Certified Neural Networks: from Verification to Synthesis

MATTEO ZAVATTERI*, Fondazione Bruno Kessler, Italy
DAVIDE BRESOLIN, Department of Mathematics, University of Padova, Italy
NICOLÒ NAVARIN, Department of Mathematics, University of Padova, Italy

Neural networks find applications in many safety-critical systems that raise concerns about their deployment: Are we sure they never advise doing anything catastrophic? Formal verification has been recently applied to prove whether an existing neural network is certified for some property; i.e., if it satisfies the property for all possible inputs or not. Formal verification can prove that a network satisfies a property but cannot fix the network in case it doesn't. In this paper we focus on the automated synthesis of certified neural networks, that is, on how to automatically build a network that is guaranteed to respect some required properties expressed as logical constraints. We exploit a Counter Example Guided Inductive Synthesis (CEGIS) loop that alternates Deep Learning, Formal Verification, and a novel data generation technique that augments the training data to synthesize certified networks in a fully automated way. We identify a few conditions guaranteeing termination of the approach. We also investigate a soft constraint acceleration technique to reduce the number of CEGIS iterations and we test our approach to synthesize safety neural controllers for four case studies: a social robot application scenario, an expense prediction scenario, and two versions of the Airborne Collision Avoidance System X for unmanned aircraft benchmark (HCAS and ACASXu).

JAIR Track: Integration of Logical Constraints in Deep Learning

JAIR Associate Editor: Francesco Calimeri

JAIR Reference Format:
Matteo Zavatteri, Davide Bresolin, and Nicolò Navarin. 2026. Certified Neural Networks: from Verification to Synthesis. *Journal of Artificial Intelligence Research* 87, Article 3 (September 2026), 57 pages. DOI: 10.1613/jair.1.22825

1 Introduction and Related Work

Applications of neural networks include fields where they act as action advisors and controllers for safety-critical systems, where safety is paramount, and errors can be extremely expensive or life-threatening. When a neural network is deployed in a safety-critical system, it is of extreme importance to prove that it satisfies the desired properties in all possible scenarios and under all possible inputs. Formal verification provides algorithms and methodologies that can exhaustively explore the state space of a system to guarantee that the system itself respects the desired properties, and it is routinely applied in the design of hardware and software systems. More recently, the challenge of the certification of neural networks has been receiving attention from the formal verification community: the annual competition VNN-COMP (Brix et al. 2023) and the standard format VNN-LIB (Tacchella et al. 2022) have been established to compare state-of-the-art neural network verification tools and share benchmarks and test cases. The interest by the industry is testified by the recent discussion on the use of formal methods for neural networks certification in aviation (EASA and Collins Aerospace 2023), and by

*Corresponding Author.

Authors' Contact Information: Matteo Zavatteri, ORCID: 0000-0001-6696-2972, zavatteri@fbk.eu, Fondazione Bruno Kessler, Trento, Italy; Davide Bresolin, ORCID: 0000-0003-2253-9878, davide.bresolin@unipd.it, Department of Mathematics, University of Padova, Padova, Italy; Nicolò Navarin, ORCID: 0000-0002-4108-1754, nicolo.navarin@unipd.it, Department of Mathematics, University of Padova, Padova, Italy.

DOI: 10.1613/jair.1.22825

legislative texts such as the AI act by the European Union (European Parliament 2024), that will require for all AI systems categorized as *high-risk* to be assessed before being deployed.

Many of the current approaches to verify a neural network reduce the problem to finding a solution to a constraint satisfaction problem (CSP). Such a CSP consists of a set of constraints encoding the neural network and a set of constraints encoding the property, and it is typically formulated within the frameworks of Mixed Integer Linear Programming (MILP), Satisfiability Modulo Theories (SMT), or more generally Constraint Programming (CP). Some selected references of these (and other) formal verification methods can be found in (Albarghouthi 2021). Verifying ReLU neural networks against linear properties is NP-complete (Katz, Barrett, et al. 2017). To address this computational challenge, some tools such as RELUPLEX (Katz, Barrett, et al. 2017) and Marabou (Katz, Huang, et al. 2019) provide specialized versions of the SIMPLEX method to boost scalability of the formal verification.

