Weight Discretization for Quotient Model Abstraction in Spiking Neural Network Verification

Research Note — January 2026
(Fourth Revision)

1. Introduction

This note extends the filtration-based quotient model abstraction [1] to preserve *synaptic weight information* during formal verification. While the original quotient model abstracts membrane potentials into equivalence classes, it treats all synapses uniformly, losing essential structural information.

We address this by introducing a *weight discretization scheme* that:

1. Maps continuous weights to a finite discrete range
2. Preserves the relative contribution of synapses to membrane potential
3. Ensures threshold feasibility is maintained
4. Retains weight visibility in the generated PRISM model

What's New in Revision 4:

- Threshold-dependent leak factor formulation (§4.2)
- Soundness theorem: "If we don't receive input, we shouldn't spike" (§3.4)
- Biological property preservation analysis (§5.3)
- Variable dependency graph (§6)
- Corrected code examples and notation

2. Preliminaries

This section introduces key concepts and notation used throughout the proofs.

2.1. Notation Summary

Symbol	Meaning
W	Discretization parameter: the number of positive weight levels
$w_{\max}$	Maximum absolute weight in the original model (typically 100)
δ_W	Weight discretization function: maps original weights to discrete values
T	Original firing threshold of a neuron
T_d	Discretized firing threshold
m	Fan-in: the number of incoming synapses to a neuron
S	Weighted sum of spiking inputs in the original model
S_d	Weighted sum of spiking inputs in the discretized model
γ	Class width: potential range represented by each equivalence class
λ_d	Discretized leak factor (always ≤ 0), *dependent on* T_d
k	Number of threshold levels (equivalence classes)
r	Retention rate: fraction of potential retained per step ($r \in [0, 1]$)

Table 1: Summary of notation used in this document

2.2. The Rounding Property

The standard mathematical rounding function $\text{round}(x)$ rounds x to the nearest integer. A crucial property we use throughout is the *rounding bound*:

> **Definition.** For any real number x, the rounding function satisfies:
>
> $$x - \frac{1}{2} \leq \text{round}(x) \leq x + \frac{1}{2}$$
>
> Equivalently: $\text{round}(x) \geq x - \frac{1}{2}$ (lower bound) and $\text{round}(x) \leq x + \frac{1}{2}$ (upper bound).

> 💡 **Intuition:** Rounding can shift a value by at most $\frac{1}{2}$ in either direction. When we sum m rounded values, the total error is bounded by $\frac{m}{2}$ — this is the *cumulative rounding error*.

2.3. The Clamp Function

> **Definition.** The *clamp function* restricts a value to a specified range $[a, b]$ where $a < b$:
>
> $$\text{clamp}(x, a, b) = \max(a, \min(b, x)) = \begin{cases} a \text{ if } x < a \\ x \text{ if } a \leq x \leq b \\ b \text{ if } x > b \end{cases}$$

💡 **Intuition:** Clamping prevents values from exceeding safe bounds. In our model, we clamp class indices to $[0, k]$ and class deltas to $[-k, k]$ to avoid invalid states or runaway accumulation.

2.4. Fan-in

Definition. The *fan-in* of a neuron, denoted m, is the number of incoming synaptic connections. A neuron with fan-in m receives input from m presynaptic neurons.

Remark. Fan-in is critical because the cumulative rounding error scales linearly with m. High fan-in neurons require finer discretization to maintain accuracy.

3. Weight Discretization

3.1. Formal Definition

Definition. Given a weight range $[w_{\min}, w_{\max}] = [-100, 100]$ and a discretization parameter $W \in \mathbb{N}^+$, the *weight discretization function* $\delta_W : \mathbb{R} \to \mathbb{Z}$ is defined as:

$$\delta_W(w) = \operatorname{round}\left(w \cdot \frac{W}{w_{\max}}\right)$$

The discretized weight range is $[-W, W] \subset \mathbb{Z}$.

💡 **Intuition:** We scale the original weight by $\frac{W}{w_{\max}}$ to map the range $[-w_{\max}, w_{\max}]$ to $[-W, W]$, then round to get an integer. This preserves the *relative magnitude* of weights while reducing the number of distinct values.

