Configuring Safe Spiking Neural Controllers for Cyber-Physical Systems through Formal Verification

Arkaprava Gupta, Sumana Ghosh, Ansuman Banerjee, and Swarup Kumar Mohalik

***Abstract*—In this paper, we address the problem of safety verification for Spiking Neural Networks (SNNs) with Spiking Rectified Linear Activation (SRLA). The SNNs are obtained by first training Artificial Neural Networks (ANNs) and then translating to SNN with subsequent hyperparameter tuning. We propose a solution which tunes the *temporal window* hyperparameter of the translated SNN to ensure both accuracy and compliance with the safe range specification that requires the SNN outputs to remain within a safe range. We demonstrate our approach with experiments on 5 benchmark neural controllers.**

***Index Terms*—Spiking Neural Networks, Verification, Safe Range Computation, Spiking Rectified Linear Activation**

I. Introduction

In recent times, ANNs and their variants have been increasingly used as controllers for controlling the physical system (plant) in complex Cyber-physical systems (CPSs) [1]–[3]. In particular, the usage of SNNs for CPS control has garnered considerable attention due to their energy advantage over ANNs while providing comparable accuracies [4]–[7]. SNNs with Spiking Rectified Linear Activation (SRLA) [8] have been gaining attention recently [9] because of their seamless conversion from ANNs with ReLU activation [8], thereby, making them efficacious for both classification and regression tasks. ANN-based controllers mostly use ReLU activations because of their robust performance in noisy environments and with complex nonlinear plant dynamics.

A popular trend in the SNN community today is to first train an ANN model and then transform it into an SNN ensuring a high level of accuracy relative to the original ANN in terms of metrics like Mean Squared Error (MSE). This is done to avoid resource-hungry SNN training algorithms while taking advantage of the already verified ANN models. During this conversion, a crucial hyperparameter to be fixed is the SNN *temporal window* or the input sequence length (called NUMSTEPS). Larger temporal windows can improve accuracy but increase computational demands and latency.

State-of-the-art ANN-SNN conversion algorithms attempt to strike a balance between accuracy and latency during the selection of NUMSTEPS, taking into consideration constraints on SNN latency. While there has been a handful of research efforts in establishing the equivalence between ANN and SNN [7], [10], [11] (though not in the CPS context), and a plethora of work in the verification of ANN-controlled CPSs [3], [12]–[15], the problem of safety assurance of the translated SNN controllers remains completely unexplored in literature. We address this problem here illustrating the solution for the safe range requirement which ensures that the SNN output always remains within a safe range. In particular, we derive an upper bound for NUMSTEPS from the control period of the CPS and the execution time for one step of the SNN and provide an iterative procedure to compute the least value of NUMSTEPS ensuring required accuracy and safe range compliance. Further, given an SNN and value of NUMSTEPS, and a specified safe range, we provide a formal verification procedure to check for compliance. Additionally, we present an iterative bound-tightening method to produce tight bounds for the output ranges of the SNN. We validate our framework with experiments on 5 benchmark neural controllers [16], [17].

Arkaprava Gupta (email: arkaprava.gupta@gmail.com) and Swarup Kumar Mohalik (email: swarup.kumar.mohalik@ericsson.com) are with Ericsson India Pvt Ltd, Bangalore 560048, India. Sumana Ghosh (email: sumana@isical.ac.in) and Ansuman Banerjee (ansuman@isical.ac.in) are with Indian Statistical Institute, Kolkata, India. This work was partially supported by funding from Ericsson India Pvt Ltd.

II. System Model and Problem Definition

An SNN consists of neurons that process a sequence of spike inputs over the temporal window (i.e., NUMSTEPS) and produce spike outputs of the same length. The spikes can have binary or non-binary amplitudes. The final output is an aggregation of the output spikes (e.g., the average of the spike values over NUMSTEPS). We denote by $\mathcal{N}_T$ an SNN $\mathcal{N}$ where NUMSTEPS has been set to T. When the plant observables I are sent to the SNN $\mathcal{N}_T$, the input I is repeated for all the T steps as input. The upper bound T_{up} on T is computed from the control period p and the execution time e as: $T_{up} = \lfloor p/e \rfloor$. Generally, SRLA neurons accumulate potential (P) by calculating the product of the incoming weights with the output spike amplitudes of the neurons in the previous layer until they reach their threshold (θ). Upon reaching θ, the neurons spike with an amplitude $\lfloor P/\theta \rfloor$, and their membrane potential is reset to $P - \lfloor P/\theta \rfloor$.

