eNNcode: Optimization-Based Analysis of Neural Networks

Muhammad Sabri Farag Atallah[1] , **Lukas Dankwart**[1,2] , **Daniel Neider**[1,2] and **Mustafa Yalçıner**[1,2]
[1]TU Dortmund University
[2]Center for Trustworthy Data Science and Security, University Alliance Ruhr, Dortmund, Germany
muhammad.sabri@tu-dortmund.de, lukas.dankwart@tu-dortmund.de, daniel.neider@tu-dortmund.de, mustafa.yalciner@tu-dortmund.de

Abstract

The increasing use of neural networks (NNs) in high-stakes decision-making requires rigorous analysis to ensure safety, fairness, and explainability. Formal verification tools for neural networks typically focus on determining the *satisfiability* of properties such as safety or fairness. However, many fairness and explainability tasks go beyond satisfiability and instead rely on arbitrary *optimization* objectives. To address such problems, neural networks are often encoded as mixed-integer linear optimization (MILO) problems with linear objectives. In practice, these encodings are usually implemented in an ad-hoc manner, limiting comparability across works, reducing transparency, and increasing implementation effort. We address this gap by introducing eNNcode, a user-friendly PyPI library that converts any neural network with piecewise-linear activation function in Open Neural Network Exchange (ONNX) format into a MILO instance. The library supports arbitrary linear constraints on input and output nodes, as well as user-defined optimization objectives. Our experiments show that eNNcode achieves performance comparable to existing libraries, despite its simplicity and ease of use. Overall, eNNcode facilitates reproducible and standardized optimization-based analysis of neural networks.

1 Introduction

Deep neural networks have achieved remarkable success in domains such as computer vision and natural language processing, leading to their widespread adoption in numerous real-world AI applications, including safety-critical areas such as healthcare and autonomous driving. Despite these achievements, neural networks often operate as black-box models, making their internal reasoning difficult to interpret. Moreover, several types of vulnerabilities have been identified in modern architectures, one of the most prominent being *adversarial fragility* [Szegedy *et al.*, 2014]. This term refers to the tendency of models to produce drastically different predictions when their inputs are modified by small, carefully crafted perturbations. The growing awareness of such reliability issues has even led to the establishment of specialized databases to track AI-related incidents and prevent recurring failures [McGregor, 2021].[1]

To address these challenges, formal verification techniques for neural networks have emerged as a powerful tool to ensure the safety and reliability of neural networks. Unlike statistical methods such as cross-validation that evaluate model behavior on finite datasets, formal verification aims to mathematically prove whether certain properties, such as safety or robustness, hold for all possible inputs. These approaches typically check the *satisfiability* of logical specifications [Demarchi *et al.*, 2023], which is sufficient to determine whether a given property is guaranteed by the network.

However, tasks related to explainability or fairness often go beyond satisfiability and instead require solving optimization problems, for example, determining the minimal change a loan applicant must make for their application to be accepted. While user-friendly tools for formal verification exist already, such as the CAISAR platform [Girard-Satabin *et al.*, 2022], there is still no widely adopted framework that supports optimization-based analysis of neural networks. In fact, encoding a neural network into MILO is usually implemented in an ad-hoc manner, limiting comparability across works, reducing transparency, and increasing implementation effort. To bridge this gap, we introduce eNNcode, a lightweight and user-friendly library that converts any neural network with piecewise-linear activation functions into a MILO instance. In contrast to existing tools such as Gurobi-ML [Gurobi Optimization, LLC, 2024b], eNNcode accepts models in the standardized ONNX fformat, making it independent of underlying machine-learning libraries, and supports convolutional layers that Gurobi-ML does not handle. Moreover, unlike OMLT [Ceccon *et al.*, 2022], eNNcode remains lightweight, easy to install, and designed for maximum usability. Our experiments demonstrate that eNNcode achieves runtime and memory performance comparable to existing libraries while overcoming their practical limitations in usability and model compatibility. A demonstration of our PyPI-library[2] is available as a video.[3]