Example. For $W = 3$ (giving 7 discrete levels: $-3, -2, -1, 0, 1, 2, 3$):
- $\delta_3(100) = \operatorname{round}(100 \cdot \frac{3}{100}) = \operatorname{round}(3) = 3$ (strong excitatory)
- $\delta_3(67) = \operatorname{round}(67 \cdot \frac{3}{100}) = \operatorname{round}(2.01) = 2$ (medium excitatory)
- $\delta_3(33) = \operatorname{round}(33 \cdot \frac{3}{100}) = \operatorname{round}(0.99) = 1$ (weak excitatory)
- $\delta_3(0) = \operatorname{round}(0) = 0$ (negligible)
- $\delta_3(-50) = \operatorname{round}(-50 \cdot \frac{3}{100}) = \operatorname{round}(-1.5) = -2$ (medium inhibitory)
- $\delta_3(-100) = \operatorname{round}(-3) = -3$ (strong inhibitory)

3.2. Threshold Calibration

The key challenge is ensuring that threshold reachability is preserved after discretization. We must calibrate the *discretized threshold* T_d to be consistent with the original threshold T.

Definition. The *discretized threshold* for a neuron with original threshold T and weight discretization parameter W is:

$$T_d = \operatorname{ceil}\left(T \cdot \frac{W}{w_{\max}}\right)$$

💡 **Intuition:** We use ceiling (round up) rather than standard rounding to ensure the discretized threshold is *at least as hard* to reach as the original. This prevents false positives: if a discretized neuron fires, we can be confident the original would too.

3.3. Threshold Preservation Theorem (Completeness)

Theorem (Threshold Preservation — Completeness)**.** Let $\mathcal{N}$ be a neuron with incoming weights $\{w_1, ..., w_m\}$ and threshold T. Let $\mathcal{N}'$ be the discretized version with weights $\{\delta_W(w_1), ..., \delta_W(w_m)\}$ and threshold T_d.

If $\mathcal{N}$ can fire in a single step (i.e., $\exists$ input pattern $\boldsymbol{y} \in \{0,1\}^m$ such that $\sum_{i=1}^m w_i \cdot y_i \geq T$), then $\mathcal{N}'$ can also fire in a single step.

Proof. We prove this in six steps.

Step 1 (Setup)**:** Let $\boldsymbol{y}^*$ be an input pattern that causes the original neuron $\mathcal{N}$ to fire. Define:

- $S = \sum_{i=1}^m w_i \cdot y_i^*$ — the weighted sum of spiking inputs in the original model
- By assumption, $S \geq T$ (the neuron fires)

Step 2 (Discretized contribution)**:** The corresponding weighted sum in the discretized model is:

$$S_d = \sum_{i=1}^{m} \delta_W(w_i) \cdot y_i^*$$

We need to show $S_d \geq T_d$ to prove the discretized neuron also fires.

Step 3 (Apply the rounding property)**:** By the rounding property (see §2.2), for each weight:

$$\delta_W(w_i) = \mathrm{round}\left(w_i \cdot \frac{W}{w_{\max}}\right) \geq \frac{w_i \cdot W}{w_{\max}} - \frac{1}{2}$$

Since $y_i^* \in \{0,1\}$, when $y_i^* = 1$ this bound applies, and when $y_i^* = 0$ the term is zero. Summing over all inputs:

$$S_d \geq \sum_{i=1}^{m} \left(\frac{w_i \cdot W}{w_{\max}} - \frac{1}{2}\right) \cdot y_i^* = \left(\sum_{i=1}^{m} w_i \cdot y_i^*\right) \cdot \frac{W}{w_{\max}} - \frac{\sum_{i=1}^m y_i^*}{2}$$

Step 4 (Bound the cumulative error)**:** Let $m^* = \sum_{i=1}^m y_i^*$ be the number of active inputs. Then:

$$S_d \geq S \cdot \frac{W}{w_{\max}} - \frac{m^*}{2}$$

Since $m^* \leq m$ (at most m inputs can be active):

$$S_d \geq S \cdot \frac{W}{w_{\max}} - \frac{m}{2}$$

The term $\frac{m}{2}$ is the *cumulative rounding error* — the worst-case total error from rounding m weights.

