NeuroCodeBench: a plain C neural network benchmark for software verification

Edoardo Manino[1], Rafael Sá Menezes[1,2], Fedor Shmarov[1], Lucas C. Cordeiro[1,2]

[1]The University of Manchester, Manchester, United Kingdom
[2]Universidade Federal do Amazonas, Manaus, Brazil

Abstract

Safety-critical systems with neural network components require strong guarantees. While existing neural network verification techniques have shown great progress towards this goal, they cannot prove the absence of software faults in the network implementation. This paper presents *NeuroCodeBench* – a verification benchmark for neural network code written in plain C. It contains 32 neural networks with 607 safety properties divided into 6 categories: maths library, activation functions, error-correcting networks, transfer function approximation, probability density estimation and reinforcement learning. Our preliminary evaluation shows that state-of-the-art software verifiers struggle to provide correct verdicts, due to their incomplete support of the standard C mathematical library and the complexity of larger neural networks.

1 Introduction

In contrast to classic software development, neural networks are crafted via a long process of trial and error that terminates when their predictive performance reaches a satisfactory level [14, 45]. The iterative and performance-driven nature of this process leaves neural networks vulnerable on many fronts [26]: poor performance on out-of-distribution [21] and adversarial inputs [39], misspecification of the neural architecture and training process [27], invocation of broken and deprecated libraries [37], outright software bugs [23]. Unfortunately, many of these vulnerabilities are not easy to catch early in the development process and may remain hidden until after deployment.

The most successful techniques for guaranteeing the functional correctness of neural networks operate at a high level of abstraction, where finite precision and other implementation details are not considered [30, 46, 38]. Although efforts to debug the actual implementation of neural networks exist, they are based on automatic testing and thus cannot prove correctness for all inputs [41, 23, 20]. This lack of guarantees is especially concerning for safety-critical systems since common software vulnerabilities [1] (e.g., arithmetic overflows, invalid memory accesses) can make the networks produce wrong results, expose sensitive data or corrupt the system they are executed on.

While off-the-shelf software verifiers can be used to check neural network code [44, 35], there has been no systematic attempt at assessing their performance on such tasks. Typically, state-of-the-art verification tools (e.g., CPAChecker [12], ESBMC [22], CBMC [31], UAutomizer [24]) are compared on SV-COMP [11] - the largest software verification competition with over 15'000 C programs ranging from hand-crafted code to real-world software (e.g., drivers, Linux kernel). However, this

Benchmark Category	Safe	Unsafe	Ground Truth
`math_functions`	33	11	A Priori
`activation_functions`	40	16	A Priori
`hopfield_nets`	47	33	A Priori
`poly_approx`	48	48	Brute Force
`reach_prob_density`	22	13	VNN-COMP'22
`reinforcement_learning`	103	193	VNN-COMP'22

Table 1: Overview of *NeuroCodeBench*. The "Unsafe" column comprises all properties for which a counterexample exists. The "Ground Truth" column reports the source of our verdicts.

competition lacks a dedicated benchmark for either neural networks or mathematical libraries (e.g., `math.h`).

This paper presents *NeuroCodeBench* – a reasoned benchmark of neural network code in plain C. It is designed to exercise the capabilities of existing software verifiers without overwhelming them with excessively large instances. More specifically, it contains 32 neural networks with 607 safety properties in SV-COMP format divided into the following 6 categories: maths library, activation functions, error-correcting networks, transfer function approximation, probability density estimation and reinforcement learning. The last two categories are converted to C code from the VNN-COMP'22 suite [38], whereas the rest are entirely new. As a demonstration, we run the leading tools of SV-COMP 2023 in reachability, falsification and floating point arithmetic [11]. Our preliminary results show that these verifiers have incomplete support of the `math.h` library and struggle on larger neural networks. Lastly, we make *NeuroCodeBench* publicly available at [3] and [33].

2 The Benchmark

2.1 Design Requirements

In designing *NeuroCodeBench*, we target two main requirements. First, our benchmark must be representative of existing neural network code. Mainstream libraries like PyTorch [6] and Tensorflow [7] have an opaque multi-language interpreted structure that can be easily tested [23, 20], but does not lend itself to automated software verification. For this reason, we opt for micro-controller frameworks, where the source code of the network is fully available. We use two existing tools for converting high-level neural network specifications to standalone C code: `onnx2c`[4] and `keras2c`[2, 18].

