Toward Precision-Aware Safe Neural-Controlled Cyber–Physical Systems

Harikishan Thevendhriya, Sumana Ghosh, and Debasmita Lohar

***Abstract*—The safety of neural network (NN) controllers is crucial, specifically in the context of safety-critical applications. Current safety verification focuses on the reachability analysis, considering the bounded errors from the noisy environments or inaccurate implementations. However, it assumes real-valued arithmetic and does not account for the fixed-point quantization often used in the embedded systems. Some recent efforts have focused on generating the sound quantized NN implementations in fixed-point, ensuring specific target error bounds, but they assume the safety of NNs is already proven. To bridge this gap, we introduce *Nexus*, a novel two-phase framework combining reachability analysis with sound NN quantization. Nexus provides an end-to-end solution that ensures CPS safety within bounded errors while generating mixed-precision fixed-point implementations for the NN controllers. Additionally, we optimize these implementations for the automated parallelization on the FPGAs using a commercial HLS compiler, reducing the machine cycles significantly.**

***Index Terms*—Cyber–physical system (CPS), quantization, reachability analysis.**

I. Introduction

MODERN cyber–physical systems (CPSs) are rapidly evolving to handle the complex functionalities in highly dynamic, nonlinear, and uncertain environments. These closed-loop systems are involving neural networks (NNs) as controllers for their flexibility and adaptability. However, deploying NN-controlled systems in safety-critical applications like the adaptive cruise control for the cars or airplane collision avoidance systems [1] requires ensuring safety and being optimized for efficiency.

Recent efforts have focused on automatically verifying the safety of NN-controlled systems [2], [3], [4], [5]. These methods consider the NN controllers trained in high-precision floating-point arithmetic on the powerful machines with GPUs emulating the exact real-valued arithmetic. These controllers sense system states at discrete intervals, compute control values, and adjust system dynamics based on ordinary differential equations. The aim of the verification is to estimate safe, reachable states of the system within a finite time, considering the bounded errors from the noisy environments or inaccurate implementations.

Manuscript received 6 August 2024; accepted 9 August 2024. This work was partially supported by funding from KiKIT, the pilot program for Core Informatics of the Helmholtz Association (HGF). This manuscript was recommended for publication by A. Shrivastava. *(Corresponding author: Debasmita Lohar.)*

Harikishan Thevendhriya and Sumana Ghosh are with the Department of Computer Science, Indian Statistical Institute, Kolkata 700108, India (e-mail: harikishants9899@gmail.com; sumana@isical.ac.in).

Debasmita Lohar is with the KASTEL, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany (e-mail: debasmita.lohar@kit.edu).

Digital Object Identifier 10.1109/LES.2024.3444004

However, directly implementing high-precision floating-point NN controllers on the embedded architectures with the constrained resources is often impractical due to high computation costs or the lack of dedicated floating-point co-processors or software-based emulation capabilities. Consequently, these NNs, in practice, are quantized using the low-precision fixed-point arithmetic [6], [7] to optimize the area or latency. It is thus crucial to verify that after quantization, the implementations still satisfy the error bounds derived during the safety verification. Unfortunately, current safety verification methods do not consider the final implementations and thus do not guarantee that the implementations meet the derived error bounds.

Various automated quantization approaches exist for NNs. Some adopt uniform precision for all the layers [6], [7], while the others focus on the mixed-precision quantization [10], [11], [12], which has gained more popularity for using different precisions for different operations or layers, leading to greater resource savings. While most quantization methods dynamically compare the classification accuracy without guaranteeing for all the possible inputs, only the tool Aster [12] generates the sound NN controller implementations that meet predefined error bounds. However, Aster assumes the safety of the NN controllers has been proven within the error bounds.