Step 5 (Derive the firing condition)**:** For the discretized neuron to fire, we need $S_d \geq T_d$. By the ceiling property:

$$T_d = \text{ceil}\left(T \cdot \frac{W}{w_{\max}}\right) \leq T \cdot \frac{W}{w_{\max}} + 1$$

Combining with our bound on S_d:

$$S_d \geq S \cdot \frac{W}{w_{\max}} - \frac{m}{2}$$

Since $S \geq T$:

$$S_d \geq T \cdot \frac{W}{w_{\max}} - \frac{m}{2}$$

A sufficient condition for firing is:

$$T \cdot \frac{W}{w_{\max}} - \frac{m}{2} \geq T \cdot \frac{W}{w_{\max}} + 1$$

This simplifies to: $(S - T) \cdot \frac{W}{w_{\max}} \geq \frac{m}{2} + 1$

Step 6 (Boundary case and parameter choice)**:** For the critical boundary case where $S = T$ exactly, we need:

$$0 \geq \frac{m}{2} + 1$$

This is never satisfied, so at the exact boundary, firing is not guaranteed. However, if S exceeds T by a small margin, or we choose W large enough, firing is preserved.

Specifically, for $W \geq w_{\max} \cdot \frac{\frac{m}{2}+1}{T}$, the discretized neuron fires whenever the original does.

Practical example: With $W = 3$, $w_{\max} = 100$, $m \leq 10$, and $T = 100$:

$$W \geq 100 \cdot \frac{5+1}{100} = 6$$

Thus $W = 7$ (discrete range $[-3, 3]$) is sufficient for most practical networks.

$\square$

Remark. Critical Constraint: The proof shows that weight discretization introduces a cumulative error of $-\frac{m}{2}$. If the fan-in m is large relative to W (specifically if $m > 2W$), the rounding noise may exceed the signal of the smallest synaptic weight.

For high-fanin neurons, we strictly recommend:
1. Using a finer discretization ($W \geq \frac{m}{2}$)
2. Applying a threshold correction factor: $T_{d'} = T_d - \text{floor}\left(\frac{m}{2W}\right)$ (Note: This may increase false positive firings).

3.4. Soundness Theorem (Safety)

We now prove the converse property: if the original neuron should *not* fire, the discretized neuron should also *not* fire.

> **Theorem** (Asymptotic Silence — Soundness)**.** Let $\mathcal{N}'$ be a discretized neuron with current potential $P_t < T_d$ and leak factor $\lambda_d \leq -1$. If the input sequence is empty (i.e., total weighted input $S_d = 0$) for all steps $t' \geq t$, then $\mathcal{N}'$ will never fire.

Proof. We prove this by showing the potential strictly decreases to zero.

Step 1 (Dynamics without input)**:** In the absence of input ($S_d = 0$), the discretized update rule simplifies to:

$$P_{t+1} = \max(0, P_t + \lambda_d)$$

This is applying the potential update with only leak (no external contribution).

Step 2 (Strict decay guarantee)**:** Since $\lambda_d \leq -1$, if $P_t > 0$, then:

$$P_{t+1} \leq P_t + \lambda_d \leq P_t - 1$$

The potential strictly decreases by at least 1 unit per step.

Step 3 (Convergence to zero)**:** Starting from any $P_t < T_d$, the sequence $\{P_t\}$ is strictly decreasing until it reaches the absorbing state $P = 0$ (due to the $\max(0, ...)$ clamp).

Step 4 (Firing impossibility)**:** Since $P_t < T_d$ initially and the sequence is non-increasing, the firing condition $P \geq T_d$ is never met. The neuron remains silent indefinitely.

□

> 💡 **Intuition:** This theorem provides a *safety guarantee*: the discretization does not introduce spurious spikes. If there's no input to drive the potential up, leak ensures it decays to rest. This prevents rounding errors from accidentally triggering false firings.

4. Class Transition with Weighted Contributions

4.1. Contribution-Based Class Evolution

In the original quotient model, class evolution used a binary rule which loses weight information. We replace it with *weighted contribution-based* class evolution.

> **Definition.** The *weighted contribution* for neuron n with incoming discretized weights $\{w_1^d, ..., w_m^d\}$ is:
>
> $$C_n = \sum_{i=1}^{m} w_i^d \cdot y_i$$
>
> where $y_i \in \{0, 1\}$ is the spike output of presynaptic neuron i.

