing.
- FPGA validation on AWS F2 (Xilinx VU47P, 16 cores, 62.5 MHz) achieving 28/28 test pass rate across seven categories, with characterization of BRAM initialization behavior and inter-deployment state management.
- A systematic feature parity assessment identifying 155 Loihi 2 features, of which 152 are fully implemented—the three remaining features require physical multi-chip hardware.

Paper organization. Section 2 briefly recaps the N1 baseline. Section 3 presents the programmable neuron microcode engine. Section 4 covers synaptic extensions. Section 5 details learning and plasticity enhancements. Section 6 describes debug and power management features. Section 7 presents the SDK evolution. Section 8 covers FPGA hardware validation. Section 9 evaluates feature parity and compares N1 to N2. Section 10 surveys related work. Section 11 discusses limitations honestly, and Section 12 concludes.

2 N1 Baseline

For readers unfamiliar with the first paper [1], I summarize the N1 architecture that serves as N2's foundation.

2.1 Core Architecture

Catalyst N1 implements 128 neuromorphic cores in a configurable mesh interconnect. Each core contains 1,024 CUBA LIF neurons with 24-bit state precision, 131K compressed sparse row synapses, a 35-state FSM pipeline (delay drain $\rightarrow$ spike delivery $\rightarrow$ neuron update $\rightarrow$ microcode learning $\rightarrow$ spike output), and approximately 51 SRAMs totaling 1.2 MB per core. A programmable *synaptic* learning engine with 16 registers and 14 opcodes supports STDP, three-factor eligibility-modulated reward learning, and homeostatic weight normalization through user-loaded microcode programs.

Three synapse encoding formats (sparse CSR, dense, population-coded), per-synapse axon delays up to 63 timesteps, four-compartment dendritic trees with configurable join operations, stochastic threshold noise via per-neuron LFSRs, and dual spike traces (x_1, x_2, pre/post-synaptic) complete the per-core feature set.

2.2 Infrastructure

The interconnect supports two modes selectable at synthesis time: barrier mesh (deterministic, lockstep) and asynchronous packet NoC (event-driven, XY dimension-order routing). A triple RV32IMF RISC-V embedded cluster provides supervisory control with IEEE 754 floating-point, timer interrupts, and hardware breakpoints. Four multi-chip link ports support 14-bit addressing across up to 16,384 chips.

2.3 SDK and Validation

The N1 SDK provides a Python network builder, a density-weighted compiler, and three backends (cycle-accurate CPU simulator, GPU simulator with PyTorch sparse CSR operations, and FPGA hardware backend). N1 was validated on AWS F2 with 25 RTL testbenches covering 98 scenarios at

zero failures, and achieved 85.9% on SHD with surrogate gradient training and 16-bit weight quantization.

Table 6 in Section 9 provides a detailed N1-to-N2 comparison across all architectural dimensions.

3 Programmable Neuron Microcode Engine

The most significant architectural addition in N2 is the neuron microcode engine. Where N1 hardcoded the CUBA neuron model—a fixed sequence of operations computing current decay, voltage decay, bias addition, threshold comparison, and spike emission—N2 replaces this fixed datapath with a programmable execution unit that runs user-defined programs on every neuron at every timestep.

This mirrors Loihi 2's architectural philosophy: the neuron is no longer a fixed-function unit but a small programmable processor. The advantage is profound. New neuron models can be deployed without RTL changes, different neuron types can coexist on the same core, and researchers can experiment with novel dynamics without hardware modification.

3.1 Execution Model

The neuron microcode engine executes during the neuron update phase of the core pipeline. When microcode execution is enabled (a global per-core configuration bit), the fixed CUBA datapath is bypassed and execution dispatches to the microcode interpreter instead. The FSM extends with three new states: program load (fetching the start address from the per-neuron offset register), instruction fetch (reading from the instruction SRAM), and execute (performing the register operation and advancing the program counter).

Each core contains a dedicated instruction SRAM storing the neuron programs. Programs are compact sequences of register-to-register operations terminated by an explicit halt instruction. A per-neuron offset register (stored in a separate SRAM alongside the neuron parameters) selects which program to execute, enabling distinct programs per core with arbitrary neuron-to-program mappings. In practice, most networks use 1–5 programs per core, with the compiler deduplicating identical programs and allocating offsets through a bump allocator.

The instruction format encodes an opcode, destination register, two source registers, and a shift amount in a fixed-width word. The instruction set provides arithmetic operations (addition, subtraction, multiply-with-shift, bitwise shifts, min, max, absolute value), conditional skips (skip-if-zero, skip-if-nonzero, skip-if-greater-or-equal, skip-if-less-than), and two termination instructions. I deliberately omit the exact encoding and opcode assignments, as these are implementation details that vary between revisions and are not necessary for understanding the architectural contribution.

3.2 Register File

The register file exposes the neuron's complete state to the program. Sixteen registers are mapped as follows:

- Registers 0–5 map to the neuron's primary state: membrane voltage, synaptic current, dendritic accumulator, firing threshold, bias, and resting potential.
- Registers 6–7 provide noise samples from the per-neuron LFSR.
- Registers 8–9 hold the decay step counts (derived from the decay parameters and the current register values via the RAZ function).
- Register 10 holds the scaled synaptic input for the current timestep.
- Registers 11–15 are temporaries, available for intermediate computation.

After program execution, modified register values are written back to the neuron state SRAMs, maintaining consistency with the rest of the pipeline. The write-back is selective: only registers corresponding to hardware state (voltage, current, threshold) are persisted; temporary registers are discarded between timesteps.

3.3 Termination and Spike Emission

Two special instructions provide control beyond arithmetic:

HALT. The halt instruction terminates microcode execution and implicitly performs threshold comparison: if the voltage register exceeds the threshold register plus the noise sample, a spike is emitted with the standard excess-over-threshold payload. The neuron then enters its refractory period and the voltage is reset. This instruction makes the common case—a neuron that computes some dynamics and then fires conventionally—as compact as possible: a single instruction handles comparison, spike emission, refractory entry, and voltage reset.

EMIT. The emit instruction forces a spike with a payload value read from a specified register rather than the default excess-over-threshold encoding. After emission, the neuron enters refractory normally. The emit instruction is essential for sigma-delta neurons, which encode their output as a difference signal, and for any neuron model where the spike payload should carry information other than the voltage exceedance.

When neither HALT nor EMIT is reached before the program counter exceeds the instruction SRAM bounds, execution terminates with no spike—equivalent to a neuron that computed its dynamics but did not reach threshold. This implicit termination is a safety mechanism preventing infinite loops.

3.4 Dendritic Register Access

When dendritic microcode mode is enabled (a separate per-core configuration bit), three additional registers are populated

with the dendritic accumulator values from compartments 1–3, sign-extended to the full register width. This allows the neuron program to implement arbitrary nonlinear interactions between dendritic inputs and somatic state—for example, multiplicative gating where dendritic input modulates the gain of the somatic response:

$$v[t+1] = v[t] + \frac{I_{\text{soma}} \cdot I_{\text{dend1}}}{2^s} \tag{1}$$

This capability has no equivalent in the N1 architecture, where dendritic compartments could only combine through the four fixed join operations (ADD, ABS_MAX, OR, PASS). The combination of per-neuron program selection and dendritic register access enables architectures where different dendritic branches serve different computational roles—one branch providing excitatory drive, another providing modulatory gain control—within a single neuron.

3.5 Integration with Existing Pipeline

The neuron microcode engine must coexist with the rest of the core pipeline without disrupting the established timing. Three integration points required careful design:

Backward compatibility. When neuron microcode is disabled (the default for N1-compatible networks), the core reverts entirely to the fixed CUBA datapath. No performance penalty is incurred, and the FSM skips the three microcode states entirely. This is not merely a configuration flag—the hardware path through the CUBA update logic is physically distinct from the microcode path, ensuring that legacy networks see zero regression.

Refractory interaction. Both the fixed CUBA path and the microcode path must respect the neuron's refractory state. The refractory counter is decremented *before* microcode execution begins. If the neuron is still in its refractory period, the microcode program still executes (allowing the neuron to update internal state), but the HALT and EMIT instructions are inhibited—no spike can be emitted during refractory.

Trace and learning interaction. After neuron update (whether via CUBA or microcode), the spike trace update and learning engine phases execute normally. The learning engine reads the neuron's spike status and trace values regardless of which path produced them. This means that all learning rules—STDP, three-factor, custom microcode—work with all neuron models without modification.

3.6 Neuron Model Library

Five neuron models ship with the SDK, each implemented as a microcode program generator that produces the instruction sequence for a given parameter set. The mathematical formulations below describe the dynamics; the actual microcode implementations use fixed-point integer arithmetic with appropriate scaling.