Second, our benchmark must contain safety properties whose verdict is known, with reasonably balanced sets of safe and unsafe verdicts. Existing works rely on the verdicts of a single tool [44, 35] and thus are not a reliable source of information. Here, we establish the ground-truth verdict of our 607 safety properties in three ways (see Table 1): *A Priori* verdicts come from the specific mathematical structure of the functions and networks we verify; *Brute Force* verdicts come from exhaustive exploration of all possible floating point inputs; *VNN-COMP'22* verdicts come from the independently-run neural network verification competition [38]. For the latter, we only keep unsafe properties if we can reproduce the corresponding counterexamples.

2.2 Benchmark Description

Math Library. Typically, neural networks rely on 32-bit floating point operations[1] and invoke the corresponding functions in the `math.h` library. More specifically, most activation functions depend on

[1]We leave quantised and binarised neural network benchmarks to future work.

Neural Network Category	Inputs	Outputs	Layers	Neurons	Activations	Architecture	Conversion
`hopfield_nets`	4–64	4–64	1	4–64	Softsign/TanH	Recurrent	`keras2c`
`poly_approx`	1	1	1–4	16–1024	ReLU	Feedforward	`keras2c`
`reach_prob_density`	3–14	3–14	2–3	64–192	ReLU	Feedforward	`onnx2c`
`reinforcement_learning`	4–8	2–8	2	128–512	ReLU	Feedforward	`onnx2c`

Table 2: Neural networks in *NeuroCodeBench*. The "Layers" and "Neurons" columns refer to the hidden layers only. The networks in `hopfield_nets` have a number of iterations between 1 and 4.

exponential, logarithm, error function, absolute value, and max function (see `activation_functions` category). Similarly, positional encodings depend on sine and cosine [32], while Euclidean distances and vector normalisation depend on the square root [15].

In this light, it is worth checking whether software verifiers correctly handle calls to `math.h`. We write benchmarks that depend on the following functions: `acosf`, `asinf`, `cosf`, `erff`, `expf`, `fabsf`, `logf`, `sinf` and `sqrtf`. Since their semantics are platform-specific, we assume compliance with the IEEE 754 standard for 32-bit floating point [28] and the C99 standard for `math.h` [29]. We provide 44 safety properties (see Table 1) that check for a wide range of behavior: output bounds, monotonicity, periodicity, symmetry, function inversion and derivatives.

Activation Functions. Most of the non-linear behaviour in neural networks is concentrated in the activation layers [15]. These contain fairly restricted sets of activation functions whose implementation should be verified for correctness. Our benchmark includes the most popular ones [40, 25]: Elu, Gelu, Logistic, ReLU, Softmax, Softplus, Softsign and TanH. In turn, their definition depends on the functions `erff`, `expf`, `expm1f`, `fabsf`, `fmaxf`, `log1pf` and `tanhf`. While most activation functions are univariate, the Softmax accepts multivariate inputs. To keep our verification instances small, we limit the size of Softmax input vectors to 2 and 4.

Error-Correcting Networks. For a long time, it has been known that certain types of recurrent neural networks can act as error-correcting decoders [9, 17]. The main idea is encoding a sequence of d bits into a vector $x \in \{\pm 1\}^d$, and letting the neural network flip the sign of the incorrect entries.

Here, we choose Hopfield networks with Hebbian weights since their properties are well understood [9]. Specifically, we build networks reconstructing a single pattern $x = (1, \ldots, 1)$. We vary the pattern length in $d \in \{4, 8, 16, 32, 64\}$ and the number of recursions in $t \in [1, 4]$. For compatibility with `keras2c` [2, 18], we use Softsign and TanH activations (see Table 2) rather than traditional sign activations [9]. Our safety properties check whether the network can reconstruct x when the first $d/2 - 1$ entries can take any value in $[-1, 1]$. Due to the network symmetry, we establish the ground truth by checking the extreme inputs x and $x' = (-1, \ldots, 1)$, where $x'_i = -1$ for all $i \in [1, d/2 - 1]$.