Definition. The *class delta function* $\Delta : \mathbb{Z} \to \mathbb{Z}$ maps contribution to class change:

$$\Delta(C) = \text{clamp}\left(\text{round}\left(\frac{C}{\gamma}\right), -k, k\right)$$

where:
- γ is the *class width* (typically $\gamma = \frac{T_d}{k}$) — the potential range each class represents
- k is the number of threshold levels
- The clamp ensures the class change stays within valid bounds

💡 **Intuition:** The class delta converts a weighted sum of inputs into a class change. We divide by γ to express the contribution in "class units", round to get an integer, and clamp to prevent impossibly large jumps.

4.2. Threshold-Dependent Leak Factor

The quotient model must also account for membrane potential decay (leak). We propose a corrected *threshold-dependent* leak formula.

Definition. The *discretized leak factor* λ_d is:

$$\lambda_d = -\max(1, \lfloor (1 - r) \cdot T_d \rfloor)$$

where:
- $r \in [0, 1]$ is the original retention rate (fraction of potential retained per step)
- T_d is the discretized threshold
- The $\max(1, \ldots)$ ensures a minimum decay of 1 unit per step (preventing infinite energy trapping)
- The negative sign ensures leak *decreases* potential

💡 **Intuition:** By linking the leak factor to T_d rather than the number of classes k, we ensure the decay scales correctly with the calibrated threshold. A high threshold means larger potential values, so the decay must be proportionally larger to maintain realistic behavior.

Example. For $r = 0.9$ (90% retention, i.e., 10% decay) and $T_d = 10$:

$$\lambda_d = -\max(1, \lfloor (1 - 0.9) \cdot 10 \rfloor) = -\max(1, \lfloor 1.0 \rfloor) = -1$$

Result: Potential decreases by 1 per step without input.

For $r = 0.5$ (50% decay) and $T_d = 10$:

$$\lambda_d = -\max(1, \lfloor (1 - 0.5) \cdot 10 \rfloor) = -\max(1, \lfloor 5.0 \rfloor) = -5$$

Result: Potential decreases by 5 per step without input — aggressive decay.

For $r = 0.95$ (95% retention) and $T_d = 10$:

$$\lambda_d = -\max(1, \lfloor (1 - 0.95) \cdot 10 \rfloor) = -\max(1, \lfloor 0.5 \rfloor) = -\max(1, 0) = -1$$

Result: Even with very high retention, we enforce minimum decay of 1.

Remark. Key Insight: The previous formulation $\lambda_d = -\text{round}((1 - r) \cdot k)$ was problematic because:
1. It depended on k (number of classes) rather than T_d (actual potential scale)
2. Could result in $\lambda_d = 0$ for high retention, violating the Soundness theorem

The new formulation ensures $|\lambda_d| \geq 1$ always, guaranteeing the Asymptotic Silence property.

The class evolution rule becomes:

$$c_n' = \text{clamp}(c_n + \Delta(C_n) + \lambda_d, 0, k)$$

5. Threshold Feasibility Analysis

5.1. Definition

Definition. A neuron configuration is *threshold-feasible* if there exists at least one input pattern that can cause the neuron to reach the firing threshold within a finite number of steps.

💡 **Intuition:** Feasibility asks: "Can this neuron ever fire?" If the leak is too strong relative to the available excitation, the potential can never build up enough to reach threshold — the neuron is permanently silent.

Theorem (Feasibility Criterion)**.** A neuron with discretized weights $\{w_1^d, ..., w_m^d\}$ and threshold T_d is threshold-feasible if and only if:

$$\sum_{w_i^d > 0} w_i^d > |\lambda_d|$$

And specifically, for reliable firing:

$$\sum_{w_i^d > 0} w_i^d \geq \frac{T_d}{1 + |\lambda_d|}$$

Proof.

Step 1 (Define maximum excitation)**:** Let $E = \sum_{w_i^d > 0} w_i^d$ be the maximum possible excitatory contribution per step (achieved when all excitatory presynaptic neurons fire simultaneously).

Step 2 (Single-step case)**:** If $E \geq T_d$, the neuron can fire in a single step by receiving all excitatory inputs simultaneously. Feasibility is trivially satisfied.

