Formal Verification of Deep Neural Networks in Hardware

1st Sincy Ann Saji
Department of NEX,
Intel India Pvt. Ltd)
Bangalore, India
sincy.ann.saji@intel.com

2nd Shreyansh Agrawal
Department of NEX,
Intel India Pvt. Ltd
Bangalore, India
shreyansh.agrawal@intel.com

3nrd Surinder Sood
Department of NEX,
Intel India Pvt. Ltd)
Bangalore, India
surinder.sood@intel.com

***Abstract*—Deep neural network (DNN) verification is an emerging field. The problem is more pronounced when they are modelled as hardware logic. Existing formal verification solutions for such hardware designs are fraught with issues like scalability and compute intensive resources. We solve this problem by creating a lightweight verification infrastructure and verifying the DNN logic by way of equivalence check. The light weight formal verification model is created by applying neural network simplification approaches discussed in the paper. The DNN formal model as a consequence is made simpler by removing the unwanted nodes, as a result it is capable of verifying intricate deep neural network designs effectively as compared with simulation based conventional strategies. The approach is applied on a design which is a car collision avoidance system (CARCAS) which is a logic deployed in autonomous car driving paradigm.**

***Index Terms*—Hardware Neural Network, Sequence equivalence check, Car Collision Avoidance System,**

I. Introduction

As design size and complexity continues to grow, the verification approach needs to evolve over time. IP reuse is an integral part of design methodology today; it helps to shorten the design cycle, but it also brings new verification challenges [1]. Verification methodology needs to adopt to these changes to ensure design is bug free and time to market goals are predictably met. Simulation based verification has been very successful for the last few decades, and continues to be effective at both block and SOC level, but it does have observability and controllability challenges [2]. Simulation is very efficient for verifying deep sequential behaviour, but it is not exhaustive. For mission critical blocks and some very specific applications, exhaustive analysis is a must [3]. Property based formal analysis provides both observability and controllability, and it also provides exhaustive analysis to ensure signoff quality verification [4]. Selecting the right block for formal verification is very important, it works best on control path intensive blocks where many concurrent events must be verified. Formal can also be used for specific data path verification problems wherein math heavy logic is written in RTL, and it needs to be functionally compared against reference C/C++ model. Formal verification has two main broad applications. One is model checking and second is equivalence checking [5].

We are considering artificial neural network design of car collision avoidance system which is being coded in hardware description language. The trained hardware neural network has 4 inputs and 5 outputs. It has 3 hidden layer and each hidden layer will have 6 neuron nodes. The idea is to verify the design using the sequential equivalence checking algorithm in a formal tool.

II. Fundamentals Of Formal Verification And Neural Networks

A. Why Formal Verification?

Formal verification is an automatic verification methodology that catches common bugs in the design and uncovers any design ambiguities [5].

In Formal verification the design correctness is verified using mathematical techniques. To verify the design, the formal verification tool uses algorithms and does not perform any timing checks. Formal verification on design can be performed during the start of the IC design cycle as there is no requirement of any stimulus or a testbench in the formal tool, hence, as soon as we have the RTL code of design available. The sooner bugs will be found, the easier it is to fix [6]

B. What is Formal Verification?

Formal verification tool verifies the design correctness by taking advantage of mathematical analysis and explores the design state space exhaustively. Formal verification and simulation-based verification are complimentary to each other. In simulation, the verification progress is tracked by extensive coverage metrics and it is mainly used to explore deep sequential behavior [7].

The correctness of the design can be checked by using formal based verification which requires user defined or auto generated properties. Formal explores the design state space exhaustively after starting from known reset state and. Figure 2.1 shows how complexity of the formal problem increases with sequential depth of the analysis because more and more register states track is to be kept [8].

