Hybrid Convolutional Neural Networks with Reliability Guarantee

Hans Dermot Doran
Institute of Embedded Systems
Zurich University of Applied Sciences
Winterthur, Switzerland
donn@zhaw.ch

Suzana Veljanovska
Institute of Embedded Systems
Zurich University of Applied Sciences
Winterthur, Switzerland
veln@zhaw.ch

***Abstract*— Making AI safe and dependable requires the generation of dependable models and dependable execution of those models. We propose redundant execution as a well-known technique that can be used to ensure reliable execution of the AI model. This generic technique will extend the application scope of AI-accelerators that do not feature well-documented safety or dependability properties. Typical redundancy techniques incur at least double or triple the computational expense of the original. We adopt a co-design approach, integrating reliable model execution with non-reliable execution, focusing that additional computational expense only where it is strictly necessary. We describe the design, implementation and some preliminary results of a hybrid CNN.**

Keywords—Reliability, convolutional neural networks, functional safety, redundant execution architectures, artificial intelligence, accelerators

I. Introduction

The association of AI with a "black box" implies dependability and safety concerns. Both the model generation and the inference stages lack explainability which directly affects the applicability of this technology in safety and trust domains. There is a substantial room for improvement in this area to meet established safety standards such as the IEC 61508 [1], ISO 26262 [2] and IEC 62061 [3]. There are efforts under way to integrate AI into these safety standards [4]. On a model level the efforts to apply AI in safety-related applications focuses on hybrid schemes where AI classifiers act alongside more traditional AI approaches such as knowledge-based systems [5]. Outputs of AI classifiers can be caged on a training [6], on a model [7], or on an operational basis [8]. Methods to ensure correct execution, on various platforms, of an AI algorithm regardless of any model considerations are well summarized, along with fault modes, in [9].

Our novelty is to propose a hybrid, mixed-criticality, convolutional neural network (CNN) that ensures reliable execution of specific layers [10]. Our execution targets are edge-AI devices. We implement checkpointing and rollback at an operation-execution level to retain the value of previous executions and explicitly report persistent failures. Our proposal is generalized for execution on any computing platform, initial tests have been performed on a CPU.

We thank SERI, Innosuisse and the ZHAW for the financial support of this research under the umbrella of the KDT project REBECCA, grant agreement n° 101097224.

II. Techniques for Reliable Execution

In the context of reliability engineering, the primary objective is to ensure accurate and timely execution of calculations. When errors do arise, they should be promptly identified and addressed to prevent their propagation throughout the system. One well-understood reliability approach involves lockstep processing, where two microprocessors run in synchronization, executing the same operations in parallel or within a few clock cycles and comparing the results [11], [12]. Other methods include redundant processing, as error detection necessitates redundancy in execution. This redundancy can be achieved either spatially, by execution on two separate units, or temporally, by execution twice in series on the same unit.

Additional methods suggest checkpointing which involves saving the current state of a system at predetermined points to allow for potential rollbacks or recovery in case of failure [13]. Since a redundant execution of a whole CNN with checkpointing is computationally expensive we reduce the rollback distance to single operation considering that convolutional layers are consisted of multiplications and additions. For example, a multiplication redundantly executed with result comparison (checkpoint) and a re-multiplication (rollback) in case of failure [14]. We subsequently adopt a hybrid approach to reduce computational costs. This involves utilizing the CNN's properties as a preprocessing step, providing input for a surrogate function [6] and reliably executing (using checkpoints and rollbacks) only specific operations of predetermined convolutional layers.

III. Hybrid (Convolutional) Neural Network Design

CNN layers can be interpreted as mapping functions and adding reliability features in each layer increases computational expense. In practice not all layers must be executed reliably and therefore we can partition a CNN for reliable and non-reliable execution. Our experimental use-case is shape recognition, to facilitate reproducibility we use traffic signs and focus on the “Stop” sign. With adequate training a CNN classifier will be able to recognise this sign. A shape (octagon) recognition algorithm will be able to qualify this classification. In Figure 1 we include this shape recognition unit alongside a CNN to achieve a qualified classification. The qualifier’s output signal will qualify a relevant output of the CNN. Only subset of classes will require qualification. Other more general

DOI 10.1109/DSN-W60302.2024.00023

classifications not considered safety-critical need not be qualified. As a pre-processing step for the qualifier, edge detection is performed in the CNN which means the first layer of the CNN is executed reliably (DCNN - Figure 1). The qualifier unit implements a surrogate function with predefined upper and lower bounds which yield deterministic outputs that are entirely explainable. The surrogate function is based on converting the image into a time series through distance centroid computation followed by Symbolic Approximation (SAX) [15], which efficiently converts time-series data into a string format to perform inexpensive comparisons with other strings.