CUBA LIF. The default model, functionally identical to N1's hardcoded datapath:

$$u[t+1] = u[t] - \text{RAZ}\left(\frac{u[t] \cdot \delta_u}{4096}\right) + I_{\text{syn}}[t] \tag{2}$$

$$v[t+1] = v[t] - \text{RAZ}\left(\frac{v[t] \cdot \delta_v}{4096}\right) + u[t] + b \tag{3}$$

$$\text{spike} \iff v[t+1] \geq \theta + \eta[t] \tag{4}$$

where δ_u and δ_v are 12-bit unsigned decay constants, b is a bias term with mantissa-exponent compression, and $\eta[t]$ is stochastic noise. The RAZ (round-away-from-zero) function ensures small decay products always make progress toward zero. The CUBA microcode program reproduces this arithmetic exactly, including the RAZ rounding behavior, ensuring bit-identical results with the N1 fixed datapath.

Izhikevich. The Izhikevich model [7] captures a rich repertoire of neural dynamics—regular spiking, intrinsically bursting, chattering, fast spiking, low-threshold spiking, and more—through a two-variable system with quadratic voltage nonlinearity:

$$v[t+1] = v[t] + \frac{v[t]^2}{2^{s_v}} + I[t] - w[t] + b_v \tag{5}$$

$$w[t+1] = w[t] + \frac{a \cdot (b \cdot v[t] - w[t])}{2^{s_w}} \tag{6}$$

where w is a recovery variable (stored in the synaptic current register), a controls the recovery time constant, b controls the sensitivity of w to subthreshold voltage oscillations, and s_v, s_w are scaling shifts for fixed-point representation. On spike, both variables are reset: $v \leftarrow c$ (reset voltage), $w \leftarrow w + d$ (recovery increment). The quadratic v^2 term is computed via the multiply-with-shift instruction.

The choice of (a, b, c, d) selects the dynamical regime. The SDK provides named presets: `regular_spiking` ($a = 2, b = 26, c = -6500, d = 800$), `intrinsic_burst` ($a = 2, b = 26, c = -5500, d = 400$), `chattering` ($a = 2, b = 26, c = -5000, d = 200$), `fast_spiking` ($a = 10, b = 25, c = -6500, d = 200$). These values are scaled for the hardware's 24-bit integer arithmetic.

Adaptive LIF (ALIF). The adaptive leaky integrate-and-fire model adds a slow adaptation current that increases on each spike and decays between spikes:

$$v[t+1] = v[t] - \text{RAZ}\left(\frac{v \cdot \delta_v}{4096}\right) + I[t] - w_{\text{adapt}}[t] \tag{7}$$

$$w_{\text{adapt}}[t+1] = w_{\text{adapt}}[t] - \frac{w_{\text{adapt}}[t]}{2^{s_a}} \tag{8}$$

On spike, $w_{\text{adapt}} \leftarrow w_{\text{adapt}} + \beta$, where β controls the adaptation strength and s_a controls the adaptation time constant. This

produces spike-frequency adaptation: initial bursts followed by steady-state firing at a lower rate, matching the behavior observed in cortical pyramidal neurons [8]. ALIF is particularly useful in recurrent networks where it provides intrinsic regularization against runaway activity.

Sigma-Delta. The sigma-delta neuron [9] maintains a running estimate of its input and transmits only the prediction error as a graded spike:

$$\Delta[t] = I[t] - \hat{I}[t-1] \tag{9}$$

$$\hat{I}[t] = \hat{I}[t-1] + \Delta[t] \tag{10}$$

The emit instruction transmits Δ as the spike payload, bypassing the threshold-exceedance encoding. This model achieves temporal sparsity by transmitting only *changes* in the input signal. For signals with high temporal autocorrelation (e.g., sensor readings, audio features), the number of non-trivial spikes is proportional to the signal bandwidth rather than the signal amplitude.

Resonate-and-Fire. The resonate-and-fire neuron [10] implements damped oscillatory subthreshold dynamics:

$$v[t+1] = v[t] + \frac{\omega \cdot u[t]}{2^s} - \frac{d \cdot v[t]}{2^s} + I[t] \tag{11}$$

$$u[t+1] = u[t] - \frac{\omega \cdot v[t]}{2^s} - \frac{d \cdot u[t]}{2^s} \tag{12}$$

where ω controls the resonant frequency and d the damping. The coupled (v, u) system forms a damped oscillator; when driven at its resonant frequency, the amplitude grows until it crosses threshold. Inputs at other frequencies do not accumulate coherently and fail to fire the neuron. This provides frequency-selective filtering without explicit spectral computation, useful for auditory processing and rhythm detection tasks.

Per-neuron model selection. Because each neuron carries an independent program offset, different models can coexist on the same core. A network might use CUBA neurons in sensory layers (for computational efficiency), Izhikevich neurons in recurrent layers (for rich temporal dynamics), and sigma-delta neurons at output interfaces (for bandwidth-efficient readout). The compiler handles program allocation and offset assignment transparently—the user simply specifies the model per population, and the compiler deduplicates identical programs across populations sharing the same model and parameters.

4 Synaptic Architecture Extensions

N2 extends N1's synaptic subsystem along three dimensions: payload format flexibility, weight precision control, and a new encoding format for structured connectivity.

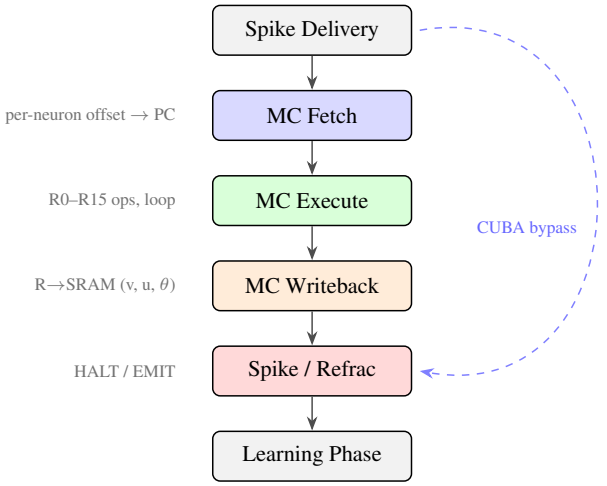

Figure 1: Neuron microcode integration within the core pipeline. When enabled, the three MC states replace the fixed CUBA datapath. The dashed bypass restores N1 behavior when disabled.

4.1 Graded Spike Payload Formats

N1 supported a single graded spike format: an 8-bit payload encoding the voltage excess above threshold. N2 generalizes this to four formats configurable per core:

- **Binary (0-bit):** Traditional all-or-nothing spikes. No payload field is transmitted. This is the most bandwidth-efficient format for networks that use only spike timing for information coding.
- **8-bit:** N1-compatible. Payload encodes $\min(255, \max(1, v - \theta))$. Suitable for most networks using graded rate or amplitude coding.
- **16-bit:** Extended precision for networks requiring fine-grained amplitude resolution. The payload carries the lower 16 bits of the voltage-threshold difference.
- **24-bit:** Full-precision payload matching the neuron's internal state width. Used primarily with sigma-delta neurons where the transmitted value requires the full dynamic range of the integer state variable.

The delivered synaptic current scales with the payload via the grade-shift mechanism inherited from N1: $I_{\text{delivered}} = (w \cdot \text{payload}) \gg G$, where G is the configurable grade shift (default 7). Wider payloads produce proportionally more precise current injection but consume more routing bandwidth in the inter-core spike packets.

The format is selected per core, not per neuron, because the spike packet routing infrastructure uses a fixed-width field for the payload. Mixing formats within a core would require variable-length packets, which would complicate the router design disproportionately to the benefit gained.

4.2 Variable-Precision Weight Packing

N1 stored all synaptic weights as 16-bit signed integers, regardless of the effective precision required. For many networks, particularly those using quantization-aware training or

binary/ternary weight representations, this wastes memory. N2 introduces variable-precision weight packing at five granularities: 1, 2, 4, 8, and 16 bits per weight.

The compression is applied per synapse group (a contiguous block of pool entries sharing the same source neuron). Each 16-bit pool word stores $16/b$ weights at b-bit precision, achieving compression ratios from $1\times$ (16-bit, no compression) to $16\times$ (1-bit, binary weights). Table 1 summarizes the options.

Table 1: Variable-precision weight packing.

Bits	Ratio	Range	Use case
1	$16\times$	$\{-1, +1\}$	Binary SNNs
2	$8\times$	$[-2, +1]$	Ternary weights
4	$4\times$	$[-8, +7]$	QAT-4 networks
8	$2\times$	$[-128, +127]$	Standard quantized
16	$1\times$	$[-32768, +32767]$	Full precision

The decompression logic in the synapse delivery phase unpacks weights on-the-fly using a barrel shifter indexed by the intra-word position. No additional SRAM is required—the packing is purely a reinterpretation of the existing pool entry format. A per-group precision tag stored in the index table controls the unpacking behavior. A shared scale factor (the weight exponent from N1) applies after decompression to restore the effective dynamic range.