C. Equivalence Checking:

Two RTL designs written in synthesizable system-verilog or VHDL can be compared using formal tool SEQ. Outputs

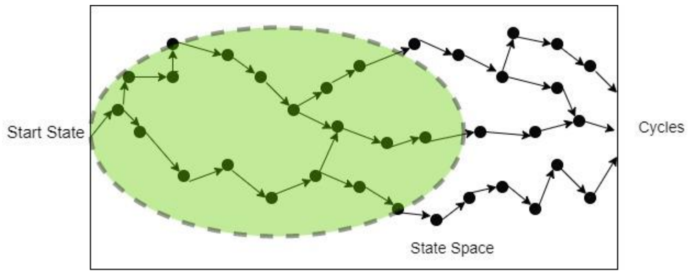

Fig. 1. Property Based Formal Verification [8]

identicality of two RTL designs can be checked by giving identical inputs on a cycle-by-cycle basis. One of the designs is termed as the implementation or impl and the other one as specification or spec. The post-modified and unverified impl design behaviour is verified against spec design behaviour using the6 formal tool SEQ as typically the spec design is known to be correct [11]. Figure 2.3 shows the functionality of equivalence check.

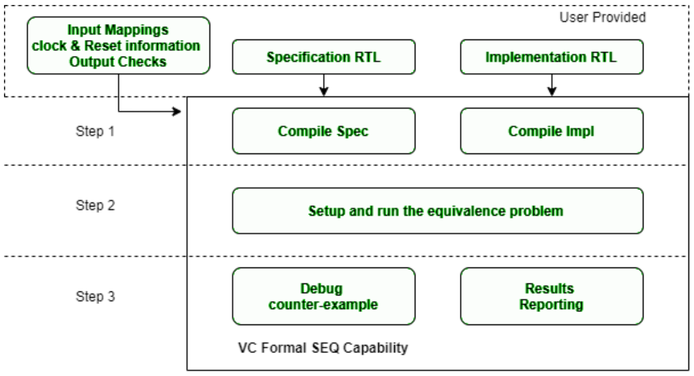

Fig. 2. SEQ Methodology Diagram [12]

D. Artificial Neural Network:

The behaviour of the human brain to process information and do analysis on data, can be simulated using a piece of computing system known as artificial neural networks (ANN). The problems which are difficult or sometimes impossible to solve by the human brain can be solved by these ANN's. They are also considered as the foundation of artificial intelligence systems. ANNs have the capability of self-learning which allows them to process more data variables and produce better results [13].

- An artificial neural network (ANN) is the component of artificial intelligence that is meant to simulate the functioning of a human brain.
- Processing units make up ANNs, which in turn consist of inputs and outputs. The inputs are what the ANN learns from to produce the desired output.
- Backpropagation is the set of learning rules used to guide artificial neural networks.
- The practical applications for ANNs are far and wide, encompassing finance, personal communication, industry, education, and so on.

E. Hardware Neural Network

In the last decade, neural network hardware has seen rapid development. Earlier we had Von-Neumann architecture, which had the sequential processors, whereas the artificial8 neural networks (ANNs) massively work on parallel-processing. Now we have large number of neural network models whose hardware units are designed to exploit the perks of parallel processing. although we have witnessed the tremendous growth of digital computing-power of GPU's (general-purpose processors), In the specialized applications the neural network hardware are showing some promising results, in the application of image processing units, pattern-recognition, speech synthesis and analysis, high energy physics etc [14].

F. Hardware versus Software

For the development of ANNs simulation environment on sequential machines, a good amount of work was done. Conventional Von-Neuman processors performance, like the Intel-pentium series, is continuously improving drastically. When fastness is not the criteria for a certain task, many neural network designers do the implementation using software without any special hardware [14]. Although real time responses cannot be provided even by the fastest sequential processor for large networks which have huge numbers of neurons and synapses. On the other side, speed can be increased tremendously by parallel processing with multiple simple processing elements (PEs). There are some special applications like speech recognition devices and silicon retinas for which hardware neural networks are required [15].

III. Implementation Of CARCAS Neural Network

The CARCAS model is referred to as a car collision avoidance system. We are considering a design which takes four inputs and produces five outputs. This neural network design can control the movement of the car based on the input received by its input sensors. The diagram of automatic car which is having sensors installed is shown in Figure 4.1, There are four sensors installed in our CARCAS model which provides the input to the neural network

The trained neural network design of the above model will have 3 hidden layers and each layer will have 6 nodes. The input layer has 4 inputs and output layer will have 5 output nodes. The path to each node will have weight and bias values. These weight and bias values are generated after training the model.