Given an ANN $\mathcal{A}$, we denote by $|N_{op}|$ the number of outputs and by $R_i(\mathcal{A})$ the range (a closed interval $[l, u]$) for the i-th output. Let the SNN $\mathcal{N}$ be obtained by an ANN-SNN conversion procedure $Trans(.)$. $\mathcal{N}$ has the same number and order of outputs as the ANN $\mathcal{A}$. We denote by $S_i(\mathcal{N})$ the ranges of the i-the output of $\mathcal{N}$. $MSE_i(D, \mathcal{A}, \mathcal{N})$ denotes the MSE value for the i-th output computed using an input dataset D and obtaining the output value by simulating $\mathcal{A}$ and $\mathcal{N}$.

Problem and Motivation: Given an ANN $\mathcal{A}$, the SNN $\mathcal{N} = Trans(\mathcal{A})$, an input dataset D, an MSE bound ϵ, and an integer upper bound T_{up}, *our objective* is to find the smallest timestep $T \leq T_{up}$ such that for each $i \in \{1, \ldots, |N_{op}|\}$, $MSE_i(D, \mathcal{A}, \mathcal{N}_T) \leq \epsilon$ and for each i, $S_i(\mathcal{N}_T) \leq R_i(\mathcal{A})$.

According to the problem definition, the desired SNN controller should have good accuracy (in terms of MSE) relative to the original ANN, and also ensure that the outputs never exceed the bounds of the ANN for any input. Since the ANN is assumed to be safe, this ensures that the SNN also remains safe. The accuracy alone is not sufficient for the safe range satisfaction as illustrated next. We consider an SNN controller for a double pendulum [16] system having two output neurons and an MSE objective of 0.4. Table I shows the MSE values recorded for the SNN controller for these two output neurons along with the verification results of its safe range requirement. The safe ranges of these two output neurons are $R_1(\mathcal{A}) = [-5.86571, -3.69253]$ and $R_2(\mathcal{A}) = [-6.35836, -3.41698]$. It can be expected that a lower MSE corresponds to a better convergence of the SNN output towards the ANN output. However, the satisfaction of the MSE bound does not ensure the safe range requirement, as shown in Table I. This mandates a formal safety verification.

# of Timesteps	MSE Output 1	MSE Output 2	Verification Result
1	4.27391	1.71482	Unsafe
2	0.54261	0.13999	Unsafe
3	0.36017	0.07155	Unsafe
4	0.20660	0.04399	Safe

TABLE I: Safety Verification Exmaple

III. Proposed Methodology

Algorithm 1 is an outline of our framework for finding the least value of NUMSTEPS for a given SNN to satisfy safety.

```
Algorithm 1: Find NUMSTEPS for Safe Range
Input  : ANN $\mathcal{A}$, SNN $\mathcal{N}$, Minimum MSE $\epsilon$, Samples
         $D$ from the input space, Timestep upper bound $T_{up}$,
         Safe Ranges $\{[l_i, u_i]\}_{i=1}^{|N_{op}|}$, for output neurons in $N_{op}$
Output: NUMSTEPS
$range \leftarrow \{[l_i, u_i]\}_{i=1}^{|N_{op}|}$
for $T \leftarrow 1$ to $T_{up}$ do
    if MSE($D$, $\mathcal{A}$, $\mathcal{N}_T$) $< \epsilon$ then
        if VERIFY( $\mathcal{N}_T$, $D$, $range$) holds then
            return $T$
        else
            $res \leftarrow$ counterexample returned by VERIFY()
            $snn\_range \leftarrow$ SNN_BOUNDS( $\mathcal{N}_T$, $res$)
            if $snn\_range$ is acceptable then
                return $T$
    display " $No\ such\ T\ is\ found$ "
```

The algorithm first sets the safe range of the output neurons. It then iteratively searches for a NUMSTEPS value starting from 1, where the MSE values calculated for all the outputs using the dataset D are less than the accuracy objective ϵ. Only then the safe range verification procedure is called. If the verification succeeds (i.e., no violation), we return the iteration step as the value for NUMSTEPS. When the verification fails for the safe range requirements from $\mathcal{A}$, we return a counterexample res, and compute the actual range supported by $\mathcal{N}_T$ using res. In our work, the safe range specifications are calculated from the original ANN using the most efficient reachability analyzer tool, POLAR-Express [3]. MSE computation is done from the outputs resulting from simulating the ANN $\mathcal{A}$ and SNN $\mathcal{N}$ on D.