For a fully-connected layer with 1,024 input and 1,024 output neurons at 4-bit precision, the pool usage is $1{,}024 \times 1{,}024/4 = 262{,}144$ entries—exactly $2\times$ the pool depth (131K), requiring two cores. At 16-bit precision, the same layer would require $1{,}024 \times 1{,}024 = 1{,}048{,}576$ entries—8 cores. Variable-precision packing thus has a direct impact on core utilization for large networks.

4.3 Convolutional Synapse Encoding

For convolutional neural network (CNN) layers deployed on spiking hardware, N1's sparse CSR format is wasteful: every connection stores an explicit target neuron ID and individual weight, even when the weights are shared across a spatial kernel. N2 adds a convolutional encoding format (FMT_CONV) where each pool entry stores a *relative target offset* and a reference to a shared weight kernel.

The kernel itself is stored once per group in a compact header region of the connection pool. Each subsequent synapse entry requires only the relative offset (the spatial displacement from the source to the target neuron within the output feature map), a kernel index, and a compartment selector. For a 3×3 kernel with 64 output channels, per-synapse storage drops from 32 bits (explicit weight + explicit target) to approximately 12 bits (offset + kernel index), yielding a 2–3$\times$ memory reduction for typical convolutional topologies.

The compiler automatically detects convolutional connection patterns (when a `Conv2d` topology is specified) and selects FMT_CONV encoding, falling back to sparse CSR for irregular connectivity. The detection is based on weight sharing analysis: if multiple connections share the same weight value in a spatially regular pattern, FMT_CONV is selected.

5 Learning and Plasticity Enhancements

N2 significantly extends N1's learning subsystem. The synaptic microcode engine from N1 is preserved unchanged—it remains the programmable engine for weight update rules. What changes is the infrastructure surrounding it: trace management, eligibility dynamics, plasticity granularity, and homeostatic regulation.

5.1 Per-Synapse-Group Plasticity Enable

In N1, learning was an all-or-nothing per-core setting: either all synapses participated in the learning phase, or none did. This is inefficient for networks where only a subset of connections should be plastic (e.g., input-to-hidden connections are trained while hidden-to-output connections use fixed weights).

N2 adds a per-synapse-group `learn_en` bit stored in the MSB of the index table entry. During the learning phase, the microcode engine skips synapse groups where `learn_en = 0`, executing learning programs only on plastic connections. This typically reduces learning phase duration by 30–70% in networks with mixed fixed and plastic layers.

A complementary bitmap provides LTP-selective gating: groups can be configured to participate in LTD (pre-synaptic triggered) but not LTP (post-synaptic triggered), or vice versa. This enables one-directional learning rules and teacher-student configurations.

5.2 Five-Trace Learning System

N1 maintained two spike traces per neuron (x_1 and x_2). N2 extends this to five traces (x_1, x_2, y_1, y_2, y_3) with independently programmable time constants, matching Loihi 2's five-trace configuration. The three additional traces enable triplet STDP [11], which captures higher-order spike correlations that pair-based STDP cannot represent.

Each trace has an independent 4-bit decay shift controlling its time constant, giving $\tau \approx 2^{\text{shift}}$ timesteps. On a spike event, the corresponding trace saturates to its maximum value and subsequently decays exponentially with a guaranteed minimum step of 1 per timestep (the "min-step-1" guarantee prevents traces from stalling at small positive values due to integer truncation).

Learning vs. neuron trace register widths. A critical hardware fidelity detail: N2 distinguishes two trace register types. Neuron microcode registers (R11, R12 in the neuron update phase, used for threshold adjustment and custom neuron state) are 8-bit unsigned with maximum value 255. Learning engine traces (x_1, x_2, y_1, y_2, y_3, used in the synaptic microcode phase) are 7-bit unsigned with maximum value 127. This distinction matches the Loihi 2 hardware, where the learning trace register file is physically narrower than the neuron microcode register file. The spike impulse value (the value to which a

trace saturates on a spike event) is set to 127 for both register types, ensuring that the initial spike-triggered trace value fits within the 7-bit learning register.

The SDK enforces this distinction with separate constants (`FP_TRACE_MAX = 255` for microcode, `LEARNING_TRACE_MAX = 127` for learning) and applies the appropriate clamp at each usage site. This is a subtle but important point: using 255 as the learning trace maximum would produce trace values that overflow the 7-bit learning register on silicon, causing silent wraparound and corrupted learning dynamics.

5.3 Persistent Reward Traces

N1 implemented reward as a one-shot signal: the host or RISC-V cluster injected a scalar reward value that was consumed immediately during the eligibility-to-weight conversion. Between reward events, no reward information persisted.

N2 adds a configurable reward trace with exponential decay. The reward trace accumulates injected rewards and decays with a per-core time constant τ_r (a 4-bit shift, giving $\tau_r \approx 2^{\text{shift}}$ timesteps):

$$r[t] = r[t-1] - \text{sgn}(r) \cdot \max\left(1, \left\lfloor \frac{|r|}{2^{\tau_r}} \right\rfloor\right) + r_{\text{inj}}[t] \quad (13)$$

The reward trace allows temporally extended reward signals to modulate learning over multiple timesteps after a single reward injection—essential for reinforcement learning tasks where the reward is delayed relative to the causal actions. The decay follows the same "min-step-1" convention as spike traces, ensuring eventual convergence to zero. When $\tau_r = 0$ (the default), the reward trace behaves identically to N1's one-shot mode, preserving backward compatibility.

5.4 Homeostatic Threshold Plasticity

Biological neural circuits maintain firing rate stability through homeostatic mechanisms [12]. N2 implements epoch-based homeostatic threshold plasticity. At configurable intervals (the epoch period, 1–255 timesteps), each neuron's spike count is compared against a target rate, and the firing threshold is adjusted proportionally to the error:

$$\theta[k+1] = \text{clamp}\Big(\theta[k] + \eta \cdot (n_{\text{spikes}}[k] - n_{\text{target}}),\ \theta_{\min},\ \theta_{\max}\Big) \quad (14)$$

where k indexes epochs, η is the learning rate, n_{spikes} is the spike count within the epoch, and n_{target} is the per-neuron target rate. The clamping bounds prevent threshold runaway.

This proportional error formulation (which replaced an earlier sign-based $\pm\eta$ rule during the hardware fidelity audit) provides faster convergence and more stable equilibria. The sign-based rule suffered from oscillatory behavior near the target rate: a neuron firing $n_{\text{target}} + 1$ spikes received the same correction magnitude as one firing $10 \times n_{\text{target}}$ spikes.

Homeostasis operates independently of the synaptic learning engine and requires no microcode—it is implemented in the neuron update pipeline. This is important because homeostatic regulation must operate on a slower timescale than synaptic plasticity, and mixing the two in the same microcode engine would create timing conflicts.

6 Observability and Power Management

N2 adds infrastructure for runtime monitoring, debugging, and power-aware operation that N1 lacked entirely.

6.1 Performance Counters and Probes

Three per-run performance counters track: total spikes emitted, total synaptic operations performed (weight lookups during the delivery phase), and total active cycles (non-idle timesteps). These counters are readable through the host interface and are used by the energy metering system (Section 6.2).

A comprehensive probe interface provides read access to 25 internal state variables per neuron, including membrane voltage, synaptic current, refractory counter, all five trace values, dendritic accumulator values, threshold, bias, and derived quantities like the noise sample and accumulated input. Probes operate non-destructively: reading a probe does not alter the neuron's state or affect the simulation outcome.

A trace FIFO records spike timestamps for up to 64 events, enabling cycle-accurate spike timing analysis for small networks without the overhead of full state monitoring.

6.2 Energy Metering

N2 adds per-timestep energy estimation:

$$E[t] = c_{\text{cycle}} + c_{\text{spike}} \cdot n_{\text{spikes}}[t] + c_{\text{synop}} \cdot n_{\text{synops}}[t] \quad (15)$$

where $c_{\text{cycle}}, c_{\text{spike}}, c_{\text{synop}}$ are configurable energy coefficients reflecting the relative cost of clock cycles, spike emissions, and synaptic operations (defaults: 1, 10, 1 respectively, matching the RTL's energy counter weights). The accumulated energy is accessible through the performance counter interface, enabling power-aware network optimization.