A. Implementing the Neural Network RTL

Steps to implement the RTL of neural network design which is given in Figure 4.4:

- The neural network design has 4 input variables which take real values as input.But to feed this design to a

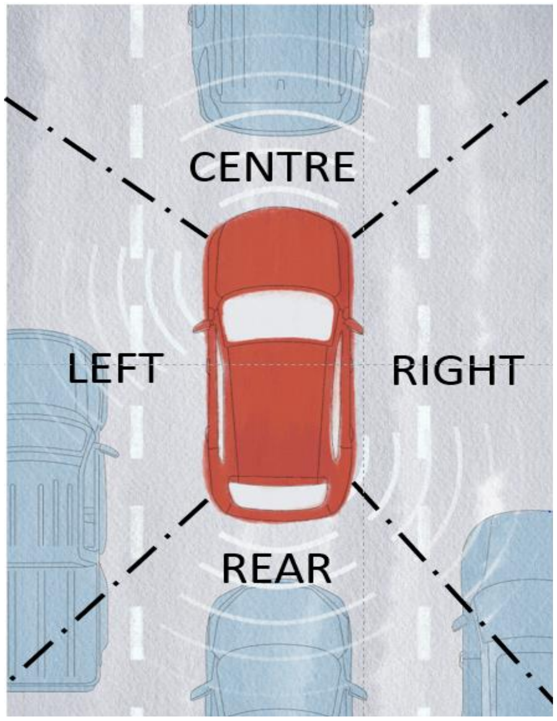

Fig. 3. Position of Sensors in Practical Model [24]

formal tool we need to convert these input real valuesinto IEEE 32-bit notation.

- Similarly all the weight and bias values are converted to 32-bit notation.
- To implement the sigmoid function, which has the exponential function. we need to linearly approximate the exponential function. So that equation has only linear arithmetic functions.
- Formal tool black boxes the behavioural model during verification. So we need to implement the floating-point unit for performing arithmetic operations.
- Implementing hidden neurons and output neurons as separate modules.
- Instantiate the modules to do the interconnection between modules.
- Whole design is synthesizable.

IV. Formal Verification Of Hardware Neural Network Design

A. *Simplifying Hardware Neural Network by NN-simplify algorithm:*

In order to verify hardware neural structures implemented in HDL language by formal methods. We need to simplify the design of HNN. To reduce the complexity of such designs we are applying the NN Simplify algorithm to identify the redundant nodes and paths. We are considering that the removal of redundant nodes and paths does not affect the overall functionality of the design. Figure 5.1 shows the overall flow.

The implemented algorithm identifies the redundancy in following steps:

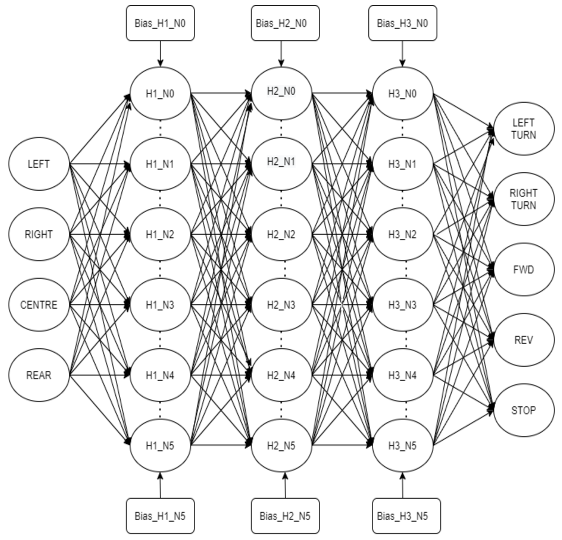

Fig. 4. CARCAS Neural Network internal structure

- Firstly we perform random simulation on the NN verification model, marking hidden nodes as candidates for removal.
- We check if the node values is greater than 0.01.
- If the above condition holds true, then we remove that node from the candidate of removal list. Otherwise we feed back the design for running simulation again.
- Once we have removed the redundant node from the design RTL, we feed the design to check the redundant path in the design.
- We identify the negative value of weight, coming as an input to a particular node.
- In the next step, we identify the same in magnitude but opposite in sign weight value which can cancel the effect of negative weight value in the calculation of output of that node.
- RTL is updated after removing the redundant nodes and path and fed to the verification model.