[1]https://incidentdatabase.ai/
[2]https://pypi.org/project/enncode/
[3]https://doi.org/10.5281/zenodo.20427178

2 Related Work

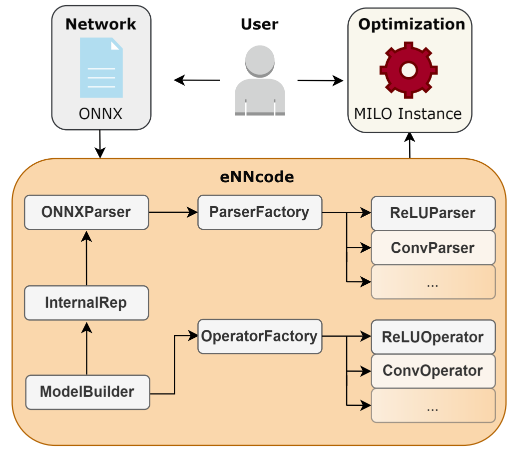
Constrained optimization in machine learning is an active research area.[4] In particular, mathematical optimization over neural networks is a central theme in research on counterfactual explanations and various other topics [Jiang *et al.*, 2023b; Marzari *et al.*, 2024; Jiang *et al.*, 2024; Benussi *et al.*, 2022; Jiang *et al.*, 2023a; Wu *et al.*, 2020; ElAraby *et al.*, 2023; Ovalle *et al.*, 2025; Dong *et al.*, 2025; Stinchfield *et al.*, 2025; Grimstad and Andersson, 2019; Papalexopoulos *et al.*, 2022; Lorenz *et al.*, 2025]. Despite this relevance, there is no widely adopted tool for encoding neural networks into optimization models. Many of the above works implement custom formulations rather than relying on a shared library—hindering comparability and increasing prototyping effort. We therefore review existing libraries for encoding neural networks as constraints and discuss limitations that may impede broader adoption. The most prominent tool is Gurobi Machine Learning (GurobiML) [Gurobi Optimization, LLC, 2024b], which converts trained models directly into constraints within a Gurobi model. While reliable, GurobiML supports only linear layers with ReLU activations and lacks compatibility with the standardized ONNX format; it instead requires access to source code from frameworks such as PyTorch or Keras. Other tools build upon algebraic modeling languages such as Pyomo [Bynum *et al.*, 2021]. For instance, ReluMIP [Lueg *et al.*, 2021] translates ReLU-based networks into MILO formulations via Pyomo or Gurobi interfaces. Similarly, the Optimization & Machine Learning Toolkit (OMLT) [Ceccon *et al.*, 2022] embeds trained ML models into optimization problems using Pyomo to express network architectures as constraints. Although this promotes solver independence, significant additional effort emerges during model instantiation. Finally, verification frameworks such as DNNV [Shriver *et al.*, 2021] offer unified interfaces for verifiers like MIPVerify [Tjeng *et al.*, 2019] and Marabou [Wu *et al.*, 2024]. However, these tools only support verification tasks rather than optimization, placing them outside of our scope.

3 Overview

We visualize the workflow of eNNcode in Figure 1. Consider a user who has trained a neural network and exported it in the standardized Open Neural Network Exchange (ONNX) format. The goal is to perform rigorous mathematically grounded analysis, such as optimization or verification, on the network. To enable this, eNNcode converts the ONNX model into a MILO instance, where the input nodes of the neural network become free decision variables and all intermediate and output nodes are defined as linear constraints that faithfully encode the network's behavior. This transformation allows users to express arbitrary linear objectives or additional constraints on inputs and outputs while preserving exact functional equivalence with the original neural network.