However, verification alone can only give a yes or no answer and cannot *synthesize* a neural network that is guaranteed to respect the property when the answer to the verification problem is negative.

A few attempts in the literature to deal with the synthesis of certified neural networks have been provided. Some of them are restricted to general monotonicity of the network (Liu et al. 2020; Sivaraman et al. 2020), whereas others are limited to constraining the output for Hierarchical Multi-label classification Problems (Deng et al. 2014; Dragone et al. 2021; Giunchiglia and Lukasiewicz 2021). Considering more general rules, some works (Li, Gupta, et al. 2019; Li and Srikumar 2019; Wang and Pan 2020; Xie et al. 2019; Xu et al. 2018) propose to inject constraints in the loss function of a neural network in such a way that the training of the model should prefer solutions where the constraints are satisfied. (Minervini and Riedel 2018) generates counterexamples (solving an optimization problem) that violate some constraints that are expressed in first-order logic (FOL) and include them in the training set in NLP tasks. (Daniele and Serafini 2022) expresses constraints as logical clauses directly incorporated in the network structure as an additional layer that adds trainable parameters (clause weights) that represent the satisfiability level of constraints, providing explainability to the model as well as increased generalization. Some other methods additionally provide guarantees on the considered constraints, even though they either pose limitations on the constraints (Hoernle et al. 2022; Xu et al. 2018), or rely on an external component to enforce the desired properties at run time (Alshiekh et al. 2018; Cornelio et al. 2023; Zhu et al. 2019). Finally, a recent review of deep learning with logical constraints is provided in (Giunchiglia, Stoian, et al. 2022).

The problem of automatically generating a computational system that provably satisfies a given high-level specification has been studied in computer science since the seminal works of Church (Church 1964). One relevant methodology developed by this line of research is the Counter Example Guided Inductive Synthesis (CEGIS) workflow, initially proposed in (Solar-Lezama et al. 2006) for the automated synthesis of programs. The CEGIS workflow has been recently applied to prove stability of neural controllers (Abate et al. 2021; Chang et al. 2019) and to generate invariants for dynamical systems (Chatterjee et al. 2023; Peruffo et al. 2021). While these works exploit a learning-verify loop similar to our approach, they differ in a key aspect: they use learning to prove that the system satisfies the desired properties, starting from an analytical description of the dynamical system. A framework that is closer to our methodology is described in (Goldberger et al. 2022), where the CEGIS loop maintains a list of points on which the network is known to produce a faulty result, and then finds a modification to the weights of the network that simultaneously corrects all of these points. The loop is repeated until the network is verified. While this work aims to minimize the changes needed to certify the network, as we do, it restricts itself to change only the last layer of the network in the modification phase, due to the computational complexity of the underlying optimization problem. Likewise, SpecREPAIR (Bauer-Marquart et al. 2022) is an architecture based on CEGIS where counterexamples are sought by solving an optimization problem and a repair phase injects information on these counterexamples in the loss function to retrain the network. Certification is guaranteed by a final phase of formal verification. (Boetius et al. 2023) studies the problem of

Table 1. A comparison of Certified AI techniques in neural networks.

Approach	Guarantees	Scalability	Properties	Cost	Main references
Verification methods	Sound approach (if counterexamples are found, then we have evidence that the network is not certified). May be incomplete (depends on type of networks and properties). Complete for ReLU neural networks with linear properties.	Typically NP-hard (handles networks with a few hundreds of neurons).	Robustness (i.e., find counterexamples around a given point); linear (relating pairs of inputs and outputs).	From P to NP-hard, depending on property type.	CROWN (Zhang et al. 2018), Reluplex (Katz, Barrett, et al. 2017), Marabou (Katz, Huang, et al. 2019)
Repair / synthesis guided by counterexamples	Iterative improvement: each iteration aims at eliminating a set of counterexamples. In general, there is no termination guarantee.	Depends on the strategy employed to repair parts of the network or to find counterexamples. Also, typically grows linearly with the number of iterations.	Depend on the type of network.	At least NP-hard due to formal verification and/or solving optimization problems for repair.	DeepSADE (Goyal et al. 2024), SperRE-PAIR (Bauer-Marquart et al. 2022), Other CEGIS approaches (Zavatteri et al. 2024), (Boetius et al. 2023).

termination in the counter-example guided inductive synthesis setting pointing out that for many interesting classes of specifications, termination still remains to be fully investigated. Finally, another iterative approach worth mentioning is DeepSADE (Goyal et al. 2024) in which a combination of MaxSAT and gradient descend is used to certify the networks by changing the weights of the last layer of network. Table 1 provides a high-level comparison of these Certified AI techniques.