B. *Verification Model*

After simplifying the neural network design by removing the redundant nodes and reducing the number of redundant paths. The verification model is updated with reduced paths and nodes. This model is less complex and needs to do less calculation as the reduction in number of nodes and paths will lead to reduction in test stimulus. The design with reduced design logic and data paths, which is producing the similar output as the DUT or output with negligible error. can be considered a good verification model.

C. *Verifying Design by SEQ (Sequential Equivalence Check):*

- In the formal tool, actual design is fed as DUT and the simplified design is fed as the verification mode.
- SEQ is performed over both designs to compare the output values.

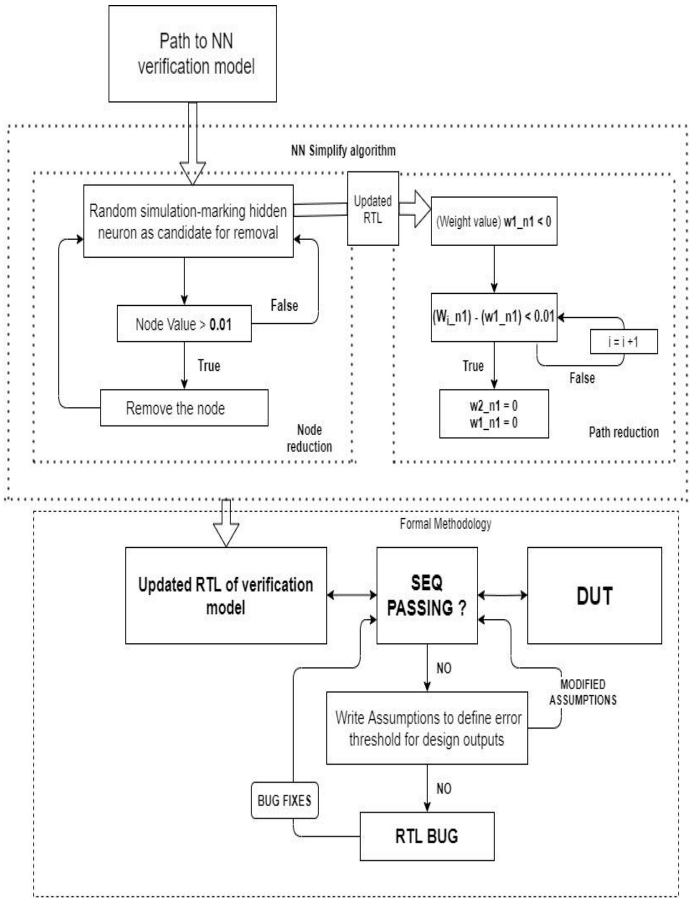

Fig. 5. Proposed Verification Methodology Diagram

- If the SEQ is passing, then we can say that design is verified.
- If it is not passing, then we have to write assumptions to define error thresholds for design output.
- If assumptions with max assumed threshold are also failing then there could be a RTL bug.
- We need to fix the RTL bug, if there is any and the process should be repeated again.

V. Results Summary

A. Initial Results

Following are the results that we got after doing the first round of formal verification:

- In the first iteration we are clearly getting an output mismatch and all the output sequences are failing. which we can see in the given figure. This failure is due to the fact that, even if there is any slight difference in the output of the DUT and verification model, we get an equivalence check failure.
- The output of both the design i.e. The DUT and verification model is given in the following diagrams. It can be observed that the output xout1 which is supposed have logic one has the value greater than 0.1. And the rest of the outputs xout2, xout3, xout4 and xout5 which supposed to have logic low are producing output less than 0.1.