Step 3 (Accumulation case)**:** If $E < T_d$, the neuron must accumulate potential over multiple steps. With leak factor $\lambda_d \leq 0$, each step adds a net contribution of:

$$\text{Net gain} = E + \lambda_d$$

(Note: λ_d is negative, so this is $E - |\lambda_d|$)

For accumulation to be possible, we require:

$$E + \lambda_d > 0 \quad \Rightarrow \quad E > |\lambda_d|$$

If this holds, the minimum steps to reach threshold is:

$$n = \mathrm{ceil}\left(\frac{T_d}{E + \lambda_d}\right) = \mathrm{ceil}\left(\frac{T_d}{E - |\lambda_d|}\right)$$

Step 4 (Necessity: why excitation ≤ leak implies impossibility)**:** If $E \leq |\lambda_d|$, then the net gain per step is $E - |\lambda_d| \leq 0$. The potential cannot grow — any excitation is immediately cancelled (or overpowered) by leak. The neuron can never reach threshold.

□

5.2. Implementation

The feasibility check should be performed at PRISM generation time:

```
fn check_feasibility(
    weights: &[i32],  // discretized weights
    threshold: i32,
    leak_factor: i32, // expected to be <= 0 (e.g. -1, -2)
) -> Feasibility {
    // Sum only positive excitatory weights
    let max_excitation: i32 = weights.iter()
        .filter(|&&w| w > 0)
        .sum();

    // Ensure we are working with the magnitude of the leak
    let leak_magnitude = leak_factor.abs();

    // Safety Check (Soundness): Input must overcome leak
    if max_excitation <= leak_magnitude {
        return Feasibility::Impossible;
    }

    let min_required = threshold / (1 + leak_magnitude);

    if max_excitation >= threshold {
        Feasibility::SingleStep
    } else if max_excitation >= min_required {
        // Steps = ceil(Threshold / Net_Gain)
        let net_gain = max_excitation - leak_magnitude;
        let steps = (threshold + net_gain - 1) / net_gain;
        Feasibility::MultiStep { min_steps: steps }
    } else {
        Feasibility::Impossible
    }
}
```

5.3. Biological Property Preservation

We verify the quotient model against standard Leaky Integrate-and-Fire properties defined in the literature [2].

Definition. The following biological properties should be preserved by the discretized model:

Tonic Spiking: The model fires periodically under constant input C_{in} if and only if:

$$C_{\text{in}} > |\lambda_d|$$

Integrator: The model fires on simultaneous inputs n if and only if:

$$\sum_{i=1}^{n} \delta_W(w_i) \geq T_d$$

Excitability: The inter-spike interval (ISI) decreases monotonically as input frequency increases. Specifically, the number of steps between spikes is:

$$N_{\text{steps}} = \left\lceil \frac{T_d}{C_{\text{in}} - |\lambda_d|} \right\rceil$$

which decreases as C_{in} increases.

💡 **Intuition:** These properties ensure our discretized model behaves like a real LIF neuron:

- *Tonic spiking* means sustained input produces sustained output
- *Integrator* means simultaneous inputs can trigger immediate firing
- *Excitability* means "the more input you receive, the faster you fire"

Remark. Immersion Memory: After emitting a spike, the potential resets to 0. The neuron has no memory of past input signals — only the accumulated potential matters. This is preserved by our model since we explicitly reset $P = 0$ after firing.

6. Variable Dependencies

This section clarifies the relationships between the key variables in our discretization scheme.

6.1. Dependency Graph

```
┌──────────────────────────────────────────────────────────────────┐
│                     Variable Dependency Graph                    │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│    ┌──────────┐                                                  │
│    │  w_max   │ (Original max weight, typically 100)             │
│    └────┬─────┘                                                  │
│         │                                                        │
│         ▼                                                        │
│    ┌──────────┐      ┌──────────┐                                │
│    │    W     │◄─────│    m     │ (Fan-in constraint: W ≥ m/2)   │
│    └────┬─────┘      └──────────┘                                │
│         │                                                        │
│         ├────────────────┐                                       │
│         ▼                ▼                                       │
│    ┌──────────┐      ┌──────────┐                                │
│    │  δ_W(w)  │      │   T_d    │ (Discretized threshold)        │
│    └──────────┘      └────┬─────┘                                │
│                           │                                      │
│         ┌─────────────────┤                                      │
│         ▼                 ▼                                      │
│    ┌──────────┐      ┌──────────┐                                │
│    │    r     │─────►│   λ_d    │ (Leak depends on T_d & r)      │
│    └──────────┘      └──────────┘                                │
│    (Retention)            │                                      │
│                           ▼                                      │
│                      ┌───────────┐                               │
│                      │Feasibility│                               │
│                      └───────────┘                               │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

Listing 1: Variable dependencies in the discretization scheme. The crucial link is λ_d depending on T_d rather than k.

6.2. Summary Table

Variable	Previous Definition	Revised Definition (v4)
λ_d	$-\mathrm{round}((1-r)\cdot k)$ (Depends on classes k)	$-\max(1, \lfloor(1-r)\cdot T_d\rfloor)$ (Depends on threshold T_d)
T_d	$\mathrm{ceil}\left(T \cdot \frac{W}{w_{\max}}\right)$	Unchanged
k	Independent parameter	Remains for class discretization, but λ_d now independent of k

Table 2: Summary of variable definition changes between revisions

7. PRISM Model Structure

The weighted quotient model generates PRISM code with explicit weight constants and contribution formulas. Note that weights are static constants, minimizing state explosion.