On a technical level, eNNcode begins by parsing the provided ONNX file into an internal representation of the neural network. The parser follows a factory-pattern architecture, instantiating specialized parsers for each supported node operation to ensure modularity and extensibility. Currently, eNNcode supports all piecewise linear operators defined in the Verification of Neural Networks Competition [Demarchi *et al.*, 2023], including `Add`, `Sub`, `Mul`, `Div`, `MatMul`, `AveragePool`, `MaxPool`, `Conv`, `ReLU`, `Concat`, `Reshape`, `Flatten`, `Unsqueeze`, `BatchNormalization`, `Dropout`, `Gemm`, and `Identity`. In practice, these operations can represent architectures that are powerful enough for image-based tasks [Redmon *et al.*, 2016; Krizhevsky *et al.*, 2012; Simonyan and Zisserman, 2015].

[4]See NeurIPS'25 Workshop: Constrained Optimization for ML




```python
from enncode.gurobiModelBuilder import GurobiModelBuilder

onnx_path = "path/to/your/network.onnx"
model_builder = GurobiModelBuilder(onnx_path)
model_builder.build_model()
gurobi_model = model_builder.get_gurobi_model()
input_vars = model_builder.get_input_vars()
output_vars = model_builder.get_output_vars()
# Define arbitrary objectives...
```

Figure 1: Workflow with eNNcode

Once parsed, the internal representation contains all information required to encode the network as a MILO instance. Similarly to parsing, encoding also adopts a factory pattern in which each operation type has a dedicated implementation that converts the operation into linear constraints. After successful initialization, the resulting MILO model is returned to the user, who can then define additional constraints or optimization objectives directly on this instance. To facilitate usability, metadata such as variable names corresponding to input and output nodes are exposed through the interface. The lower half of Figure 1 illustrates, from a user-perspective, how simple it is to obtain the Gurobi instance from an ONNX-formatted network with eNNcode.

Compatibility ONNX files represent the computational graph of a neural network. The set of supported operators, their attributes and expected semantics are defined by the Operator Set (Opset) [The ONNX Community, 2026]. Because newer Opset versions may introduce additional operators or modify existing ones, the current implementation of eNNcode is based on the specifications of the Opset version 11 but remains compatible with other versions, provided that they do not alter the definitions of node types supported

Variant	**Time** (Sec.) MNIST	MNIST *(CNN)*	Concrete	Wine	**Storage** (MByte) MNIST	MNIST *(CNN)*	Concrete	Wine
eNNCode	10.686 ± 5.147	13.901 ± 09.291	0.0256 ± 0.0013	0.0335 ± 0.0014	01.299 ± 0.0852	01.712 ± 0.085	0.0451 ± 0.0107	0.0534 ± 0.0110
OMLT	08.057 ± 1.706	14.077 ± 23.281	0.0405 ± 0.0136	0.0474 ± 0.0140	21.770 ± 0.1271	13.919 ± 0.356	0.2911 ± 0.2662	0.3451 ± 0.2670
Gurobi-ML	05.919 ± 4.657	-	0.0223 ± 0.0005	0.0241 ± 0.0009	00.818 ± 0.0006	-	0.0521 ± 0.0008	0.0559 ± 0.0006

Table 1: Overview on the time and memory consumption.

by eNNcode. To ensure correct translation, compatibility of the ONNX model with the GurobiPy instance is automatically verified. When incompatibilities are detected, detailed and precise error messages are returned to the user. Upon successful conversion, eNNcode performs an automatic correctness validation to ensure that the MILO encoding faithfully represents the original neural network. This validation compares the outputs produced by the encoded optimization model with those computed by the ONNX Runtime [Microsoft and ONNX Runtime Contributors, 2024]. Specifically, MILO input variables are fixed to random values, and the optimizer's output is compared against ONNX Runtime references. If discrepancies occur, eNNcode iteratively analyzes subgraphs to isolate faulty nodes and provide precise user feedback.

4 Experiments

Although the main contribution of our library is qualitative, namely providing a user-friendly interface with broad support for neural layer types, we also present a quantitative comparison in terms of runtime and memory usage.