Transfer Function Networks In several engineering areas, neural networks are used to approximate the transfer function of electrical components [47, 34]. Here, we emulate this process by defining a hypothetical polynomial component $f(x) = 0.125x^4 - 0.25x^3 - 0.75x^2 + x + 0.5$ with oscillating transfer function. Then, we create a training set by uniformly sampling $f(x)$ in $x \in [-2, 3]$ and train 16 different feedforward ReLU networks $\hat{f}(x)$. The smallest has four layers with four neurons each, and the largest has a single hidden layer with 1024 neurons (see `poly_approx` category in Table 2).

We formally verify the approximation quality by measuring the difference between $\hat{f}(x)$ and $f(x)$ for each possible 32-bit floating point value in $[-2, 3]$. With this information, we write 96 robustness properties (see Table 1). Specifically, we check the input domain in a small interval $\mathcal{X}$ of size 0.1

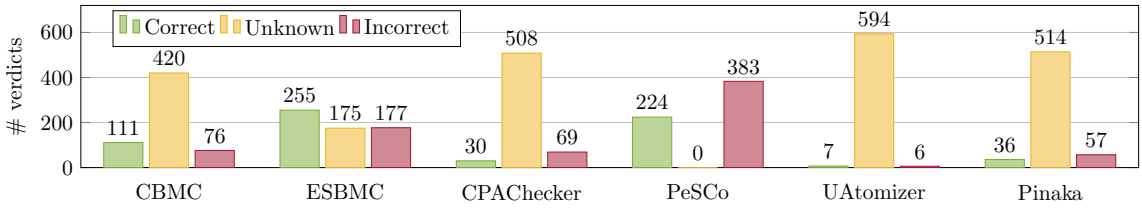

Figure 1: Results of state-of-the-art software verifiers on *NeuroCodeBench* after 900 seconds.

around the worst approximation error. There, we make sure that the error is always below a given threshold $\epsilon \geq |f(x) - \hat{f}(x)|, \forall x \in \mathcal{X}$. We select six decreasing values of ϵ for each network: three make the property hold and three yield a counterexample.

VNN-COMP Networks Since its first edition in 2020, the International Verification of Neural Networks Competition (VNN-COMP) publishes all its benchmarks [38]. These benchmarks do not contain implementation details since they target a higher level of abstraction (real number arithmetic, no memory model). To provide a reference implementation, we propose the following conversion process: we translate the networks from ONNX format [5] to C with `onnx2c` [4], and the safety properties from VNN-LIB [8] to a minimal `main()` function with pre- and post-conditions.

Among all categories of the 2022 edition [38], we select two that contain relatively small neural networks (see Table 2): `reach_prob_density` are networks that approximate probability densities [36], `reinforcement_learning` are control networks trained via reinforcement learning [42].

2.3 Preliminary Evaluation

Here, we use *NeuroCodeBench* to evaluate six software verifiers which achieved top-places[2] in the *Reachability*, *Falsification* and *Floats* categories of SV-COMP 2023 [11]. We keep our experimental setup as similar to SV-COMP as possible: we use the benchmarking tool BenchExec [13] with 2 CPU cores, 6GB of RAM and 900 seconds of total CPU time per verifier for each verification task.

Our preliminary results in Figure 1 show that all six verifiers produce a large ratio of *incorrect-to-correct* verdicts. One of the likely reasons is incomplete support of `math.h` functions, which appear in the first three categories of Table 1. Indeed, CBMC, ESBMC, CPAChecker and UAutomizer produce many math-related warnings in their output, even when their verdict is correct or unknown. At the same time, approximately half of the unknown verdicts are due to timeouts on the larger neural networks of *NeuroCodeBench*, which suggests that the verifiers struggle with their complexity.

3 Conclusions and Future Work

NeuroCodeBench is a challenging benchmark of neural network code in plain C. Our preliminary analysis demonstrates that state-of-the-art verifiers cannot produce correct verdicts on most of our safety properties. In the future, we plan to provide complete operational models for the `math.h` library, whose absence impacts existing verifiers. Furthermore, we plan to contribute *NeuroCodeBench* to SV-COMP and draw the attention of that community to the challenges of verifying neural code.

[2]We omit VeriAbs [10] and VeriAbsL [19] due to licence restrictions. We omit BRICK [16] due to technical issues. We omit cooperative verifiers for clarity. We run PeSCo [43] with the CPAChecker binary from SV-COMP 2023.