```
// Threshold-Dependent Leak Calculation
const int T_d = ceil(T_orig * W / w_max);
```

```
const int r_percent = 90;  // Retention as percentage
const int lambda_d = -max(1, floor((100 - r_percent) * T_d / 100));

// Weight constants (discretized)
const int W = 3;  // Discretization parameter
const int W_in0_2 = 3;   // δ_3(100) = 3
const int W_n1_2 = -2;   // δ_3(-67) = -2
const int W_n3_2 = 1;    // δ_3(33) = 1

// Contribution formula (evaluated at runtime)
formula contrib_2 = W_in0_2 * x0 + W_n1_2 * z1_2 + W_n3_2 * z3_2;

// Class delta (clamped)
formula delta_2 = max(-4, min(4, contrib_2));

module Neuron
    p : [0..T_d + 5] init 0;
    spike : bool init false;

    // Update with Safety Clamp (max(0, ...))
    [tick] true ->
        (p' = max(0, min(T_d + 5, p + contrib_2 + lambda_d)))
        & (spike' = (p' >= T_d));
endmodule
```

8. Visualizations

> ⚠️ **TODO: Visualization 1: Analog vs Discrete Dynamics**
>
> Create a side-by-side comparison showing:
> - Left: Continuous exponential decay curve (analog neuron)
> - Right: Step function decay (discrete neuron)
> - Highlight the "conservative bound" — discrete model fires slightly earlier/more often than analog to ensure completeness
> - Show scenarios: (a) spike occurs, (b) decay without input, (c) accumulation to threshold

> ⚠️ **TODO: Visualization 2: Variable Dependency Graph**
>
> A proper graphical diagram showing:
> - Nodes: W, $w_{\max}$, T_d, λ_d, r, k, m
> - Edges (directed): showing functional dependencies
> - Crucial highlighted edge: $T_d \to \lambda_d$ (new in v4)
> - Color coding: input parameters (blue), derived parameters (green), constraints (orange)

9. Conclusion

We have established a formal framework for weight discretization in quotient model abstraction. The key contributions in this revision include:

1. **Threshold-Dependent Leak:** The new formulation $\lambda_d = -\max(1, \lfloor (1-r) \cdot T_d \rfloor)$ ensures leak scales with the actual potential range, not the arbitrary number of classes.

2. **Soundness Theorem:** Proof that no spurious spikes occur — if inputs don't cause firing in the original model, the discretized model also remains silent.
3. **Biological Property Preservation:** Verification that tonic spiking, integrator behavior, and excitability are maintained in the quotient model.
4. **Variable Dependency Clarification:** Clear documentation of how λ_d depends on T_d and r, breaking the previous incorrect dependency on k.
5. **Corrected Code Examples:** Fixed syntax errors in feasibility check implementation.

Future Work:

- Implement graphical visualization comparing analog vs discrete dynamics
- Formal PRISM model checking with the revised leak formulation
- Extend to multi-neuron networks with inter-neuron weight calibration

---

Bibliography

[1] C. Baier and J.-P. Katoen, *Principles of Model Checking.* MIT Press, 2008.

[2] E. D. Maria, C. D. Giusto, and L. Laversa, "Spiking Neural Networks Modelled as Timed Automata with Parameter Learning," *Natural Computing*, vol. 19, pp. 135–155, 2020.