Algorithm 2 explains the working of the `VERIFY()` function. `VERIFY()` consists of two separate steps:

```
Algorithm 2: VERIFY
Input  : SNN $\mathcal{N}_\mathcal{T}$, I/p Samples $D$, Safe Range $range$
Output: $False$ with a counterexample input in $res$
        that violates the safe range spec., $True$ otherwise
procedure VERIFY($\mathcal{N}_\mathcal{T}$, $D$, $range$)
    if SIMULATE($\mathcal{N}_T$, $D$, $range$) returns no
      violation then
        if FV( $\mathcal{N}_T$, $range$) is $True$ then
            return $True$
    return $False$, $res$
```

- *Simulation*: The SNN is simulated using the input samples D to check if there is a safe range violation by some input (returned in res as a counterexample). This is an inexpensive step. When the simulation finds no violation, only then the more rigorous verification is invoked.
- *Formal Verification*: This involves encoding the SNN in MILP and using an MILP solver to verify if the given safe range for the SNN is always satisfied. When `FV()` returns $True$, the safe range for the SNN holds for all possible inputs or $False$ otherwise with a counterexample res.

The MILP encoding $F_{\mathcal{N}_T}$ of a given SNN is a conjunction of constraints expressing its operation across all neurons over NUMSTEPS T. The description of `FV()` is given in the detailed version [18] and is mostly based on the formulation in [5]. We now discuss the safe range specification encoding.

Safe Range Specification: We encode the *negation of the safe range requirement* so that we can verify its satisfaction. We are given the safe ranges for all the output neurons in the SNN, i.e, $\{[l_i, u_i]\}_{i=1}^{|N_{op}|}$, where $|N_{op}|$ is the total number of output neurons. We check for satisfaction against these two bounds with two separate queries as below.

$$\psi_{ub} \triangleq F_{\mathcal{N}_T} \ \wedge \ (op_i \geq u_i), \quad \psi_{lb} \triangleq F_{\mathcal{N}_T} \ \wedge \ (op_i \leq l_i) \tag{1}$$

Note that op_i is the output value of the i-th output neuron. With both queries, we search for an input that violates the given safe range specification. It holds only if both the above queries are proven infeasible by the solver, indicating that no input can trigger an output that is outside the safe range. We can extend this for checking the upper (lower) bounds of all the output neurons in N_{op} by taking the disjunction of the upper bounds (lower bounds) on op_i for each $i \in N_{op}$.

Formal Verification: `FV()` outlined in Algorithm 3 first encodes the SNN $\mathcal{N}_T$. Next, it calculates the lower bound L and the upper bound U from the safe range, $range$, given as input to it. It then invokes the `FV_LB()` and `FV_UB()` for verifying the safe output bounds. When any of these returns

Algorithm 3: FV

```
Input  : SNN runs for T timesteps, $\mathcal{N}_T$, Safe range
for SNN outputs, i.e., $range = \{[l_i, u_i]\}_{i=1}^{|N_{op}|}$
Output: $False$ with a counterexample input in $res$
that violates the safe range, $True$ otherwise
procedure FV($\mathcal{N}_T, range$)
    $F_{\mathcal{N}_T} \leftarrow$ ENCODE($\mathcal{N}_T$)
    $L \leftarrow \{l_i\}_{i=1}^{|N_{op}|}$,   $U \leftarrow \{u_i\}_{i=1}^{|N_{op}|}$
    if FV_LB($F_{\mathcal{N}_T}$, $L$) returns $False$ then
        return $False$, $res$
    if FV_UB($F_{\mathcal{N}_T}$, $U$) returns $False$ then
        return $False$, $res$
    return $True$
```

$False$ (violation of safe range), FV() terminates and returns $False$ with the counterexample, otherwise, it returns $True$.

Algorithm 4: FV_UB

```
Input  : SNN encodings which runs for T timesteps
i.e., $F_{\mathcal{N}_T}$, Safe upper bounds, $U = \{u_i\}_{i=1}^{|N_{op}|}$
Output: $False$ when an input that violates the safe
upper bound exists, $True$ otherwise
procedure FV_UB($F_{\mathcal{N}_T}$, $U$)
    $\psi_{ub}^T \leftarrow Q_{ub}$($F_{\mathcal{N}_T}$, $U$)
    if $\psi_{ub}^T$ is $feasible$ then
        return $False$
    else
        return $True$
```

FV_UB() in Algorithm 4 uses the $Q_{ub}()$ function internally to encode the upper output bound for the safety check. Similarly FV_LB() is used for the lower bound check.