TABLE I
Output values after running SEQ

	SEQ Output	
Output	***impl output***	***spec output***
xout1	1.6248e-01	2.80382e-01
xout2	2.43867e-04	6.31146e-04
xout3	4.12743e-02	2.10696e-02
xout4	2.16332e-03	2.92424e-03
xout5	1.74903e-02	2.03741e-02

B. Final Results:

The outputs that we got after the initial results are not exactly the same. So we need to make following assumptions on outputs:

- If the output value is greater than 0.1 we assign it value "1" (high logic).
- If the output value is less than 0.1 we assign the value "0" (low logic).

On writing the above assumptions, we got our SEQ checks passing

TABLE II
Property Status

Serial No.	**Output properties**	
	Property Status	***property***
1.	failing	xout1
2.	failing	xout2
3.	failing	xout3
4.	failing	xout4
5.	failing	xout5

C. Summary:

- With increasing design complexity large no. of input stimulus, the timing and effort to verify the design also Increases exponentially in simulation formal property verification environment.
- Whereas proposed verification approach is almost independent of these factors. Table-III gives the comparison among different verification techniques.

VI. Conclusion And Future Scope

The main goal of this work to develop and implement the formal verification methodology and verify the hardware neural network designs which can be implemented by using any hardware description language was completed. Results showed that the reduced and less complex verification model was performing almost like our DUT, from which we can conclude that if design with lesser number of neuron nodes and paths is bug free than the actual DUT will also be bug free. The increasing design size and complexity of hardware neural networks demands faster and simpler ways of verification. Therefore the future work will be focused on improving

TABLE III
COMPARISON AMONG DIFFERENT VERIFICATION TECHNIQUE

Input Stimulus Value	*Simulation Infrastructure*	*Layer by Layer Formal Property Verification*	*SEC with Simplified Verification Mode (Proposed Methodology)*
Either 0 or 1	1) Need to code 16 testcase scenarios 2) Require min 10-12 hours to simulate	1) End to end properties =16 2) Total Inner Layer properties = 16*6 + 36*6 +36*6 + 36*5 = 708 3) Require min 6-8 hours to write all the properties	1) Only need to specify input stimulus Values in tcl file. 2) Require less than 10min to verify the design
Real Values between 0 and 1	1) For total number of n possible values, total testcase scenarios = 4^n	1) End to end properties = 4^n 2) Total inner layer properties = $4^n * 6 + 6^n * 6 + 6^n * 6 + 6^n * 5 = N$	1) Only need to specify input stimulus vlaues in tcl file.

the nn-simplify algorithm for such designs and developing a verification model which is much more less complex and gives the more accurate outputs so that the output node deltas between the DUT and verification model can be minimized.

REFERENCES

[1] S. Kotha, R. Ravimony, N. Mohankumar, "Automated UVM Based Verification of Device Life Cycle Management IP", Intelligent Computing, Information and Control Systems (ICICCS 2019), vol. 1039, pp. 574-583, 2019.

[2] K. K. Yadu and R. Bhakthavatchalu, "Block Level SOC Verification Using System verilog", 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), pp. 878-887, 2019.

[3] Z. Patel, S. Gupta, Y. B. Kumar, "Reusability and Scalability of an SOC Testbench in Mixed-Signal Verification—The Inevitable Necessity", Advances in VLSI and Embedded Systems. Lecture Notes in Electrical Engineering, vol. 676, pp. 1-16, 2021.

[4] K. Hofer-Schmitz and B. Stojanović, "Towards formal verification of IoT protocols", A Review,Computer Networks, vol. 174, pp. 1-6, 2020.

[5] A. Souri, A.M. Rahmani, N.J. Navimipour, "A symbolic model checking approach in formal verification of distributed systems", Human-centric Computing and Information Sciences, vol. 9, pp. 1-27, 2019.

[6] J. He, X. Guo, T. Meade, R. Gautam Dutta, Y. Zhao, Y. Jin, "SOC interconnection protection through formal verification Integration", vol. 64, pp. 143-151, 2019.

[7] https://www.techdesignforums.com/practice/guides/formal-verification.

[8] https://semiengineering.com/knowledge-centers/eda-design/verification/formalverification.