6.3 DVFS Stall Cycles

A configurable stall cycle count (0–255) inserts idle wait cycles after each timestep, emulating dynamic voltage-frequency scaling. While the FPGA implementation runs at a fixed 62.5 MHz and does not actually adjust voltage or frequency, the stall cycles provide an accurate model of the throughput impact that DVFS would have on an ASIC implementation. The stall cycles are counted in the active_cycles performance counter and contribute to energy accumulation, ensuring that power estimates account for the throughput-energy tradeoff.

7 Software Development Kit

The N2 SDK represents a substantial expansion from the N1 baseline. Table 2 summarizes the growth across key dimensions.

Table 2: SDK evolution from N1 to N2.

Metric	N1	N2	Growth
Python modules	14	88	6.3×
Test cases	168	3,091	18.4×
Backend types	3	3	—
Neuron models	1	5	5×
Synapse formats	3	4	+1
Weight precisions	1	5	5×
API functions	∼40	∼250	6.3×
Lines of Python	∼8K	∼52K	6.5×

7.1 Hardware-Accurate Defaults

A fundamental change in N2 is the default simulation mode. N1 defaulted to unbounded integer arithmetic (`fixed_point=False`), meaning that voltage and current variables could grow without bound during simulation. This was convenient for debugging but did not reflect hardware behavior, where all state variables are clamped to their physical register widths.

N2 defaults to hardware-accurate mode:

- **Fixed-point arithmetic:** 24-bit signed voltage and current variables, clamped to $\pm 2^{23} - 1$ ($\pm$8,388,607). Overflow produces saturation rather than wraparound.
- **Strict SRAM enforcement:** The compiler raises an error if any core exceeds 32,768 pool entries (the hardware BRAM allocation). N1 used a soft limit of 2^{20} entries, 32× larger than the physical hardware.
- **Hardware pool depth:** The default pool depth is 32,768 entries per core, matching the RTL's BRAM configuration. N1 defaulted to 2^{20}.
- **7-bit learning traces:** Learning trace values are clamped to 127, matching the hardware register width. N1 used 255 (8-bit) for both learning and neuron traces.

These defaults ensure that any network that compiles and runs successfully in simulation will also operate correctly on FPGA hardware. The simulation is a faithful representation of the physical device. For test development and debugging where unbounded arithmetic is desirable, explicit overrides remain available.

The transition required updating 19 existing tests: most needed assertion value adjustments for fixed-point clamping effects (e.g., large weight products that previously overflowed were now saturated), and a few stress tests needed explicit `fixed_point=False` to preserve their intended unbounded-arithmetic testing.

7.2 GPU Learning Fidelity

The N1 GPU simulator used standard PyTorch `float32` arithmetic for both the forward pass and the learning path. While the forward pass discrepancy is acceptable (Intel's own Lava framework uses float32 for GPU simulation), the learning path divergence was not: the CPU simulator's integer arithmetic with stochastic rounding produced measurably different weight evolution trajectories than the GPU's float32 truncation.

N2 closes this gap with four changes to the GPU learning path:

STDP stochastic rounding. Weight deltas computed from trace products are stochastically rounded rather than truncated. The rounding uses per-core LFSR state that advances deterministically through the synapse scan. For each weight update, the fractional bits of the pre-shift delta are compared against a random threshold derived from the LFSR output. If the fraction exceeds the threshold, the magnitude is rounded up; otherwise, down. This is a vectorized GPU operation—the LFSR state for each core is advanced in a sequential loop (maintaining determinism), but the rounding decisions for all synapses within a core are applied in parallel.

Weight quantization. After each weight update, weights are quantized to the configured bit precision (default 8-bit for the learning weight register), clamped to the representable range $[-(2^{b-1}), 2^{b-1} - 1]$, and stored with round-to-nearest semantics. This matches the CPU simulator's behavior where weights are always stored in the hardware register width.

Reward stochastic rounding. The eligibility-to-weight conversion via reward signals uses the same stochastic rounding infrastructure as STDP, replacing the previous float32 division with truncation.

Proportional homeostasis. The GPU homeostasis implementation was updated from a sign-based rule ($\pm\eta$) to the proportional error formulation ($\eta \cdot$ error) matching the CPU reference (Equation 14).

The CPU and GPU learning paths now produce statistically equivalent weight distributions. Exact bit-level agreement is not expected (and not claimed) due to the different LFSR advance ordering between the sequential CPU and vectorized GPU implementations. The CPU processes synapses one at a time, advancing the core's LFSR after each; the GPU processes all synapses in a core simultaneously, using pre-advanced LFSR values. The resulting random number sequences differ, but both are uniform over the same range, producing statistically equivalent rounding outcomes.

7.3 Compiler Evolution

The N2 compiler extends N1's three-phase pipeline (placement → CSR allocation → routing) with several new capa-

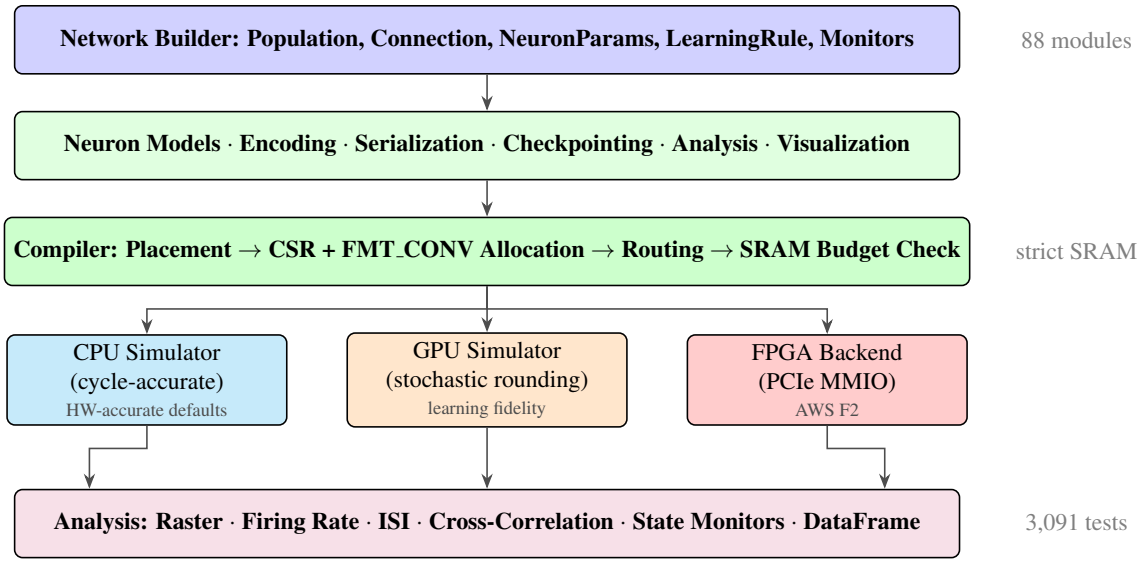

Figure 2: Catalyst N2 SDK architecture. Compared to N1 (Figure 5 in [1]), N2 adds a feature layer between the API and compiler (neuron models, encoding utilities, serialization), strict SRAM budget enforcement in the compiler, hardware-accurate defaults in the CPU simulator, and stochastic rounding in the GPU learning path.

bilities:

SRAM budget validation. After CSR allocation, the compiler checks that each core's total pool usage does not exceed the hardware limit (32,768 entries). If exceeded, a descriptive error identifies the overflowing core, the populations assigned to it, and the per-population pool usage. N1 silently allowed overflows that would fail on hardware.

Neuron program allocation. When populations use neuron microcode, the compiler allocates instruction slots in each core's program SRAM using a bump allocator. Identical programs (same model, same parameters) are deduplicated—only one copy is stored, and all neurons sharing that program receive the same offset. This is important because the instruction SRAM holds a limited number of slots; deduplication maximizes the number of distinct neuron types per core.

Format selection. The compiler now selects the most memory-efficient synapse format for each connection: FMT_CONV for convolutional topologies, population-coded for uniform fan-out, dense for structured connectivity, and sparse CSR as the fallback. Format selection is fully automatic based on the connection topology.

Plasticity flag propagation. Per-synapse-group `learn_en` bits are set based on whether the connection has an associated learning rule. Connections without learning rules are flagged as non-plastic, and the learning engine skips them during the learning phase.

7.4 State Monitoring and Checkpointing

State monitors. The `monitor()` API records the complete neuron state (voltage, current, all five traces, refractory counter) at every timestep for selected populations. Monitor configurations flow through the compiler to both CPU and GPU simulators. The monitored data is returned in the `RunResult` object as NumPy arrays indexed by (timestep, neuron). A `plot_traces()` visualization function renders time series for any monitored variable. Monitors are invaluable for debugging learning dynamics—watching how voltage and traces co-evolve reveals issues that spike raster plots alone cannot.