Binary Search for Iterative Bound Tightening: When the given safe range specification of the SNN is violated, we compute the actual range supported by the SNN outputs employing the function SNN_BOUNDS(). However, we observe that computing the SNN output range through an objective function often leads to timeouts. Hence, we use a binary search algorithm that uses FV_UB() and FV_LB() as subroutines to estimate tight ranges within a factor δ of the actual ranges obtained during the safe range verification in Algorithm 2.

SNN_BOUNDS() is implemented using two functions FIND_UB() and FIND_LB() to tighten the upper and lower bounds of the outputs of a given SNN $\mathcal{N}_T$ respectively. Algorithm 5 outlines FIND_UB(). The algorithm for FIND_LB() is similar in nature. Recall that in Algorithm 1, we store the counterexample in res, returned by the VERIFY() in Algorithm 2. The value stored in res is the input for which the safe range gets violated. FIND_UB() starts by setting the variable U_i^{ce} of the i-th output neuron, to the actual upper bound it gets by executing the counterexample in res. The value of U_i^{ce} is then incremented iteratively by a value of β and then verified using the procedure FV_UB() for K iterations till it returns $True$ (here, we run FV_UB() with Eq. (1) only). This ensures an upper bound supported by the SNN $\mathcal{N}_T$. At this point, the least upper bound lies in the interval $(U_i^{vio}(left),\ U_i^{ce}(right)]$. The interval is tightened using a variant of binary search [19] between these two values.

Algorithm 5: FIND_UB

```
Input  : Encoding $F_{\mathcal{N}_T}$ of SNN $\mathcal{N}_T$, Value of $T$ for
which MSE < $\epsilon$, Safe upper bound of $i$-th output
neuron $u_i$, Iteration bound $K$, parameters $\beta$ and $\delta$
Output: The tightened upper bound $U_i^{tgt}$
procedure FIND_UB($F_{\mathcal{N}_T}$, $\mathcal{L}$, $U_i$)
    // Increase bound until FV_UB() returns $True$
    $U_i^{ce} \leftarrow u_i$
    for $k \leftarrow 1$ to $K$ do
        $U_i^{vio} \leftarrow U_i^{ce}$ ,   $U_i^{ce} \leftarrow U_i^{ce} + \beta$
        $ans \leftarrow$ FV_UB($F_{\mathcal{N}_T}, U_i^{ce}$)
        if $ans$ is $True$ then
            $break$
    // Binary Search to find a tightened bound
    $left \leftarrow U_i^{vio}$,   $right \leftarrow U_i^{ce}$
    while $(right - left) > \delta$ do
        $mid \leftarrow$ RAND($left + \delta, right$)
        if SIMULATE($\mathcal{N}_T, mid$) $finds\ violation$ then
            $left \leftarrow mid$
        else if FV_UB($F_{\mathcal{N}_T}, mid$) = $False$ or $TO$ then
            $left \leftarrow mid$
        else
            $right \leftarrow mid$
    $U_i^{tgt} \leftarrow right$
    return $U_i^{tgt}$
```

The function RAND(.) generates a random real value mid within the range $(left{+}\delta, right)$. We check if the value of mid is not an appropriate upper bound of the i-th output neuron of $\mathcal{N}_T$ through a random simulation (using input samples D) or a formal FV_UB($F_{\mathcal{N}_T}, mid$), in which case the $left$ bound of the interval is updated to mid. When FV_UB($F_{\mathcal{N}_T}, mid$) has a Timeout (TO), we conservatively increment $left$ to mid. If FV_UB($F_{\mathcal{N}_T}, mid$) is True, it implies we have a better upper bound than the current $right$, hence it is updated to mid. The while loop stops when we get an upper bound ($right$) within δ of the lower bound ($left$) and returns $right$ as the δ-tight upper bound U_i^{tgt}. The value of δ is user-specified, taken as 0.001 for our experiments.

IV. Implementation and Results

Benchmarks: We consider 5 benchmark ANN controllers from ARCH workshop competition years 2019 [16] and 2022 [17]. These include controllers for a linear inverted pendulum (LIP), a double pendulum (DP), a single pendulum (SP), and adaptive cruise controllers with 3 (ACC3) and 5 hidden layers (ACC5). Details are given in Column 1 of Table II.

Experimental Setup: All experiments were done on an Intel Core i7 CPU with a 1.30 GHz clock speed and 16 GB RAM.