[9] https://www.synopsys.com/verification/static-and-formal-verification.html

[10] V. N. Possani, A. Mishchenko, R. P. Ribas and A. I. Reis, "Parallel Combinational Equivalence Checking", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 10, pp. 3081-3092, 2020.

[11] J. Hu, G. Wang, G. Chen and X. Wei, "Equivalence Checking of Scheduling in High-Level Synthesis Using Deep State Sequences," in IEEE Access, vol. 7, pp. 183435-183443, 2019.

[12] https://www.synopsys.com/glossary/what-is-equivalence-checking.html

[13] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress", Neurocomputing, vol. 74, no. 1–3, pp. 239-255, 2020.

[14] J. Clements and Y. Lao, "Hardware Trojan Design on Neural Networks," IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5, 2019.

[15] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, N.I. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation", Mathematics and Computers in Simulation, vol. 177, pp. 232-243, 2020.

[16] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, N.I. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation", Mathematics and Computers in Simulation, vol. 177, pp. 232-243, 2020.

[17] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, N.I. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation", Mathematics and Computers in Simulation, vol. 177, pp. 232-243, 2020.

[18] L. Kuper, G. Katz, J. Gottschlich, K. Julian, C. Barrett, M. Kochenderfer, "Toward scalable verification for safety-critical deep networks", Accepted for presentation at SysML, pp. 1-3, 2018.

[19] S. Han, H. Mao, W. Dally, "Deep compression, compressing deep neural networks with pruning, trained quantization and Huffman coding", published as a conference paper at ICLR, pp. 1-14, 2016.

[20] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer: SqueezeNet, "AlexNet-level accuracy with 50x fewer parameters and ¡0.5 MB model size", pp. 10-23, 2016.

[21] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, A. Criminisi: "Measuring neural net robustness with constraints", Proceedings of 30th Conference on Neural Information Processing Systems (NIPS), vol. 29, pp. 2613- 2621, 2016.

[22] https://towardsdatascience.com/implementing-the-xor-gate-usingBackpropagation-in-neural-networks-c1f255b4f20d.

[23] S. Sood, S. Agrawal, "Formal Verification of Hardware Neural Network Design using Formal tool", presented in Snug South Asia-2021 conference, 2021.

[24] https://sitn.hms.harvard.edu/flash/2017/self-driving-cars-technology-riskspossibilities/

[25] https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/

[26] K. Mori, H. Fukui, T. Murase, T. Hirakawa, T. Yamashita and H. Fujiyoshi, "Visual Explanation by Attention Branch Network for End-to-end Learning-based Self-driving," 2019 IEEE Intelligent Vehicles Symposium (IV), pp. 1577-1582, 2019.

[27] K. Mori, H. Fukui, T. Murase, T. Hirakawa, T. Yamashita and H. Fujiyoshi, "Visual Explanation by Attention Branch Network for End-to-end Learning-based Self-driving," 2019 IEEE Intelligent Vehicles Symposium (IV), pp. 1577-1582, 2019.

[28] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks", Proceedings of 15th International Symposium on Automated Technology for Verification and Analysis (ATVA), pp. 269–286, 2017.

[29] K. Julian, M. Kochenderfer, M. Owen, "Deep neural network compression for aircraft collision avoidance systems", Journal of Guidance, Control, and Dynamics, vol. 3, pp. 598–608, 2019.

[30] G. Katz, C. Barrett, D. Dill, K. Julian, M. Kochenderfer, "Reluplex: an efficient SMT solver for verifying deep neural networks". Proceedings of 29th International Conference on Computer Aided Verification (CAV), pp. 97–117, 2017.

[31] X. Sun, H. Khedr, Y. Shoukry, "Formal verification of neural network controlled autonomous systems", Proceedings of 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC), pp. 147–156, 2019.

[32] X. Sun, H. Khedr, Y. Shoukry, "Formal verification of neural network controlled autonomous systems", Proceedings of 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC), pp. 147–156, 2019.

[33] B. Oliver, and M. Zamani, "A framework for formal verification of behaviour trees with linear temporal logic", in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 2341-2348, 2020.