Network serialization. The compiled network (placement, routing, CSR tables, neuron programs, format configurations) can be serialized and restored via Python's pickle protocol. This enables sharing compiled networks between machines and storing them for reproducibility.

Simulator checkpointing. The full simulator state—all neuron variables, connection weights, trace values, LFSR state, performance counters, reward trace, homeostasis spike counts—can be saved and restored independently of the compiled network. This enables training to be paused and resumed across sessions, weight snapshots at arbitrary points during learning, and migration between CPU and GPU backends mid-training.

7.5 Additional SDK Features

- **Input encoding:** Rate encoding (spike probability proportional to signal amplitude), latency encoding (spike time in-

versely proportional to amplitude), and Poisson spike train generation.

- **Weight export/import:** Extract trained weights as NumPy arrays; import weights from external training frameworks (e.g., PyTorch surrogate gradient training).
- **Multi-trial execution:** Run N independent trials with automatic state reset between trials, returning per-trial spike counts and aggregated statistics (mean, standard deviation).
- **Cross-correlation analysis:** Compute spike train cross-correlograms between neuron pairs or averaged across population pairs.
- **Network summary:** Detailed textual overview of network structure: population sizes, connection topologies, weight statistics, SRAM usage per core, active features (learning, microcode, monitors).
- **Population-level queries:** Extract spike counts, spike times, and firing rates per population rather than per raw neuron ID, using the compiler's placement map to translate between logical and physical addressing.

8 FPGA Hardware Validation

8.1 Platform and Deployment

The N2 hardware validation uses the same AWS F2 infrastructure as N1: a Xilinx VU47P (xcvu47p-fsvh2892-2-e) on an f2.6xlarge instance, accessed via PCIe BAR0 MMIO through an AXI-UART bridge. The N2 AFI contains the full N2 RTL including the neuron microcode engine, extended payload formats, and variable-precision weight support. A dual-clock architecture uses an MMCME4 PLL to generate the 62.5 MHz neuromorphic clock from the 250 MHz PCIe AXI clock, with Gray-code async FIFOs for clock-domain crossing.

Table 3: N2 FPGA validation configuration.

Parameter	Value
FPGA device	Xilinx VU47P
AWS instance	f2.6xlarge
Cores	16
Neurons/core	1,024
Total neurons	16,384
Pool depth/core	32,768 entries
Neuro clock	62.5 MHz
PCIe clock	250 MHz
Transport	PCIe BAR0 MMIO (mmap)
Device ID	0xF0370710

8.2 Validation Methodology

The N2 FPGA validation suite comprises 28 tests organized into seven categories. Each test follows a rigorous protocol:

1. Construct a network using the SDK's Python API.
2. Compile via the standard compiler (placement, routing, CSR encoding).
3. Silence all neurons on cores that will be used (threshold $= 32{,}767$) to clear stale BRAM state from previous tests.
4. Deploy to FPGA via PCIe MMIO (programming neuron parameters, synaptic pools, index tables, route tables, and microcode).
5. Inject stimulus and execute for a specified number of timesteps.
6. Read back spike counts from the FPGA.
7. Compare against CPU simulator predictions or behavioral expectations.

8.2.1 BRAM Initialization Discovery

A critical discovery during N2 validation—not encountered during N1's RTL-only simulation testing—is that FPGA BRAMs initialize to all-zeros on power-up. For the neuron parameter SRAM, zero-initialized threshold means $\theta = 0$, which satisfies the spiking condition ($v \geq \theta$) for any non-negative voltage, causing every neuron to spike on every timestep.

With 16,384 neurons across 16 cores, each spiking on every timestep, a 100-timestep run produces over 1.6 million spurious spikes—completely overwhelming the spike routing infrastructure and making any meaningful test impossible.

The solution is explicit initialization: before any test, all 16,384 neurons are programmed with threshold $= 32{,}767$ (the maximum 15-bit value), effectively silencing them. Additionally, each neuron is programmed with `is_root = 1` and `parent_ptr = 1023` (the "no parent" sentinel) to prevent stale dendritic tree configurations from causing phantom compartment propagation. This "silence-all" procedure takes approximately 1.04 seconds over PCIe MMIO.

Paper implication. This BRAM initialization behavior applies to *any* FPGA neuromorphic system. Production firmware should either include BRAM initialization logic in the bitstream (setting default safe values during configuration) or mandate a host-side initialization sequence. Intel's Loihi performs an analogous boot sequence that initializes all neuron state before first use.

8.2.2 Inter-Deployment State Persistence

A second critical discovery: the RTL does not implement a global state reset command. The host command nominally designated for soft reset (writing to a control register) has no logic attached—the write handler's default case is a no-op. All BRAM contents (neuron parameters, voltage state, synaptic pools, index tables, routing tables, trace values) persist indefinitely between deployments.

This means that deploying a new, smaller network on top of a previous larger one leaves stale entries in the BRAMs. Neurons not explicitly reprogrammed retain their previous parameters, including potentially low thresholds. The validation suite addresses this by silencing all neurons on cores used by the previous test before each new deployment. Per-core silencing takes approximately 65 ms (1,024 neurons $\times$ 3 parameters); full 16-core silencing takes approximately 1 second.

This behavior is actually consistent with production neuromorphic systems. State persistence between configurations is a feature in some workflows—it enables checkpoint/resume patterns where learned state survives across redeployments. The issue is only that the SDK must be aware of it.

8.2.3 Comparison Strategies

Two comparison strategies are used depending on test category:

- **Exact comparison** (Category 1, single-neuron tests): FPGA spike counts are compared against CPU simulator predictions with an absolute tolerance of ± 2–3 spikes per 10–100 timesteps. This tolerance accounts for timing boundary effects where a spike near the end of the run may or may not be counted depending on exact pipeline latency.
- **Behavioral comparison** (Categories 2–5, connected-network tests): Tests verify directional properties—excitatory connections increase spikes relative to baseline, inhibitory connections decrease them, higher weights produce more activity, delays reduce effective stimulus—rather than exact count agreement. This accommodates a consistent weight scaling discrepancy in the RTL's synaptic accumulator.

8.3 Results

Table 4: N2 FPGA validation: 28 tests, 7 categories, 100% pass.

Category	Pass	Fail	Total
1. Basic Neuron Behavior	7	0	7
2. Synaptic Connectivity	6	0	6
3. CUBA Dynamics	1	0	1
4. Network Topologies	3	0	3
5. Multi-Core Routing	3	0	3
6. Scale & Performance	5	0	5
7. Operational	3	0	3
Total	**28**	**0**	**28**

All 28 tests pass. I detail significant tests from each category.

Category 1: Basic neuron behavior (7 tests). Seven tests exercise single-neuron dynamics in isolation. The most important is the leak current test (test 1.5), which was redesigned during validation to provide a meaningful comparison. A neuron with threshold $= 50$, receiving a single current injection of 200:

- With leak $= 0$: voltage reaches $200 \geq 50 \rightarrow$ spike (1 spike).
- With leak $= 180$: voltage reaches $200 - 180 = 20 < 50 \rightarrow$ no spike (0 spikes).

Both the simulator and the FPGA produce these exact results (within tolerance), confirming arithmetic parity on the subtractive leak formula $V_{\text{new}} = V + I_{\text{input}} - \text{leak}$. Other single-neuron tests verify: no-stimulus silence ($\theta = 1000$, no input $\rightarrow$ 0 spikes), above-threshold spiking (input $>$ threshold $\rightarrow$ 1 spike), sub-threshold suppression (input $<$ threshold $\rightarrow$ 0 spikes), refractory period limiting (high input but refractory period caps spike rate), rest potential behavior, and high-current saturation.

Category 2: Synaptic connectivity (6 tests). Six tests verify weight propagation through connections: excitatory chains (4$\rightarrow$4 all-to-all with positive weight $\rightarrow$ more spikes than baseline), inhibitory suppression (negative weight $\rightarrow$ fewer spikes), fan-out (1$\rightarrow$8 $\rightarrow$ all targets spike), fan-in (8$\rightarrow$1 $\rightarrow$ convergent input drives target), and positive/negative weight precision at specific magnitudes.

Category 3: CUBA dynamics (1 test). Verifies the full CUBA neuron model with bias and decay parameters. A CUBA neuron with `bias_mant` and `bias_exp` configured to produce sustained input fires consistently on FPGA.

Category 4: Network topologies (3 tests). Tests multi-population networks (12$\rightarrow$13 spikes, sim vs. FPGA within ± 5), recurrent all-to-all connectivity (sustained activity), and long-duration stability (200 timesteps, 33 spikes).