Benchmarks	# of Timesteps	MSE	Verification Result	Range Obtained from Bound Tightening	Total Time Taken
Linear Inverted Pendulum	1	13.03031	–	–	–
Arch.: $4 \times 10 \times 1$	2	3.32945	–	–	–
$T_{up} = 25$	⋮	⋮	⋮	⋮	⋮
Safe Range: $[-15.50883, 15.34465]$	9	0.16450	–	–	–
	10	0.13235	Safe	–	$0.071s$
Double Pendulum	1	$4.27391, 1.71482$	–	–	–
Arch.: $4 \times 25 \times 25 \times 2$	2	$0.54261, 0.13999$	–	–	–
$T_{up} = 25$	⋮	⋮	⋮	⋮	⋮
Safe Range: $[-5.86571, -3.69253]$,	5	$0.12962, 0.02461$	–	–	–
$[-6.35836, -3.41698]$	6	$0.08968, 0.02089$	Safe	–	$3.736s$
Single Pendulum	1	0.43389	–	–	–
Arch.: $2 \times 25 \times 25 \times 1$	4	0.04138	Unsafe	$[-0.78130, -0.29494]$	$0.187s$
$T_{up} = 20$	⋮	⋮	⋮	⋮	⋮
Safe Range: $[-0.78130, -0.54282]$	19	0.00099	Unsafe	$[-0.78130, -0.52872]$	$5673.547s$
	20	0.00103	Unsafe	$[-0.78130, -0.51042]$	$9962.996s$
ACC3	1	0.01412	–	–	–
Arch.: $5 \times 20 \times 20 \times 20 \times 1$	2	0.00356	–	–	–
$T_{up} = 5$	3	0.00164	Unsafe	$[-1.47700, -0.61709]$	$424.906s$
Safe Range: $[-1.46030, -0.73179]$	4	0.00090	Unsafe	$[-1.46733, -0.64829]$	$2817.854s$
	5	0.00058	Unsafe	$[-1.46616, -0.66411]$	$7902.435s$
ACC5	1	0.00590	–	–	–
Arch.: $5 \times 20 \times 20 \times 20 \times 20 \times 20 \times 1$	2	0.00145	–	–	–
$T_{up} = 5$	3	0.00070	Unsafe	$[-8.82560, -6.81908]$	$1019.328s$
Safe Range: $[-8.81953, -6.85014]$	4	0.00038	Unsafe	$[-8.91954, -6.81121]$	$23918.844s$
	5	0.00024	Unsafe	$[-8.90126, -6.78736]$	$15508.534s$

TABLE II: Safe Range Verification Results on SNN-Controllers

We used POLAR-Express (commit 13d42b0 downloaded from GitHub on Aug. 18, 2023), to calculate the safe range of the above-mentioned ANN controllers and the Gurobi MILP solver (version 10.0.3) for constraint solving and verification. We used the Nengo [8] framework for ANN-SNN translation. Nengo accomplishes the transformation by replacing the ReLU activation with SRLA, while preserving the same network architecture, including weights and biases. For any neuron, the default value of the threshold in Nengo is 1.

Computing Safe Range and Upper bounds on NUMSTEPS: We calculate the safe range specifications (Table II Column 1) for all ANN controllers running POLAR-Express [3] based on the initial conditions of the input variables as reported in [16], [17]. In particular, we extract the safe bounds of the controllers after a single plant-control iteration. Though it can be doable considering all the plant-control iterations, however, this is currently out of the scope of this paper. Now, the queries to be verified by the solver are the negation of these safe ranges. The upper bound, T_{up}, on NUMSTEPS for SNN controller (Table II Column 1) is calculated from the control period given in [16], [17] and by assuming the execution time of the respective SNN for a single timestep as $0.002\,\mathrm{s}$ for LIP, $0.008\,\mathrm{s}$ for DP, $0.0025\,\mathrm{s}$ for SP, and $0.02\,\mathrm{s}$ for the ACCs.

Results and Analysis: We run our framework as described in Algorithm 1 for each of the five benchmark controllers. We consider the minimum MSE bound ϵ for LIP, DP, SP, ACC3, and ACC5 as 0.15, 0.10, 0.05, 0.002 and 0.001 respectively. These bounds are selected within 1% envelope of the safe range specifications. Table II summarizes the overall results. Column 3 shows the MSE recorded for each of the SNN controllers based on 5000 random samples while running our framework. As per Algorithm 1, we proceed with the verification only when the MSE values obtained by the SNN controllers are below the given minimum bound ϵ. Note that we do not verify (respective cells are marked by –) the safe ranges for timesteps up to 9, 5, 3, 2 and 2 for LIP, DP, SP, ACC3, and ACC5 respectively. For LIP and DP, the respective SNN controllers satisfy the given safe range requirements, hence, we obtain the NUMSTEPS values as 10 and 6 respectively and stop running Algorithm 1 further. These NUMSTEPS values (i.e., 10 and 6) are quite small, and hence, suitable for SNN implementation in practice.