References

[1] 2023 cwe top 25 most dangerous software weaknesses, 2023. https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html [Accessed: 25 August 2023].

[2] keras2c github repository, 2023. https://github.com/f0uriest/keras2c [Accessed: 25 August 2023].

[3] Neurocodebench github repository, 2023. https://github.com/emanino/plain_c_nn_benchmark [Accessed: 29 August 2023].

[4] onnx2c github repository, 2023. https://github.com/kraiskil/onnx2c [Accessed: 25 August 2023].

[5] Open neural network exchange: The open standard for machine learning interoperability, 2023. https://onnx.ai/ [Accessed: 31 August 2023].

[6] Pytorch, 2023. https://pytorch.org/ [Accessed: 31 August 2023].

[7] Tensorflow, 2023. https://www.tensorflow.org [Accessed: 31 August 2023].

[8] Vnn-lib: The international benchmarks standard for the verification of neural networks, 2023. https://www.vnnlib.org/ [Accessed: 31 August 2023].

[9] Y. Abu-Mostafa and J. St. Jacques. Information capacity of the hopfield model. *IEEE Transactions on Information Theory*, 31(4):461–464, 1985.

[10] M. Afzal, A. Asia, A. Chauhan, B. Chimdyalwar, P. Darke, A. Datar, S. Kumar, and R. Venkatesh. Veriabs : Verification by abstraction and test generation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1138–1141, 2019.

[11] D. Beyer. Competition on software verification and witness validation: Sv-comp 2023. In S. Sankaranarayanan and N. Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 495–522, Cham, 2023. Springer Nature Switzerland.

[12] D. Beyer and M. E. Keremoglu. Cpachecker: A tool for configurable software verification. In G. Gopalakrishnan and S. Qadeer, editors, *CAV*, pages 184–190, 2011.

[13] D. Beyer, S. Löwe, and P. Wendler. Reliable benchmarking: requirements and solutions. *International Journal on Software Tools for Technology Transfer*, 21:1–29, 2019.

[14] S. Biderman and W. J. Scheirer. Pitfalls in machine learning research: Reexamining the development cycle. In J. Zosa Forde, F. Ruiz, M. F. Pradier, and A. Schein, editors, *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*, volume 137 of *Proceedings of Machine Learning Research*, pages 106–117. PMLR, 12 Dec 2020.

[15] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[16] L. Bu, Z. Xie, L. Lyu, Y. Li, X. Guo, J. Zhao, and X. Li. Brick: Path enumeration based bounded reachability checking of c program (competition contribution). In D. Fisman and G. Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 408–412, Cham, 2022. Springer International Publishing.

[17] R. Chaudhuri and I. Fiete. Bipartite expander hopfield networks as self-decoding high-capacity error correcting codes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[18] R. Conlin, K. Erickson, J. Abbate, and E. Kolemen. Keras2c: A library for converting keras neural networks to real-time compatible c. *Engineering Applications of Artificial Intelligence*, 100:104182, 2021.

[19] P. Darke, B. Chimdyalwar, S. Agrawal, S. Kumar, R. Venkatesh, and S. Chakraborty. Veriabsl: Scalable verification by abstraction and strategy prediction (competition contribution). In S. Sankaranarayanan and N. Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 588–593, Cham, 2023. Springer Nature Switzerland.

[20] Z. Deng, G. Meng, K. Chen, T. Liu, L. Xiang, and C. Chen. Differential testing of cross deep learning framework APIs: Revealing inconsistencies and vulnerabilities. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 7393–7410, Anaheim, CA, Aug. 2023. USENIX Association.

[21] S. Fort, J. Ren, and B. Lakshminarayanan. Exploring the limits of out-of-distribution detection. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 7068–7081. Curran Associates, Inc., 2021.

[22] M. R. Gadelha, F. R. Monteiro, J. Morse, L. C. Cordeiro, B. Fischer, and D. A. Nicole. Esbmc 5.0: An industrial-strength c model checker. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE '18, page 888–891, New York, NY, USA, 2018. Association for Computing Machinery.

[23] Q. Guo, X. Xie, Y. Li, X. Zhang, Y. Liu, X. Li, and C. Shen. Audee: Automated testing for deep learning frameworks. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, page 486–498, New York, NY, USA, 2021. Association for Computing Machinery.