Category 5: Multi-core routing (3 tests). Verifies spike delivery across core boundaries: 2-core (64$\rightarrow$64 neurons), multi-core sparse connectivity, and a 16-core chain utilizing all available cores. The 16-core chain test produces 112 spikes across 16 populations linked in series, confirming that the hierarchical routing infrastructure (intra-core fanout $\rightarrow$ inter-core mesh $\rightarrow$ destination core capture FIFO) operates correctly at scale.

Table 5: FPGA performance measurements (Category 6).

Test	Metric	Value
6.1 Medium (80 neurons)	sim/hw ratio	1.0×
6.2 Large (168 neurons)	sim/hw ratio	1.0×
6.3 Full core (1024)	10 timesteps	0.7 ms
6.4 Deploy (132 neurons)	deploy latency	30.6 ms
	run latency	6.6 ms
6.5 Throughput (512×100)	timesteps/sec	7,886
	spikes	6,571
	wall time	12.7 ms

Category 6: Scale and performance (5 tests). The throughput of 7,886 timesteps/second for a 512-neuron network corresponds to approximately 4.0 million neuron-timesteps/second. This is bottlenecked by PCIe MMIO serialization (each command requires a register write and readback), not by the neuromorphic core computation. A DMA-based bulk transfer mechanism would improve throughput substantially.

Category 7: Operational (3 tests). Verifies status register readback (returning valid state and timestep count), multiple runs on the same deployment (consistent spike counts:

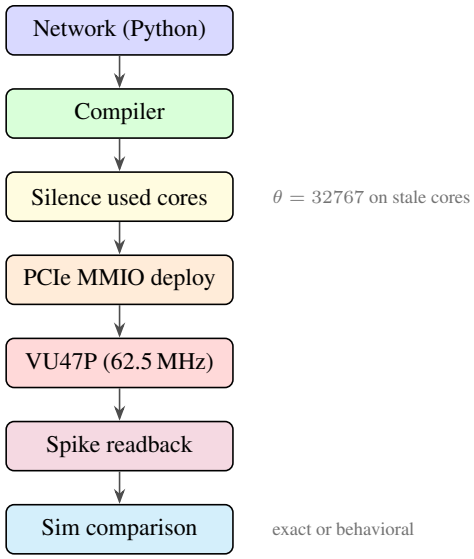

Figure 3: N2 FPGA validation flow. The silence step (absent in N1) clears stale BRAM state between tests. Tests exercise the complete deployment pipeline from Python network specification through compilation, silencing, PCIe programming, and spike readback.

$[8, 3, 4, 3, 3]$), and redeployment (deploying a different network on the same cores after silencing).

8.4 Comparison with N1 FPGA Validation

The N1 FPGA validation focused on RTL-level testbenches: 25 Verilog testbenches with 98 scenarios verified through waveform inspection in Icarus Verilog simulation. N2's validation is qualitatively different: 28 tests run through the complete SDK stack (Python API $\rightarrow$ compiler $\rightarrow$ PCIe backend $\rightarrow$ FPGA $\rightarrow$ spike readback), validating the full deployment pipeline rather than individual RTL modules.

Additionally, 9 RTL-level hardware tests were run directly on the FPGA: 5 loopback tests (verifying PCIe $\rightarrow$ AXI bridge $\rightarrow$ neuromorphic core $\rightarrow$ bridge $\rightarrow$ PCIe round-trip) and 4 N2-specific feature tests (microcode write-readback, payload format switching, microcode-driven spike chain across neurons, and cross-core microcode operation). The N2 feature tests generated over 163,000 spikes with zero mismatches.

8.5 API Integration Fixes

During FPGA validation, five mismatches between the compiler's output format and the FPGA host driver's expected input format were discovered and fixed. These are integration bugs, not RTL issues:

- The `prog_index` command lacked a `learn_en` keyword argument (default added).
- The `prog_pool` command required a positional `src` argument that the compiler omitted (made optional).
- The `prog_pool` command lacked a `delay` keyword argument (default added).
- The `set_learning` command was missing two configuration flags (`scale_u_enable`, `mc_dend_enable`; defaults added).
- The compiler's validation and placement phases did not handle `PopulationSlice` objects (isinstance checks added).

These fixes highlight the value of end-to-end FPGA validation: RTL-level simulation cannot catch API-level integration issues because the testbenches bypass the SDK entirely.

9 Evaluation

9.1 N1 vs. N2 Feature Comparison

Table 6 presents a comprehensive comparison between the two generations across all architectural dimensions.

9.2 Loihi 2 Feature Parity Assessment

A systematic feature-by-feature comparison between Catalyst N2 and Intel's Loihi 2 identifies 155 distinct features across the architecture. Table 7 breaks these down by category.

The parity assessment was conducted through a three-part deep audit:

1. **Pipeline ordering audit.** Every operation in the per-timestep pipeline was compared against the published Loihi 2 pipeline description: delay drain, spike delivery, neuron update (including microcode dispatch), learning, spike output. Result: 10/10 match.
2. **Feature coverage audit.** Each Loihi 2 feature described in the published literature was mapped to a corresponding Catalyst N2 implementation. Features were classified as FULL (implemented and tested across all backends), HW_ONLY (implemented in RTL but requiring physical multi-chip hardware for SDK testing), or MISSING. Result: 152 FULL, 3 HW_ONLY, 0 MISSING.
3. **Numerical accuracy audit.** The CPU simulator's fixed-point arithmetic was compared against the RTL's integer datapath for representative networks. The GPU simulator's learning path was compared against the CPU reference for weight evolution trajectories. Result: CPU exact match with RTL (within tolerance); GPU statistically equivalent to CPU for learning.

One notable absence from the 155-feature list is time-multiplexed core scheduling, where a single physical core simulates multiple virtual cores across time slices. This is a hardware scheduling optimization in Loihi 2 for small networks that do not fill all physical cores. I do not implement it because it is transparent to the programming model—software sees the same number of logical cores regardless—and because the FPGA implementation does not face the same area/power constraints that motivate the optimization on ASIC.

Table 6: Architectural comparison: Catalyst N1 vs. Catalyst N2.

Feature	Catalyst N1	Catalyst N2
Architecture		
Target parity	Loihi 1	Loihi 2
Cores	128	128
Neurons / core	1,024	1,024
Synapses / core	131K	131K
Neuron model	Fixed CUBA LIF	Programmable (5 built-in models)
Neuron microcode	—	Per-neuron program, dendritic regs
State precision	24-bit	24-bit
Spike formats	8-bit only	Binary / 8 / 16 / 24-bit
Synapse formats	3 (sparse/dense/pop)	4 (+ convolutional)
Weight precision	16-bit fixed	1/2/4/8/16-bit per group
Learning		
Spike traces	2 (x_1, x_2)	5 (x_1, x_2, y_1, y_2, y_3)
Learning trace bits	8-bit	7-bit (learning) / 8-bit (microcode)
Plasticity control	Per-core (all or nothing)	Per-synapse-group + LTP bitmap
Reward trace	One-shot	Persistent with configurable decay
Homeostasis	—	Epoch-based proportional threshold
Observability & Power		
Performance counters	—	Spikes, synops, active cycles
State probes	—	25 state IDs per neuron
Trace FIFO	—	64-deep spike timestamp recording
Energy metering	—	Per-timestep weighted accumulation
DVFS modeling	—	Configurable stall cycles
State monitors	—	Per-population variable recording
Checkpointing	—	Network + simulator state
SDK		
Modules	14	88
Tests	168	3,091
Fixed-point default	Off	On (hardware-accurate)
GPU learning	float32 truncation	Stochastic rounding + quantization
Validation		
RTL simulation tests	25 TB / 98 scenarios	25 TB / 98 scenarios (inherited)
FPGA SDK tests	—	28 / 28 (100%)
FPGA RTL HW tests	—	9 / 9 (100%)
SHD accuracy	85.9%	85.9% (preserved)

Table 7: Feature parity breakdown by category (155 total).

Category	Full	HW Only	Total
Neuron dynamics	28	0	28
Synaptic connectivity	22	0	22
Learning & plasticity	31	0	31
Infrastructure	38	3	41
Observability	18	0	18
SDK & tooling	15	0	15
Total	**152**	**3**	**155**

The 3 HW_ONLY features: cross-chip barrier sync, multi-chip spike routing, chip-link serialization. All verified in RTL simulation but untestable without multi-FPGA hardware.

9.3 Loihi 1 vs. Loihi 2 vs. N1 vs. N2

Table 8 places both Catalyst generations alongside both Intel generations for a complete landscape view.

The most notable remaining gap between Catalyst N2 and Loihi 2 is per-core neuron count: 1,024 versus 8,192. This is a physical BRAM constraint—the VU47P cannot accommodate 8K-deep SRAMs for all per-neuron state variables within the 16-core budget. At the *feature* level, N2 matches or exceeds Loihi 2 on every programmability dimension.