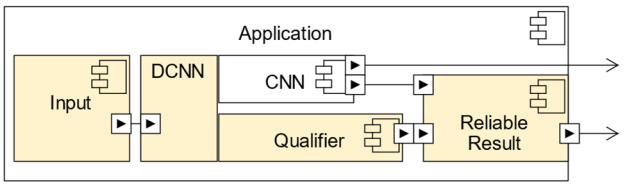

Figure 1: A hybrid CNN featuring reliable (DCNN) and non-reliable (CNN) execution.

We substantiate the concept of a hybrid CNN by taking image sizes with reasonably defined edges. We utilize AlexNet [16] trained on German traffic sign recognition benchmark (GTSRB) [17] as it uses images dimensions that are large enough (227*227*3) to be useful for shape recognition. The first convolutional layer of the AlexNet consists of 96 11*11*3 filters. In the search for a pragmatic integration of qualifier functions and the CNN without delving into work-flow complications, we simply replace the first (machine trained) filter with a Sobel-x, Sobel-y, Sobel-x kernel. The CNN classification accuracy remains the same albeit the confusion matrices change slightly. The classification accuracy results measured after replacement of all the 96 filters, one at a time, with this Sobel filter are shown in [10].

IV. Hybrid Network Implementation

Our eventual aim is to propose implementations of this hybrid CNN suitable for FPGA devices. Our FPGA workflow includes designing hardware components in VHDL and simulating them in GHDL [18], integrated with cocotb [19], an open verification methodology (OVM) supporting verification tool in Python. This gives us the flexibility to utilize Python to construct various versions of the algorithm seamlessly across the two computing platforms, a CPU and, later, an FPGA. We implement trivial basic operations (multiplication and addition) in both redundant (Algorithm 2) and non-redundant (Algorithm 1) versions. In both, to conserve consistency with the calling algorithm, we return the value together with a qualifier indicating whether the operation was carried out correctly. Algorithm 3 performs a convolution operation assuming that every operation fails unless asserted (by the qualifier) otherwise. The implementation executes an overloaded multiplication and addition and verifies for any errors. In case of an error during operation execution [20], an error counter is incremented by a specified value (factor, line 12), which is then compared against a threshold (defined in line 12). For each correct operation, this error counter is decremented by one, stopping at zero (lines 18-19). First performance results are available in [10].

Single Multiplication Operation

```
1:  start procedure
2:      result ← k · i                  → compute one calculation
3:      qualifier = True                → set the flag to True
4:      return qualifier, result        → return the flag and the result from the multiplication
5:  end procedure
```

Algorithm 1: Example of non-reliably executed multiplication.

Redundant Multiplication Operation

```
1:  start procedure
2:      result_1 ← k · i                → compute one product
3:      result_2 ← k · i                → repeat the calculation
4:      qualifier ← False               → set qualifier to false
5:      sub_1 ← result_1 - result_2     → compare the multiplications
6:      sub_2 ← result_1 - result_2
7:      if sub_1 = sub_2 and sub_1 = 0 then   → if the subtractions are 0, set the qualifier as True
8:          qualifier = True
9:      end if
10:     return qualifier, result_1      → return the flag and the result from the multiplication
11: end procedure
```

Algorithm 2: Example of a reliably executed multiplication.

Algorithm 1 Reliable Convolution Kernel

```
1:  error ← 0                           → initialization of global variables
2:  factor ← 8, n ← factor*2
3:  failure ← True
4:  start procedure
5:      failure ← True                  → initialization of local variables
6:      sum ← 0, result ← 0, temp ← 0
7:      for i in kernel height do
8:          for j in kernel width do
9:              qualifier ← False
10:             qualifier, product ← k_{i,j} * i_{i,j}      → execute multiplication
11:             if qualifier = False then                   → on error
12:                 if error ← error + factor > n then      → exit if error ceiling reached
13:                     result ← 0, break
14:                 qualifier, product ← k_{i,j} * i_{i,j}  → re-execute multiplication
15:                 if qualifier = False then
16:                     if error ← error + factor > n then
17:                         result ← 0, break
18:                 if error > 0 then                       → decrement error
19:                     error ← error – 1
20:             qualifier ← False
21:             qualifier, temp ← sum + product             → execute addition
22:             if qualifier = False then                   → on error
23:                 if error ← error + factor > n then      → exit if error ceiling reached
24:                     result ← 0, break
25:                 qualifier, temp ← sum + product         → re-execute addition
26:                 if qualifier = False then
27:                     if error ← error + factor > n then  → exit if error ceiling reached
28:                         result ← 0, break
29:                 if error > 0 then                       → decrement error
30:                     error ← error – 1
31:             sum ← temp, temp ← 0
32:     result ← sum
33:     failure ← False
34: end procedure
```

Algorithm 3: Example for a reliably executed convolution kernel.