Baselines We compare eNNcode with two existing libraries: OMLT [Ceccon *et al.*, 2022] and Gurobi Machine Learning [Gurobi Optimization, LLC, 2024b]. OMLT is an optimizer-agnostic framework built on top of Pyomo [Bynum *et al.*, 2021], offering support for encoding neural networks in ONNX models as optimization models. Gurobi Machine Learning is an open-source library developed by Gurobi that supports relatively simple architectures, excluding convolutional layers, for several major machine-learning frameworks such as TensorFlow and PyTorch. We exclude ReLU-MIP [Lueg *et al.*, 2021] from our comparison because it only supports sequential TensorFlow models, which we consider too restrictive for general applicability. Verification frameworks are also omitted, as they are restricted to property checking rather than optimization tasks, which is the focus of our work. To the best of our knowledge, OMLT and Gurobi-ML represent the only publicly available libraries capable of translating neural networks into MILO instances without imposing strong restrictions.

Benchmark In our experiments, we search for counterfactual explanations (CXs) across various neural network classifiers. For a given input $x \in \mathbb{R}^n$ to a (binary) classifier $h : \mathbb{R}^n \rightarrow \{0, 1\}$, a CX is an instance $x' \in \mathbb{R}^n$ that induces an alternative classification, i.e., $h(x') \neq h(x)$, while remaining as close as possible to the original input according to the distance $d_{\ell_1}(x, x')$ [Wachter *et al.*, 2017]. This definition extends naturally to multi-class classifiers.

Our benchmark includes four networks. Two of them are taken from the example notebooks of OMLT [Cecrle *et al.*, 2024] and trained on the MNIST dataset [Deng, 2012]. The remaining two are binary classifiers for tabular datasets commonly used in counterfactual explanation research: one trained on the *Concrete Compressive Strength Dataset* [Yeh, 1998], and the other on the *Wine Quality Dataset* [Cortez *et al.*, 2009]. All four models exclusively use ReLU activation functions. We evaluate runtime and memory consumption over multiple iterations. In each iteration, a network receives an original data point from its corresponding dataset along with a randomly selected target class that differs from the ground-truth label. We measure all steps required for model initialization and subsequent optimization. Additionally, we impose a time limit of 240 seconds for each optimization call.

Results & Discussion The results are summarized in Table 1. We first discuss the runtime performance. In general, no clear winner can be identified across all benchmarks. For the fully connected MNIST network, Gurobi-ML achieves the best average runtime, but this advantage is limited since it does not support convolutional layers and cannot be applied to the MNIST CNN benchmark [Gurobi Optimization, LLC, 2024a]. In contrast, both *eNNCode* and OMLT support fully connected as well as convolutional networks and achieve comparable runtimes, with a slight advantage of *eNNCode* over OMLT on the tabular classifiers. Within the specified time limit, all variants find counterfactual explanations except for OMLT on the MNIST CNN, where approximately 40% of the runs reach time out (excluded from the reported results). Regarding memory consumption, OMLT consistently exhibits by far the highest storage requirements in all test cases, likely due to its more complex internal workflow. In comparison, *eNNCode* requires significantly less memory and remains comparable to Gurobi-ML across the benchmarks.

Key Takeaways Overall, eNNcode achieves competitive runtime performance across all benchmarks while also supporting convolutional architectures. Moreover, it consistently requires low memory and remains comparable to Gurobi-ML, making it a robust and broadly applicable encoding approach.

5 Conclusion & Future Work

We presented eNNcode, a user-friendly PyPi library for encoding neural networks in ONNX format into MILO instances. In future work, we plan to extend eNNcode to support encodings for the Z3 optimizer [de Moura and Bjørner, 2008]. Moreover, integrating our library into the CAISAR platform could facilitate a wider adoption and a seamless switching between verification and optimization tasks.

Acknowledgements

This work has been financially supported by Deutsche Forschungsgemeinschaft, DFG Project numbers 459419731 and 434592664.

References

[Benussi *et al.*, 2022] Elias Benussi, Andrea Patanè, Matthew Wicker, Luca Laurenti, and Marta Kwiatkowska. Individual fairness guarantees for neural networks. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 651–658. ijcai.org, 2022.