Our setting differs from most of current approaches as it is based on a data generation technique. We start from a dataset sampled from an unknown distribution, and we aim to preserve the accuracy of the network as much as possible, without adding restrictions on how we modify the network or how we modify the training. Such a technique is general as it works also with datasets whose data is scarce or even starting from empty datasets.

Contribution and Organization

This paper combines Formal Methods with Deep Learning and employs the Counter Example Guided Inductive Synthesis (CEGIS) workflow for the automated synthesis of certified neural networks.

This work is an extended version of (Zavatteri et al. 2024) that was presented at the *27th European Conference on Artificial Intelligence (ECAI 2024)* in Santiago de Compostela, Spain. In what follows we provide the organization of the paper and detail the extensions this paper provides.

- Section 2 provides background on feedforward ReLU neural networks and their formal verification with respect to the properties that we consider in this paper. Differently from (Zavatteri et al. 2024), here we also consider sets of properties instead of single properties.

- Section 3 introduces the notion of ε-certification to embed explicit and arbitrary closeness criteria to the satisfaction of properties. This section provides new contribution with respect to (Zavatteri et al. 2024) and will play an important role when discussing termination.
- Section 4 provides the general pipeline for the synthesis workflow and identifies, in a technology-independent way, the building blocks that we need to obtain our goal. Furthermore, it discusses a novel data generation technique that generates new data by repairing counterexamples. This part extends that introduced in (Zavatteri et al. 2024) to sets of properties guaranteeing that the repair of counterexamples does not generate new data points that violate some other property in the set.
- Section 5 provides theoretical results on termination of our approach. This section is new contribution with respect to (Zavatteri et al. 2024).
- Section 6 extends the approach to properties involving multiple input-output pairs. This part revises the repair technique introduced in (Zavatteri et al. 2024) for these kind of properties, extends it to conjunctions of properties and highlights current computational challenges and limitations.
- Section 7 discusses a *soft constraint acceleration* technique that injects soft constraints during training in order to reduce the number of CEGIS iterations. Such a technique works on the structure of the constraints expressing properties and automatically computes a factor to add to the loss used during training phases. This section provides new contribution with respect to (Zavatteri et al. 2024).
- Section 8 discusses and applies our results to a social robot scenario in which a neural network acts as a safety controller to decide which actions are allowed next. Here, with respect to (Zavatteri et al. 2024), we also consider more properties.
- Section 9 discusses and applies our results to the Horizonal Collision Avoidance Systems (HCAS) a specialzied version of the Airborne Collision Avoidance X for unmanned aircraft (ACASXu) where a set of neural networks act a safety controller to advise horizontal mid-air maneuvers. This section is new contribution with respect to (Zavatteri et al. 2024). We show that we can certify neural networks for several properties altogether.
- Section 10 compares with DeepSADE (Goyal et al. 2024) and SpecREPAIR (Bauer-Marquart et al. 2022), two state of the art, CEGIS-based approaches, for synthesizing certified neural networks. This section is new contribution with respect to (Zavatteri et al. 2024). We show that our approach outperforms DeepSADE, and we report on some limitations we encountered that made the comparison with SpecREPAIR not possible.
- Section 11 draws conclusions and discusses future work.

The sections not strictly introducing new results have been revised to provide a more thorough and clear presentation of the original results in appearing in (Zavatteri et al. 2024).

2 Background

In this section, we give the needed background on feedforward ReLU neural networks and their formal verification.