V. Conclusion

We have proposed a hybrid CNN architecture for reliable/mixed-criticality execution of a classification model by reliably executing single features of the model instead of the entire CNN. As we maintain that only single classifier classes of a CNN exhibit safety relevance, we can qualify those classes with a reliably executed qualifier, in our case using SAX as a surrogate function for shape detection. The merging of reliable execution of some layer subsection facilitates minimization of computational expense and substantiates the concept of a hybrid CNN. We also provide checkpointing and rollback and reporting should a permanent error occur.

References

[1] IEC, '61508-1:2010: Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements (see Functional Safety and IEC 61508)'. 2010.

[2] ISO, 'ISO 26262-1:2018 Road vehicles Functional safety'. 2018.

[3] IEC, 'IEC 62061:2021 Safety of machinery - Functional safety of safety-related control systems'. 2021.

[4] IEC, 'IEC and ISO launch working group to advance functional safety of AI systems'. Accessed: Jan. 29, 2024. [Online]. Available: https://www.iec.ch/blog/iec-and-iso-launch-working-group-advance-functional-safety-ai-systems

[5] J. Wörmann *et al.*, 'Knowledge Augmented Machine Learning with Applications in Autonomous Driving: A Survey'. arXiv, Nov. 20, 2023. doi: 10.48550/arXiv.2205.04712.

[6] A. Gauffriau, F. Malgouyres, and M. Ducoffe, 'Overestimation Learning with Guarantees', *Proceedings of the Workshop on Artificial Intelligence Safety 2021 (SafeAI 2021)*, Jan. 2021.

[7] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, 'AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation', in *2018 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA: IEEE, May 2018, pp. 3–18. doi: 10.1109/SP.2018.00058.

[8] F. Geissler *et al.*, 'Towards a Safety Case for Hardware Fault Tolerance in Convolutional Neural Networks Using Activation Range Supervision', *Proceedings of the Workshop on Artificial Intelligence Safety 2021 (SafeAI 2021)*, Aug. 2021.

[9] P. Rech, 'Artificial Neural Networks for Space and Safety-Critical Applications: Reliability Issues and Potential Solutions', *IEEE Transactions on Nuclear Science*, vol. 71, no. 4, pp. 377–404, Apr. 2024, doi: 10.1109/TNS.2024.3349956.

[10] H. D. Doran and S. Veljanovska, 'Hybrid Convolutional Neural Networks with Reliability Guarantee'. arXiv, May 08, 2024. doi: 10.48550/arXiv.2405.05146.

[11] C. Hernandez and J. Abella, 'Timely Error Detection for Effective Recovery in Light-Lockstep Automotive Systems', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 11, pp. 1718–1729, Nov. 2015, doi: 10.1109/TCAD.2015.2434958.

[12] H. D. Doran and T. Lang, 'Dynamic Lockstep Processors for Applications with Functional Safety Relevance', in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2021, pp. 1–4. doi: 10.1109/ETFA45728.2021.9613543.

[13] L. Pullum, 'Software Fault Tolerance Techniques and Implementation | Artech books | IEEE Xplore'. Accessed: Mar. 27, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/9100486

[14] A. Mahmoud *et al.*, 'HarDNN: Feature Map Vulnerability Evaluation in CNNs'. arXiv, Feb. 25, 2020. doi: 10.48550/arXiv.2002.09786.

[15] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, 'A symbolic representation of time series, with implications for streaming algorithms', in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, in DMKD '03. New York, NY, USA: Association for Computing Machinery, Jun. 2003, pp. 2–11. doi: 10.1145/882082.882086.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, 'ImageNet classification with deep convolutional neural networks', *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[17] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, 'The German Traffic Sign Recognition Benchmark: A multi-class classification competition', in *The 2011 International Joint Conference on Neural Networks*, Jul. 2011, pp. 1453–1460. doi: 10.1109/IJCNN.2011.6033395.

[18] 'GHDL Main/Home Page'. Accessed: Apr. 02, 2024. [Online]. Available: http://ghdl.free.fr/

[19] 'cocotb', cocotb. Accessed: Apr. 02, 2024. [Online]. Available: https://www.cocotb.org/

[20] M. Adams, J. Coplien, R. Gamoke, F. Keeve, K. Nicodemus, and R. Hanmer, 'Fault-tolerant telecommunication system patterns', in *The patterns handbooks: techniques, strategies, and applications*, USA: Cambridge University Press, 1998, pp. 189–202.