[Bynum *et al.*, 2021] Michael L. Bynum, Gabriel A. Hackebeil, William E. Hart, Carl D. Laird, Bethany L. Nicholson, John D. Siirola, Jean-Paul Watson, and David L. Woodruff. *Pyomo–optimization modeling in python*, volume 67, chapter 1, 6 & 9. Springer Science & Business Media, third edition, 2021.

[Ceccon *et al.*, 2022] Francesco Ceccon, Jordan Jalving, Joshua Haddad, Alexander Thebelt, Calvin Tsay, Carl D. Laird, and Ruth Misener. OMLT: optimization & machine learning toolkit. *J. Mach. Learn. Res.*, 23:349:1–349:8, 2022.

[Cecrle *et al.*, 2024] Jordan Cecrle, Jordan Jalving, Radwa Haddad, Alexander Thebelt, Calvin Tsay, Carl D. Laird, and Ruth Misener. OMLT documentation: Examples and notebooks. https://omlt.readthedocs.io/en/latest/notebooks.html, 2024. Accessed: 2026-04-02.

[Cortez *et al.*, 2009] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Wine Quality Dataset. UCI Machine Learning Repository, 2009. https://archive.ics.uci.edu/dataset/186/wine+quality.

[de Moura and Bjørner, 2008] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[Demarchi *et al.*, 2023] Stefano Demarchi, Dario Guidotti, Luca Pulina, and Armando Tacchella. Supporting standardization of neural networks verification with VNNLIB and coconet. In Nina Narodytska, Guy Amir, Guy Katz, and Omri Isac, editors, *Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems, FoMLAS@CAV 2023, Paris, France, July 17-18, 2023*, Kalpa Publications in Computing, pages 47–58. EasyChair, 2023.

[Deng, 2012] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[Dong *et al.*, 2025] Zihang Dong, Cheng Hu, Zeyuan Xu, Goran Strbac, and Dawei Qiu. Machine learning-based economic model predictive control for energy hubs with variable energy efficiencies. *Energy*, 334:137574, 2025.

[ElAraby *et al.*, 2023] Mostafa ElAraby, Guy Wolf, and Margarida Carvalho. *OAMIP: Optimizing ANN Architectures Using Mixed-Integer Programming*, page 219–237. Springer Nature Switzerland, 2023.

[Girard-Satabin *et al.*, 2022] Julien Girard-Satabin, Michele Alberti, François Bobot, Zakaria Chihani, and Augustin Lemesle. CAISAR: A platform for Characterizing Artificial Intelligence Safety and Robustness. In *AISafety*, CEUR-Workshop Proceedings, Vienne, Austria, July 2022.

[Grimstad and Andersson, 2019] Bjarne Grimstad and Henrik Andersson. ReLU networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131:106580, December 2019.

[Gurobi Optimization, LLC, 2024a] Gurobi Optimization, LLC. Gurobi machine learning documentation: Supported regression models. https://gurobi-machinelearning.readthedocs.io/en/stable/user/supported.html, 2024. Accessed: 2026-02-03.

[Gurobi Optimization, LLC, 2024b] Gurobi Optimization, LLC. Gurobi Machine Learning: Mathematical Optimization with Machine Learning Models. https://github.com/Gurobi/gurobi-machinelearning, 2024. Python package version 1.4.1.

[Jiang *et al.*, 2023a] Junqi Jiang, Jianglin Lan, Francesco Leofante, Antonio Rago, and Francesca Toni. Provably Robust and Plausible Counterfactual Explanations for Neural Networks via Robust Optimisation. In Berrin Yanikoglu and Wray L. Buntine, editors, *Asian Conference on Machine Learning, ACML 2023, 11-14 November 2023, Istanbul, Turkey*, Proceedings of Machine Learning Research, pages 582–597. PMLR, 2023.

[Jiang *et al.*, 2023b] Junqi Jiang, Francesco Leofante, Antonio Rago, and Francesca Toni. Formalising the robustness of counterfactual explanations for neural networks. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*, pages 14901–14909. AAAI Press, 2023.