[24] M. Heizmann, M. Barth, D. Dietsch, L. Fichtner, J. Hoenicke, D. Klumpp, M. Naouar, T. Schindler, F. Schüssele, and A. Podelski. Ultimate automizer and the commuhash normal form. In S. Sankaranarayanan and N. Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 577–581, Cham, 2023. Springer Nature Switzerland.

[25] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus), 2023.

[26] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

[27] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella. Taxonomy of real faults in deep learning systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 1110–1121, New York, NY, USA, 2020. Association for Computing Machinery.

[28] IEEE. Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.

[29] ISO. Iso c standard 1999. Technical report, ISO, 1999. ISO/IEC 9899:1999 draft.

[30] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In R. Majumdar and V. Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing.

[31] D. Kroening and M. Tautschnig. CBMC–c bounded model checker. In *TACAS*, pages 389–391, 2014.

[32] T. Likhomanenko, Q. Xu, G. Synnaeve, R. Collobert, and A. Rogozhnikov. Cape: Encoding relative positions with continuous augmented positional embeddings. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 16079–16092. Curran Associates, Inc., 2021.

[33] E. Manino, R. Menezes, F. Shmarov, and L. Cordeiro. Neurocodebench, Aug. 2023. `https://doi.org/10.5281/zenodo.8305733`.

[34] O. Massi, A. I. Mezza, R. Giampiccolo, and A. Bernardini. Deep learning-based wave digital modeling of rate-dependent hysteretic nonlinearities for virtual analog applications. *EURASIP Journal on Audio, Speech, and Music Processing*, 2023(1):12, Mar 2023.

[35] J. B. P. Matos, I. Bessa, E. Manino, X. Song, and L. C. Cordeiro. Ceg4n: Counter-example guided neural network quantization refinement. In O. Isac, R. Ivanov, G. Katz, N. Narodytska, and L. Nenzi, editors, *Software Verification and Formal Methods for ML-Enabled Autonomous Systems*, pages 29–45, Cham, 2022. Springer International Publishing.

[36] Y. Meng, D. Sun, Z. Qiu, M. T. B. Waez, and C. Fan. Learning density distribution of reachable states for autonomous systems. In A. Faust, D. Hsu, and G. Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 124–136. PMLR, 08–11 Nov 2022.

[37] M. M. Morovati, A. Nikanjam, F. Khomh, and Z. M. J. Jiang. Bugs in machine learning-based systems: a faultload benchmark. *Empirical Software Engineering*, 28(3):62, Apr 2023.

[38] M. N. Müller, C. Brix, S. Bak, C. Liu, and T. T. Johnson. The third international verification of neural networks competition (vnn-comp 2022): Summary and results, 2023.

[39] N. Narodytska and S. Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1310–1318, 2017.

[40] C. E. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions : comparison of trends in practice and research for deep learning. In *2nd International Conference on Computational Sciences and Technology*, pages 124 – 133, PAK, January 2021.

[41] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4901–4911. PMLR, 09–15 Jun 2019.

[42] U. J. Ravaioli, J. Cunningham, J. McCarroll, V. Gangal, K. Dunlap, and K. L. Hobbs. Safe reinforcement learning benchmark environments for aerospace control systems. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–20, 2022.

[43] C. Richter and H. Wehrheim. Pesco: Predicting sequential combinations of verifiers. In D. Beyer, M. Huisman, F. Kordon, and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 229–233, Cham, 2019. Springer International Publishing.

[44] L. Sena, X. Song, E. Alves, I. Bessa, E. Manino, L. Cordeiro, and E. de Lima Filho. Verifying quantized neural networks using smt-based model checking, 2021.

[45] H. Suresh and J. Guttag. A framework for understanding sources of harm throughout the machine learning life cycle. In *Equity and Access in Algorithms, Mechanisms, and Optimization*, EAAMO '21, New York, NY, USA, 2021. Association for Computing Machinery.

[46] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 29909–29921. Curran Associates, Inc., 2021.

[47] J. Xu, M. Yagoub, R. Ding, and Q.-J. Zhang. Neural-based dynamic modeling of nonlinear microwave circuits. *IEEE Transactions on Microwave Theory and Techniques*, 50(12):2769–2780, 2002.