In this letter, we introduce *Nexus*, a two-phase framework that integrates safety verification with mixed fixed-point quantization of the NN controllers. We employ a state-of-the-art scalable reachability analyzer POLAR-Express [2] to prove the safety of the closed-loop NN-controlled system in a finite time and adapt it to consider an error for implementations. If the system is proven safe, we then utilize the only existing sound NN quantizer Aster to generate a mixed fixed-point implementation guaranteeing the error bound. This implementation can then be compiled using the commercial HLS compilers. Furthermore, Nexus extends the Aster's code generation to produce the code with loops, automatically parallelizable by HLS compilers, thus reducing the latency of the final code. Our evaluation shows that the latencies of the Nexus's code are significantly less than the Aster's for all the seven safe benchmarks that we consider for our experiments. These reductions are particularly high for the three other larger benchmarks with thousands of parameters.

Though there is a large body of work on integrating different analyses, our approach is unique, as, to the best of our knowledge, Nexus is the first tool to offer an end-to-end solution for the precision-aware, safe, and NN-controlled CPSs. Our novel extensions also generate parallel code constructs, effectively leveraging the hardware. In summary, this letter makes the following contributions.

1) An end-to-end method with a prototype tool combining the closed-loop safety verification and sound quantization available at https://github.com/harikishants/Nexus.

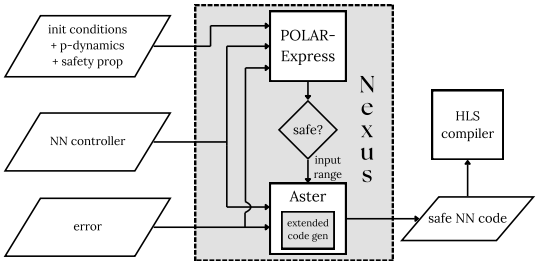

Fig. 1. Overview of Nexus.

```
def code_gen(in x̄, weight W, bias b, act α, lyr L):
  in ← x̄
  for l ← 2 to L: tmp, sum, out ← ∅
    for i ← neurons in layer l: // parallelized
      for j ← neurons in layer l−1: // parallelized
        tmp_j ← w^l_ij · in_j
      sum_i ← b^l_i
      for j ← neurons in layer l−1:
          sum_i ← sum_i + tmp_j
      out_i ← α_l(sum_i)
    in ← out
```

Fig. 2. Nexus's code generation.

2) An extended code generation compatible with the commercial HLS compilers for the automated parallelization.
3) An evaluation on the benchmarks collected from both the academic and industrial settings.

II. Nexus: Two-Phase Framework

Consider an inverted pendulum taken from [1]. The plant state is represented by a four-parameter vector $\bar{\mathbf{x}}$ (horizontal position and velocity of the cart, angular position, and velocity of the pendulum) with the initial conditions $x_1 \in [0.1, 0.11]$, $x_2 = x_3 = x_4 = 0$ and a control input $u = 0$. The plant dynamics are given by $\dot{x}_1 = x_2, \dot{x}_1 = x_2, \dot{x}_2 = 0.0043x_4 - 2.75x_3 + 1.94u - 10.95x_2, \dot{x}_3 = x_4, \dot{x}_4 = 28.58x_3 - 0.044x_4 + 4.44u + 24.92x_2$. The safety property requires keeping the pendulum vertically balanced by maintaining $x_1, x_2, x_3, x_4 \in [-0.2, 0.2]$ for 1 s with a control step size of 0.05 s. The NN controller takes $\bar{\mathbf{x}}$ as input and outputs the control action u. An error bound of $1e-5$ is considered to account for the implementation errors. Our goal is to prove that the system is safe and, if so, generate a controller implementation that satisfies the error.

Fig. 1 provides an overview of Nexus. It begins with the plant dynamics, initial conditions, safety property, and error bound and verifies system safety using the reachability analysis with POLAR-Express [2] under the real arithmetic. If the system is safe, Nexus extracts the max and min values to define the sound ranges of the plant states. Next, these ranges and the error bound are used to quantize the NN controller into a fixed-point mixed-precision C++ implementation, guaranteeing the error bound with Aster [12]. This approach ensures that Nexus not only proves the system safety but also generates the implementations that maintain the system safety and the error bound. The following sections explain each of these phases in detail.