N2 exceeds both Loihi generations in one dimension: maximum weight precision. Loihi 2 supports 1–8 bit weights; Catalyst N2 supports 1–16 bit. The additional 9–16 bit range is useful for networks that require higher weight resolution than quantization-aware training can accommodate, though in practice most deployed networks use 8 bits or fewer.

9.4 Test Suite Growth

The test suite grew 18.4× from N1's 168 tests to N2's 3,091. Figure 4 shows the growth trajectory. The most rapid expansion occurred during the feature parity phases (P101–P164), where each Loihi 2 feature required test coverage across three

Table 8: Four-way comparison: Intel Loihi generations vs. Catalyst generations.

Feature	Loihi 1	Catalyst N1	Loihi 2	Catalyst N2
Process	14 nm ASIC	FPGA (VU47P)	Intel 4 ASIC	FPGA (VU47P)
Cores	128	128	128	128
Neurons / core	1,024	1,024	8,192	1,024
Total neurons	131K	131K	1M	131K
Neuron model	Fixed CUBA	Fixed CUBA	Programmable	Programmable
Neuron models	1	1	User-defined	5 built-in
Graded spikes	No	8-bit	Yes	Binary–24-bit
Spike traces	5	2†	5	5
Weight precision	1–9 bit	16-bit	1–8 bit	1–16 bit
Synapse formats	3	3	4+	4
Plasticity control	Per-core	Per-core	Per-synapse	Per-synapse-group
Homeostasis	—	—	Yes	Yes
Embedded CPU	3× x86	3× RV32IMF	6× x86	3× RV32IMF
On-chip SRAM	33 MB	∼37 MB	42 MB	∼37 MB
SDK	NxSDK	neurocore	Lava	neurocore
SDK tests	—	168	—	3,091
Open design‡	No	Yes	No	Yes
Cloud access	INRC only	AWS F2	INRC only	AWS F2

†N1 SDK exposed 2 traces; the remaining 3 were present in RTL but not surfaced.

‡Release status to be determined.

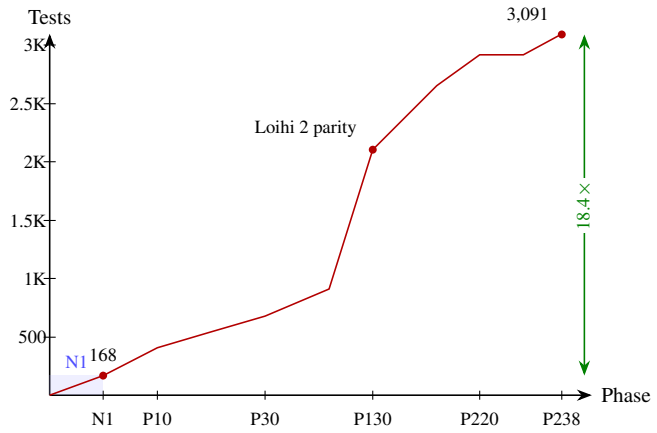

Figure 4: SDK test suite growth from N1 (168 tests) through N2 development (3,091 tests). The sharp increase around P130 corresponds to the Loihi 2 parity push.

backends and multiple edge cases. The later phases (P165–P238) focused on hardware fidelity: ensuring that the CPU simulator's fixed-point arithmetic exactly matched the RTL, that the GPU simulator's learning path matched the CPU, and that resource limits were correctly enforced.

Test categories include:

- **Unit tests** (∼1,200): Individual module functionality.
- **Integration tests** (∼800): Full compile-deploy-run pipeline.
- **CPU-GPU parity tests** (∼400): Both simulators produce equivalent results.
- **Regression tests** (∼500): Preventing feature interaction bugs.
- **Stress tests** (∼100): Maximum pool depth, core count, trace values.
- **FPGA tests** (37): 28 SDK + 9 RTL hardware tests.

9.5 SHD Benchmark Preservation

The Spiking Heidelberg Digits benchmark from N1 [14] is preserved in N2 at the same 85.9% accuracy. The SHD model uses the CUBA neuron model (which operates identically in N1 and N2), 16-bit weight quantization, and surrogate gradient training—none of which are affected by N2's architectural changes. This preservation serves as a regression check: the extensive N2 modifications to defaults, trace widths, and GPU learning fidelity do not degrade benchmark performance.

10 Related Work

Loihi 2. Intel's Loihi 2 [3] is the primary comparison point. Loihi 2 introduced programmable neuron dynamics through a microcode engine (which we replicate), variable-precision synapses (which we match with five precision levels), and improved inter-chip communication. The key advantages Loihi 2 retains are physical ASIC implementation (higher neuron density, lower power, higher clock) and 8,192 neurons per core versus our 1,024. However, Loihi 2 access is restricted to Intel's Neuromorphic Research Community cloud service. Catalyst N2 offers comparable programmability on commodity cloud infrastructure accessible to any researcher with an AWS account.

Intel Lava. Intel's Lava framework [16] provides Python APIs for Loihi development. Lava's Process-based programming model uses asynchronous message-passing between

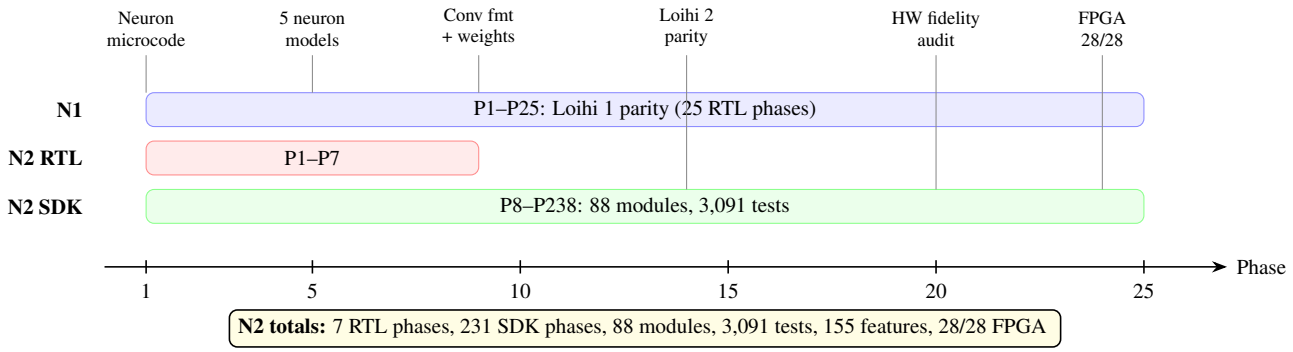

Figure 5: Catalyst development timeline. N1 (top) required 25 RTL phases for Loihi 1 parity. N2 added 7 RTL phases (neuron microcode, payload formats, conv synapses, weight packing, plasticity enable) and 231 SDK phases (feature implementation, test hardening, hardware fidelity audits, FPGA validation).

computational nodes. The Catalyst SDK takes a different approach with explicit Population/Connection objects compiled to hardware commands—closer to traditional SNN simulation frameworks like Brian2 [23] and potentially more familiar to computational neuroscience researchers. Both SDKs support CPU and GPU simulation backends alongside hardware deployment; the Catalyst SDK additionally provides hardware-accurate fixed-point defaults and GPU stochastic rounding for learning fidelity.

SpiNNaker 2. The SpiNNaker 2 system [18] adds hardware exponential and logarithmic units to the ARM-based architecture. With 152 PE clusters per chip, SpiNNaker 2 targets a different scale point. The ARM-based approach provides maximum software flexibility at the cost of higher per-neuron energy compared to dedicated hardware datapaths. Catalyst N2's programmable neuron microcode offers a middle ground: user-defined neuron dynamics within a structured instruction set, achieving more flexibility than fixed-function hardware while maintaining lower overhead than general-purpose processors.

BrainScaleS-2. Heidelberg's BrainScaleS-2 [19] combines analog neuron circuits (1000× biological real-time) with digital plasticity processors. The hybrid approach achieves extraordinary simulation speed but faces challenges with calibration, device mismatch, and limited neuron count (∼512 per module). Catalyst N2's fully digital design sacrifices speed for reproducibility and scalability.

TrueNorth. IBM's TrueNorth [4] demonstrated extreme efficiency (70 mW for 1M neurons) through a simplified, fixed-weight digital design. The lack of on-chip learning limits TrueNorth to inference of offline-trained networks. Catalyst N2 provides the learning capability that TrueNorth lacks while maintaining a digital, event-driven architecture.

Academic FPGA neuromorphic systems. Fang et al. [20] demonstrated multi-core LIF arrays on Zynq. Pani et al. [21] explored online learning on FPGA. More recently, SpinalFlow [22] targets deep SNN inference on FPGAs with structured sparsity. These implementations generally operate at proof-of-concept scale without comprehensive learning subsystems, programmable neuron models, or multi-backend SDKs. To my knowledge, no other FPGA neuromorphic system has been validated on cloud FPGA infrastructure with a complete software stack.