[Jiang *et al.*, 2024] Junqi Jiang, Francesco Leofante, Antonio Rago, and Francesca Toni. Interval abstractions for robust counterfactual explanations. *Artif. Intell.*, 336:104218, 2024.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.

[Lorenz *et al.*, 2025] Tobias Lorenz, Marta Kwiatkowska, and Mario Fritz. MIBP-Cert: Certified Training against Data Perturbations with Mixed-Integer Bilinear Programs. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, pages 98193–98216. Curran Associates, Inc., 2025.

[Lueg *et al.*, 2021] Laurens Lueg, Bjarne Grimstad, Alexander Mitsos, and Artur M. Schweidtmann. reluMIP: Open Source Tool for MILP Optimization of ReLU Neural Networks, 2021. Version 1.0.0.

[Marzari *et al.*, 2024] Luca Marzari, Francesco Leofante, Ferdinando Cicalese, and Alessandro Farinelli. Rigorous Probabilistic Guarantees for Robust Counterfactual Explanations. In *ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 1059–1066. IOS Press, 2024.

[McGregor, 2021] Sean McGregor. Preventing repeated real world AI failures by cataloging incidents: The AI incident database. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 15458–15463. AAAI Press, 2021.

[Microsoft and ONNX Runtime Contributors, 2024] Microsoft and ONNX Runtime Contributors. Onnx runtime documentation. https://onnxruntime.ai/docs/, 2024. Accessed: 2026-01-05.

[Ovalle *et al.*, 2025] Daniel Ovalle, Lorenz Biegler, Ignacio Grossmann, Carl Laird, and Mateo Dulce Rubio. Conformal Mixed-Integer Constraint Learning with Feasibility Guarantees. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, pages 80387–80423. Curran Associates, Inc., 2025.

[Papalexopoulos *et al.*, 2022] Theodore P. Papalexopoulos, Christian Tjandraatmadja, Ross Anderson, Juan Pablo Vielma, and David Belanger. Constrained Discrete Black-Box Optimization using Mixed-Integer Programming. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, Proceedings of Machine Learning Research, pages 17295–17322. PMLR, 2022.

[Redmon *et al.*, 2016] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788. IEEE Computer Society, 2016.

[Shriver *et al.*, 2021] David Shriver, Sebastian G. Elbaum, and Matthew B. Dwyer. DNNV: A framework for deep neural network verification. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*, Lecture Notes in Computer Science, pages 137–150. Springer, 2021.

[Simonyan and Zisserman, 2015] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[Stinchfield *et al.*, 2025] Georgia Stinchfield, Natali Khalife, Bashar L. Ammari, Joshua C. Morgan, Miguel Zamarripa, and Carl D. Laird. Mixed-Integer Linear Programming Formulation with Embedded Machine Learning Surrogates for the Design of Chemical Process Families. *Industrial & Engineering Chemistry Research*, 64(16):8299–8311, 2025.

[Szegedy *et al.*, 2014] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[The ONNX Community, 2026] The ONNX Community. Open neural network exchange (onnx) operators, 2026. Accessed: 2026-02-10.

[Tjeng *et al.*, 2019] Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[Wachter *et al.*, 2017] Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard Journal of Law & Technology*, 31(2):841–887, 2017.

[Wu *et al.*, 2020] Ga Wu, Buser Say, and Scott Sanner. Scalable Planning with Deep Neural Network Learned Transition Models. *J. Artif. Intell. Res.*, 68:571–606, 2020.

[Wu *et al.*, 2024] Haoze Wu, Omri Isac, Aleksandar Zeljic, Teruhiro Tagomori, Matthew L. Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, Pei Huang, Ori Lahav, Min Wu, Min Zhang, Ekaterina Komendantskaya, Guy Katz, and Clark W. Barrett. Marabou 2.0: A versatile formal analyzer of neural networks. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part II*, Lecture Notes in Computer Science, pages 249–264. Springer, 2024.

[Yeh, 1998] I-Cheng Yeh. Concrete Compressive Strength Dataset. UCI Machine Learning Repository, 1998. https://archive.ics.uci.edu/dataset/165/concrete+compressive+strength.