A. First Phase: Safety Verification

In this phase, we prove safety by simulating the exact or over-approximated reachable sets containing all the possible trajectories of the plant. We then verify if any unsafe state can be reached from a given initial state. The underlying reachability analysis tool POLAR-Express iteratively computes functional over approximations of reachable states using a layer by layer propagation of the Taylor model TM(p, I), where p is the polynomial used for approximation and I denotes the interval remainder, capturing the over-approximation error (we refer the readers to [2] for more details). While adjusting I ensures the tight over-approximations, it does not account for the implementation errors that may arise when quantizing the NN controllers for the resource-constrained embedded platforms using the fixed-point arithmetic. To address this, in Nexus, we add a predefined quantization error (ϵ) during the safety verification. In each step of the reachability analysis, the error ϵ is added to the remainder I of the NN output. Thus, safety analysis is performed considering errors from both the over approximations of the reachability analysis and code quantization.

If Nexus proves that the system is safe, it extracts ranges ($\mathbf{R}_i$) for all the plant state variables from the ith simulation step by adding I to the over-approximated ranges ensuring soundness. From the set of all $\mathbf{R}_i$ for all the plant state variables, Nexus computes the max and min values to generate the sound ranges for these variables, which are the inputs to the NN controller.

For the inverted pendulum, Nexus proves that the system is safe for 20 simulation steps, considering the initial conditions and safety property, and generates the following ranges for the plant state variables: $x_1 \in [0.07, 0.11]$, $x_2 \in [-0.05, 0.05]$, $x_3 \in [-0.01, 0.00]$, and $x_4 \in [-0.12, 0.02]$.

B. Second Phase: Sound NN Quantization

In the next phase of Nexus, the goal is to quantize the safe real-valued NN controllers to generate a mixed-precision fixed-point (represented by the total number of bits and the binary point position deciding the number of fractional bits) code that can be run efficiently in embedded systems.

Nexus utilizes the ranges of NN input variables from the first phase and the error bound to generate an implementation by solving an optimization problem. This problem aims to minimize the required number of bits for all the variables and constants while adhering to the error bound and avoiding the overflow in fixed-point quantization. Internally, Nexus employs Aster to define and solve this optimization problem (for the detailed constraints, see [12]). If it is impossible to generate an implementation within the maximum allowed number of bits (e.g., 32 bits), Nexus returns an infeasible solution. Otherwise, it generates a complete C++ implementation directly compatible with the commercial HLS compilers, such as Xilinx Vitis HLS [13]. Consequently, an FPGA design can be synthesized, and the simulated running time (or latency) can be obtained in terms of the clock cycles. It is important to note that as Aster can only handle feed-forward DNNs with "linear" and "ReLU" activations, Nexus also operates under the same constraints.

For the inverted pendulum, we use the ranges generated in the first phase and configure Nexus to use a range [10, 32] of fractional bits for all the variables and an initial error of 2^{-32}. Nexus successfully generates an implementation for it.

Extended Code Generation: The underlying tool Aster generates implementations with both fully *unrolled* code (where the matrices and vectors are converted into the scalar

TABLE I
BENCHMARK DETAILS (PLANT CONTROLLER SPECIFICATIONS)

benchmarks	#plant-vars	neural controller spec. ctrl-step	#hid-lyr	#params
InvPend	6	0.05	1	60
MountCar	3	1.00	2	336
SglPend	4	0.05	2	775
DblPendV1	7	0.05	2	825
DblPendV2	7	0.02	2	825
ACC3	10	0.10	3	980
ACC5	10	0.10	5	1,820
ACC7	10	0.10	7	2,660
Unicycle	7	0.20	1	3,500
ACC10	10	0.10	10	3,920
Airplane	19	0.10	3	13,540
TORA	5	1.00	3	20,800