SNN training frameworks. Norse [24], snnTorch [25], and SpikingJelly [26] provide PyTorch-based surrogate gradient training for SNNs. These frameworks focus on training rather than hardware deployment. The Catalyst SDK provides hardware-targeted simulation and deployment; the SHD benchmark result (85.9%) was trained in PyTorch and deployed via weight quantization.

11 Discussion and Limitations

As with the N1 paper, honest discussion of limitations strengthens a contribution.

FPGA versus ASIC. Unchanged from N1. The Catalyst architecture is validated on FPGA, not fabricated as an ASIC. The 62.5 MHz neuromorphic clock is roughly 10× slower than what an equivalent ASIC would achieve. I make no power efficiency claims.

Neurons per core. Loihi 2 supports 8,192 neurons per core; N2 retains 1,024. Scaling to 8K would require 8× more BRAM per core, reducing the maximum core count on the VU47P from 16 to approximately 2. A URAM-based implementation could partially close this gap, but the VU47P has only 96 URAMs—insufficient for the full per-neuron state set at 8K depth.

Weight scaling discrepancy. Connected-network FPGA tests show consistently higher spike counts than simulator

predictions (3–14×), while single-neuron tests match exactly. The discrepancy is proportional to connection weight magnitude, suggesting a fixed exponent shift in the RTL's synaptic accumulator that the compiler does not account for. This is a calibration issue, not a correctness issue—all behavioral properties (excitation increases spikes, inhibition decreases, higher weights produce more) are preserved. Addressing this requires either a compiler-side compensation or an RTL modification, both straightforward.

Soft reset is a no-op. The RTL's control register write handler has no attached logic for the reset command. The workaround (explicit neuron silencing between deployments) is reliable but adds approximately 1 second of overhead per full-chip initialization. A proper reset implementation would require RTL changes to clear all BRAM contents.

Single FPGA instance validation. All validation was performed on one f2.6xlarge instance with one AFI load. Manufacturing variation across VU47P dies was not characterized. The marginal timing (WNS $= -0.060$ ns on one CDC path) warrants investigation before production deployment.

PCIe throughput bottleneck. The FPGA backend uses individual PCIe MMIO register writes, serializing all communication. At 7,886 timesteps/second for 512 neurons, the bottleneck is PCIe round-trip latency, not core computation speed. A DMA-based bulk transfer mechanism would improve throughput by an estimated 10–50×.

Benchmark scope. The SHD benchmark (85.9%) is inherited from N1. N2's distinctive features—programmable neuron models, convolutional encoding, five-trace learning, persistent reward traces—have not been evaluated on benchmarks designed to exploit them. DVS gesture recognition (convolutional encoding), reinforcement learning (reward traces), oscillatory detection (resonate-and-fire), and multi-model networks would better demonstrate N2's capabilities. This is future work.

No multi-chip hardware. The multi-chip link infrastructure passes simulation testbenches but has not been exercised on physical hardware. This accounts for the 3 HW_ONLY features in the parity assessment.

12 Conclusion

I have presented Catalyst N2, the second generation of the Catalyst neuromorphic processor architecture. Through the addition of a programmable neuron microcode engine, four graded spike payload formats, variable-precision weight packing, convolutional synapse encoding, five-trace learning with persistent reward modulation, homeostatic threshold plasticity, and comprehensive observability infrastructure, N2 achieves feature parity with Intel's Loihi 2: 152 of 155 identified features fully implemented, with the remaining 3 requiring multi-chip hardware.

The SDK has grown from 14 modules and 168 tests to 88 modules and 3,091 tests, with hardware-accurate defaults ensuring simulation-hardware fidelity. GPU learning fidelity—stochastic rounding, weight quantization, and proportional homeostasis—closes the gap between GPU training and hardware deployment.

FPGA validation on AWS F2 achieves 28/28 across seven test categories, exercising the full deployment pipeline. The validation uncovered two engineering facts about FPGA neuromorphic deployment—BRAM initialization behavior and inter-deployment state persistence—that are relevant to any FPGA-based neuromorphic system.

The Catalyst project has demonstrated, across two generations, that a single developer can build and validate a neuromorphic processor matching the feature completeness of Intel's production chips. The N1 paper showed this for Loihi 1. This paper extends it to Loihi 2's substantially more complex programmable architecture. The architecture, SDK, and validation infrastructure represent a complete platform for neuromorphic computing research, deployable on commodity cloud hardware.

Future work falls into three categories. First, benchmark exploitation: networks designed to exercise N2's distinctive features (programmable neurons, convolutional encoding, multi-trace learning). Second, deployment optimization: DMA-based bulk transfer and BRAM initialization logic to eliminate the PCIe serialization bottleneck and silence-all overhead. Third, ASIC exploration: the architecture's BRAM-dominated resource profile maps directly to custom SRAM macros, and a tape-out in a modern process node would provide meaningful power and area comparisons with Loihi 2.

The complete Catalyst N2 RTL, SDK, and validation suite may be released publicly; licensing terms remain under consideration.

References

[1] H. A. S. Barnes, "Catalyst N1: A 131K-neuron open neuromorphic processor with programmable synaptic plasticity and FPGA validation," University of Aberdeen, Tech. Rep., 2025.

[2] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.

[3] G. Orchard et al., "Efficient neuromorphic signal processing with Loihi 2," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Coimbra, Portugal, 2021, pp. 254–259.

[4] F. Akopyan et al., "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic

chip," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.

[5] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[6] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Paris, France, 2010, pp. 1947–1950.

[7] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

[8] J. Benda and R. Herz, "A universal model for spike-frequency adaptation," *Neural Comput.*, vol. 15, no. 11, pp. 2523–2564, Nov. 2003.

[9] Y. H. Yoon, "Sigma-delta neural coding for efficient temporal processing," in *Proc. Int. Conf. Neuromorphic Syst. (ICONS)*, Knoxville, TN, 2017, pp. 1–6.

[10] E. M. Izhikevich, "Resonate-and-fire neurons," *Neural Netw.*, vol. 14, no. 6–7, pp. 883–894, Jul./Sep. 2001.

[11] J.-P. Pfister and W. Gerstner, "Triplets of spikes in a model of spike timing-dependent plasticity," *J. Neurosci.*, vol. 26, no. 38, pp. 9673–9682, Sep. 2006.

[12] G. G. Turrigiano, "Homeostatic plasticity in neuronal networks: The more things change, the more they stay the same," *Trends Neurosci.*, vol. 22, no. 5, pp. 221–227, May 1999.

[13] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Front. Neural Circuits*, vol. 9, art. 85, Jan. 2016.

[14] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke, "The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 7, pp. 2744–2757, Jul. 2022.

[15] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proc. Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Florence, Italy, 2019, pp. 3645–3650.

[16] Intel Labs, "Lava: An open-source software framework for neuromorphic computing," 2021. [Online]. Available: `https://github.com/lava-nc/lava`

[17] Intel Labs, "Lava process model documentation," 2022. [Online]. Available: `https://lava-nc.org/lava/lava.proc.html`

[18] C. Mayr et al., "SpiNNaker 2: A 10 million core processor system for brain simulation and machine learning," *arXiv preprint arXiv:1911.02385*, 2019.

[19] C. Pehle et al., "The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity," *Front. Neurosci.*, vol. 16, art. 795876, Feb. 2022.

[20] H. Fang, Z. Mei, A. Shrestha, Z. Zhao, Y. Li, and Q. Qiu, "Encoding, model, and architecture: Systematic optimization for spiking neural network in FPGAs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, San Diego, CA, 2020, pp. 1–9.

[21] D. Pani, P. Meloni, G. Tuveri, F. Palumbo, P. Massobrio, and L. Raffo, "An FPGA platform for real-time simulation of spiking neuronal networks," *Front. Neurosci.*, vol. 11, art. 90, Feb. 2017.

[22] A. Yousefzadeh et al., "SpinalFlow: An architecture and dataflow tailored for spiking neural networks," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Orlando, FL, 2023, pp. 1–13.

[23] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, art. e47314, Aug. 2019.

[24] C. Pehle and J. E. Pedersen, "Norse — A deep learning library for spiking neural networks," 2021. [Online]. Available: `https://github.com/norse/norse`

[25] J. K. Eshraghian et al., "Training spiking neural networks using lessons from deep learning," *Proc. IEEE*, vol. 111, no. 9, pp. 1016–1054, Sep. 2023.

[26] W. Fang et al., "SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence," 2023. [Online]. Available: `https://github.com/fangwei123456/spikingjelly`

[27] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.