However, for the other 3 systems, though the respective SNN controllers satisfy the MSE criteria, they fail the safe range verification with all timesteps up to T_{up}. This is obtained through only 500 random input simulations as described in Algorithm 2, before the call to formal verification in line 3. Thus, we avoid the costly verification calls for SP, ACC3 and ACC5. Thereafter, for these SNN controllers, tightening ranges over their actual ranges is done from the counterexample returned by the solver on running Algorithm 2. For all the unsafe instances, the tight ranges obtained after running Algorithm 5, are shown in Column 5. These can help the designer for further improvement of SNN attributes (e.g., stepsize). Column 6 reports the total time to run Algorithm 1 for each case.

V. Conclusion and Future Work

In this paper, we address the problem of determining the appropriate temporal window of an SNN controller such that the SNN is both accurate w.r.t. its respective ANN controller and ensures the safe range specification as well through formal verification. We have experimented on five benchmark neural controllers. For some of the SNN controllers that violate their safe range, we provide an iterative bound tightening method to approximate the safe range of the SNN controllers. Developing a closed-loop implementation integrating both the plant and the SNN controller is the immediate future step of this work.

REFERENCES

[1] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[2] Pavithra Prabhakar and Zahra Rahimi Afzal. Abstraction based output range analysis for neural networks. In *NEURIPS*, pages 15762–15772, 2019.

[3] Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. Polar: A polynomial arithmetic framework for verifying neural-network controlled systems. In *ATVA*, page 414–430, 2022.

[4] Elisabetta Maria, Cinzia Di Giusto, and Laetitia Laversa. Spiking neural networks modelled as timed automata with parameter learning. *Natural Computing*, 19:135–155, 03 2020.

[5] Soham Banerjee et al. Smt-based modeling and verification of spiking neural networks: A case study. In *VMCAI 2023*, pages 25–43, 2023.

[6] Ankit Pradhan, Jonathan King, Srinivas Pinisetty, and Partha S. Roop. Model based verification of spiking neural networks in cyber physical systems. *IEEE Transactions on Computers*, 72(9):2426–2439, 2023.

[7] Bingsen Wang et al. A new ann-snn conversion method with high accuracy, low latency and good robustness. In *IJCAI*, pages 3067–3075, 2023.

[8] Trevor Bekolay et al. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7(48):1–13, 2014.

[9] Amirreza Yousefzadeh et al. Asynchronous spiking neurons, the natural key to exploit temporal sparsity. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(4):668–678, 2019.

[10] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in Neuroscience*, 13, 2019.

[11] Haoran Gao et al. High-accuracy deep ann-to-snn conversion using quantization-aware training framework and calcium-gated bipolar leaky integrate and fire neuron. *Frontiers in Neuroscience*, 17, 2023.

[12] Hoang-Dung Tran et al. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems*, 18(5s), oct 2019.

[13] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *HSCC*, pages 157–168, 04 2019.

[14] Jiameng Fan, Chao Huang, Xin Chen, Wenchao Li, and Qi Zhu. Reachnn*: A tool for reachability analysis of neural-network controlled systems. In Dang Van Hung and Oleg Sokolsky, editors, *ATVA*, pages 537–542, 2020.

[15] Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas, and Insup Lee. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *CAV*, pages 249–262, 2021.

[16] Diego Manzanas Lopez et al. Arch-comp19 category report: Artificial intelligence and neural network control systems for continuous and hybrid systems plants. In *International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 61, pages 103–119, 2019.

[17] Gidon Ernst et al. Arch-comp 2022 category report: Falsification with ubounded resources. In *International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 90, pages 204–221, 2022.

[18] Arkaprava Gupta, Sumana Ghosh, Ansuman Banerjee, and Swarup Kumar Mohalik. Configuring safe spiking neural controllers for cyber-physical systems through formal verification. https://arxiv.org/abs/2408.01996, 2024.

[19] Eitan Zemel. Random binary search: A randomizing algorithm for global optimization in r1. *Mathematics of Operations Research*, 11(4):651–662, 1986.