TABLE II
SAFETY ANALYSIS (✓: SAFE, ✗: UNSAFE, ×: ANALYSIS FAILS), LATENCIES OF SAFE CONTROLLER IMPLEMENTATIONS (INF: TOOL RETURNS INFEASIBLE) AND RUNNING TIMES OF NEXUS

benchmarks	error: $1e-3$, setting A safe	latency	time(s)	error: $1e-5$, setting B safe	latency	time(s)
InvPend	✓	12	3.15	✓	14	3.16
MountCar	✗	-	-	✓	25	66.35
SglPend	✓	23	5.16	✓	27	5.15
DblPendV1	✓	26	7.10	✓	27	6.80
DblPendV2	✓	26	5.80	✓	28	5.84
ACC3	✓	40	10.49	✓	39	10.47
ACC5	✓	inf	-	✓	63	19.78
ACC7	✓	inf	-	✓	inf	-
Unicycle	×	-	-	×	-	-
ACC10	×	-	-	×	-	-
Airplane	✗	-	-	✗	-	-
TORA	✗	-	-	✗	-	-

variables) and a *looped* version that keeps the data structures intact. However, it does not leverage the inherent parallelism available in custom hardware like FPGAs. Nexus extends the vanilla code generation by adding the directives and nested looped constructs to enable the automated parallelism in the standard HLS compilers, synthesizing a parallelized hardware design that reduces the latency. We present the extended code generation in Fig. 2. In an NN, each layer depends on the previous layer, requiring the serial computation across layers. However, within each layer, the computation of each neuron's output is independent and can be parallelized (line 4). Additionally, the multiplications of weights and inputs within each neuron are independent and can be parallelized (line 5). Thus, Nexus returns a C++ code with nested loops and directives.

We utilized the Nexus's code generation for the inverted pendulum example. Nexus's nested looped code achieved a latency of 14 cycles outperforming the vanilla Aster's unrolled code with 16 cycles and looped code with 18 cycles.

III. EXPERIMENTAL EVALUATION

a) Experimental Setup: We used 12 NN controllers from the Aster's benchmark set [12] and collected the respective plant dynamics from the competition at the ARCH workshop from the years 2019 [14] and 2020 [1]. The details of the benchmarks are presented in Table I. All the experiments were done on an Ubuntu 20.04 running on an Intel Core i5 system with 3.3 GHz clock speed and 32 GB RAM. We used POLAR-Express (commit 13d42b0) and Aster (commit 2c991fb), downloaded from GitHub on 18 August 2023, and 20 March 2024, respectively. For the FPGA design synthesis, we employed Xilinx's Vitis HLS [13] (version 2023.2), downloaded on 22 February 2024.

We configured Aster with two settings and two error bounds: 1) A for error $1e-3$ and 2) B for $1e-5$. Setting A initialized the number of fractional bits to 20 (error: 2^{-20}) and setting B to 32 (error: 2^{-32}). Both the settings allowed a max of 32 bits, with ranges of fractional bits $[5, 32]$ for setting A and $[10, 32]$ for setting B. A run of Nexus was allocated a 5-hour time budget.

b) Safety Verification and Sound Code Generation: Table II summarizes all the experiments. As expected, more benchmarks (8 out of 12) are safe with the smaller error bound of $1e-5$ compared to $1e-3$ (7 out of 12). For larger benchmarks like Unicycle, ACC10, high over-approximations in reachability analysis prevented the safety verification. The largest benchmarks, Airplane and TORA, were found unsafe given our initial conditions, safety properties, and error bounds.

For benchmarks proven safe in the first phase, we attempted to generate implementations with the Nexus's code generation. Setting A, starting with a larger initial error (2^{-20}), is expected to report more infeasibility where generating a sound implementation that satisfies both the error bound and the max bit length is impossible. This was observed for ACC5. For ACC7, Nexus returns infeasible for both the errors due to increasing over-approximation errors with the increasing number of layers.

For benchmarks where we could generate implementations, we compiled them for a standard FPGA architecture using Xilinx's HLS and presented the latencies in the clock cycles that the compiler reported for the final hardware implementations. Naturally, the implementations with the error bound $1e-5$ have higher latencies due to the increased number of bits required to satisfy this smaller error bound compared to the error $1e-3$.

We also show the running times in Table II (columns 4 and 7). While time increases with the size of the networks, Nexus took a max of 20 s for the largest safe benchmark, ACC5.

c) Looped Code Generation: This section compares the Aster's code generation with Nexus's as shown in Table III. We refer to the code without parallelization directives as *serial* and the code with directives as *parallel*. We generated parallel versions of the Aster's unrolled and looped code and the serial version of the Nexus's nested-loop code. The benchmarks presented here are those for which we could generate implementations.

Our results show that using the directives in unrolled code does not improve the latencies w.r.t. the serial version due to instruction interdependencies (except for InvPend and DblPend). However, the compiler effectively identified parallelism in the looped and nested-looped versions, significantly reducing the latency compared to the serial versions. Moreover, as expected, unrolled serial implementations have lower latencies, but Nexus's nested-looped parallel implementations outperform them due to efficient parallelization, especially for the larger NNs.

TABLE III
COMPARING ASTER'S UNROLLED SERIAL AND LOOPED SERIAL IMPLEMENTATIONS WITH NEXUS'S OPTIMIZED IMPLEMENTATIONS (NESTED-LOOPED SERIAL AND ALL PARALLELS) FOR AUTOMATED PARALLELIZATION USING XILINX WITH ERROR $1e-5$ AND SETTING B

benchmarks	latencies of implementations (cycles)						design synthesis time (s)					
	unrolled		looped		nested-looped		unrolled		looped		nested-looped	
	serial	parallel	serial	parallel	serial	parallel	serial	parallel	serial	parallel	serial	parallel
InvPend	16	15	36	18	57	14	27.75	26.84	24.85	25.93	23.41	24.37
MountCar	30	30	92	38	106	25	43.37	43.41	28.33	34.81	24.43	31.32
SglPend	31	30	122	47	141	27	72.37	72.89	35.30	51.08	24.72	46.16
DblPendV1	32	31	135	50	89	27	77.90	77.34	35.30	51.22	25.46	44.34
DblPendV2	35	34	134	51	90	28	78.97	76.10	36.65	52.63	25.49	43.21
ACC3	58	58	162	65	158	39	94.06	94.64	38.54	59.29	26.26	49.91
ACC5	99	99	273	107	229	63	199.12	191.53	50.34	99.26	33.62	98.23

TABLE IV
COMPARING LATENCIES OF ASTER'S AND NEXUS'S IMPLEMENTATIONS FOR LARGER BENCHMARKS (✕: XILINX FAILS)

benchmarks	Aster (serial)		Nexus (parallel)	
	unrolled	looped	looped	nested-looped
Unicycle	29	864	265	18
Airplane	75	510	152	43
TORA	✕	604	186	41

We also compared the design synthesis times. Our results show that while the serial implementations have shorter synthesis times, the looped versions are faster than the unrolled ones due to the more compact representation of NNs.

To demonstrate the utility of the Nexus's code generation for the larger NNs, we used the second phase of Nexus to generate the code for the three largest unsafe benchmarks with the error $1e-3$ and setting B (setting A was inf due to the large initial error). Generating looped code is crucial for these benchmarks as serial implementations can become too large to compile (e.g., 62K lines of code for TORA). Table IV presents the results. It shows that the Nexus's parallel nested-looped code significantly outperforms the Aster in terms of latency for all the three benchmarks.

IV. RELATED WORK

The safety verification of NN-controlled CPSs has attracted significant attention in the recent years. Methods like [2] utilize the Taylor approximation of the NN controller and construct a flowpipe to compute the tight over-approximated reachable sets, while the others [4] convert the controller into an equivalent nonlinear hybrid system, then combine with the plant dynamics to verify the safety properties. Similarly, NN verification is done by approximating it with polynomial [3] or representing it as star sets [5] with approximated error bounds. In the principle, any of these methods could be used for the first phase of Nexus. However, these techniques assume real-valued arithmetic for their safety proofs and do not verify safety after quantization.

Alternatively, there is extensive literature on the quantization of NNs [6], [7], [10], [11], [12], but most techniques have been dynamically applied to the NN classifiers outside of safety-critical applications. Hence, they are fundamentally different from the sound mixed-precision tuning for the controllers, which is the goal of this letter.

The only work similar in flavour to ours is presented in [16]. Also, they extract a high-level model from the implementation and verify it, whereas we automatically generate the implementation that satisfies the properties of the high-level model.

V. CONCLUSION

This letter presents a novel, scalable integration of reachability analysis with sound quantization for the NN-controlled CPSs. Our generated implementations show optimization potential in terms of latency, especially for the custom hardware like FPGAs. While we currently focus on the feed-forward NN controllers with the ReLU and linear activations due to underlying tool constraints, the future holds promise for further enhancements to Nexus, including support for the nonlinear activations, hybrid controllers, and other networks like CNN.

REFERENCES

[1] T. T. Johnson et al., "ARCH-COMP$_{22}$ Category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants," in *Proc. ARCH*, 2020, pp. 142–184.

[2] Y. Wang et al., "POLAR-Express: Efficient and precise formal reachability analysis of neural-network controlled systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 3, pp. 994–1007, Mar. 2024

[3] S. Dutta, X. Chen, and S. Sankaranarayanan, "Reachability analysis for neural feedback systems using regressive polynomial rule inference," in *Proc. 22nd HSCC*, 2019, pp. 157–168.

[4] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: Verifying safety properties of hybrid systems with neural network controllers," in *Proc. HSCC*, 2019, pp. 169–178.

[5] H. Tran, F. Cai, M. L. Diego, P. Musau, T. T Johnson, and X. Koutsoukos. "Safety verification of cyber-physical systems with reinforcement learning control," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 5s, 2019, pp. 1–22.

[6] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd ICML*, 2015, pp. 1–10.

[7] S. Gopinath, N. Ghanathe, V. Seshadri, and R. Sharma, "Compiling KB-sized machine learning models to tiny IoT devices," in *Proc. PLDI*, 2019, pp. 79–95.

[8] A. Kumar, V. Seshadri, and R. Sharma, "Shiftry: RNN inference in 2KB of RAM," *ACM Program. Lang.*, vol. 4, Nov. 2020, pp. 1–30.

[9] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *Proc. ICSA*, 2018, pp. 688–698.

[10] Z. Song et al., "DRQ: Dynamic region-based quantization for deep neural network acceleration," in *Proc. ICSA*, 2020, pp. 1010–1021.

[11] H. Sharma et al., "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *Proc. ISCA*, 2018, pp. 764–775.

[12] D. Lohar, C. Jeangoudoux, A. Volkova, and E. Darulova, "Sound mixed fixed-point quantization of neural networks," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no.5s, pp. 1–26, 2023.

[13] "Vitis HLS." AMD. 2023. [Online]. Available: https://www.xilinx.com

[14] D. M. Lopez, P. Musau, H. Tran, and T. T. Johnson, "Verification of closed-loop systems with neural network controllers," *EPiC Ser. Comput.*, vol. 61, May 2019, pp. 201–210.

[15] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Proc. CAV*, 2013, pp. 258–263.

[16] J. Park, M. Pajic, O. Sokolsky, and I. Lee, "Automatic verification of finite precision implementations of linear controllers," in *Proc. 23rd Int. Conf. (TACAS)*, 2017, pp. 153–169.