2.1 Feedforward ReLU Neural Networks

A feedforward ReLU neural network is a directed weighted graph that meets the following restrictions:

- The network always includes an *input layer*, an *output layer*, and may include *hidden layers* in between. A network with exactly one hidden layer is *shallow*; it is *deep* if there exist at least two hidden layers. We denote with d the number of layers in the network. Layer 1 is the input layer, while layer d is the output layer. Layers $2, \ldots, d-1$ are the hidden layers.
- A layer is a vector of *neurons* (each neuron is modeled as a node in the graph). We denote with ℓ_i the size of the i-th layer, that is, the number of neurons in the layer. The j-th neuron of layer i is denoted by $v_{i,j}$. To ease readability, we omit the subscript j when the corresponding layer consists of only one neuron.

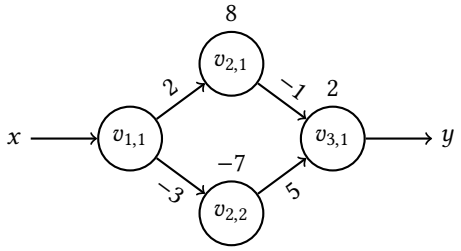


Fig. 1. A neural network with one hidden layer.

- Each neuron belonging to the i^{th} layer with $1 \leq i < d$ is connected to all neurons in the $(i+1)^{\text{th}}$ layer through directed weighted edges.
- Hidden neurons serve as computational units. Every hidden neuron $v_{i,j}$ also has an associated *bias* $b_{i,j}$ and a non-linear *activation function*. The neuron computes a linear function over the output of the neurons in the previous layer, and then it applies the activation function to generate its output. Here, we consider the Rectified Linear Unit activation function: $\text{ReLU}(x) := \max(0, x)$.
- Output neurons also have associated biases but no activation functions. In this way, they compute a linear combination of the outputs of the neurons in the last hidden layer.

Figure 1 gives an example of a feedforward ReLU neural network with a single input neuron $v_{1,1}$, a single output neuron $v_{3,1}$, and a hidden layer containing 2 neurons $v_{2,1}, v_{2,2}$ with ReLU activation functions. Biases are shown above the neurons, and weights label the edges. The network computes the function:

$$f_N(x) := -\overbrace{(\max(0, 2x + 8))}^{\text{computed by } v_{2,1}} + 5\overbrace{(\max(0, -3x - 7))}^{\text{computed by } v_{2,2}} + 2$$

where x is the single input of the network.

2.2 Properties

In this section we define the formal language we will use to formalize the properties that neural networks should satisfy. Before going ahead, we point out some basic notation we will use throughout the paper.

- We write $\vec{x} = (x_1, \ldots, x_n)$ to denote a vector of real variables;
- We write $\vec{\mathbf{x}} = (\mathbf{x_1}, \ldots, \mathbf{x_n}) \in \mathbb{R}^n$ to denote a vector of variable valuations. That is, for each $i = 1, \ldots, n$, $\mathbf{x_i}$ is the value assigned to the variable x_i.
- We write $(\vec{x}, \vec{y})$ as a shorthand for $(x_1, \ldots, x_n, y_1, \ldots, y_m)$ and, likewise, we write $(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ as a shorthand for $(\mathbf{x_1}, \ldots, \mathbf{x_n}, \mathbf{y_1}, \ldots, \mathbf{y_m})$ (i.e., vector concatenation).
- We use $n, m \in \mathbb{N}$ to represent input and output dimensions of the neural network of interest. We use $h, k, z \in \mathbb{N}$ whenever we want to talk about dimensions that are not related to input nor to output.
- As already pointed out in Section 2.1, f_N is the function computed by the neural network N.

We rely on the theory of linear real arithmetic (LRA) to write the properties. LRA formulas are first-order logic formulas built on top of atoms that are linear constraints $\sum_{i=1}^{n} c_i x_i \bowtie b$, where c_i, b are rational constants, and $\bowtie \in \{<, \leq, =, >, \geq, \neq\}$. To simplify the definitions, we restrict our attention to properties that are built from monotone formulas with no negations nor strict inequalities $<, >$.

Definition 2.1. Given a vector of real variables $\vec{x} = (x_1, \ldots, x_k)$, a *non-strict monotone formula* $F(\vec{x})$ on linear constraints over $\vec{x}$ is any formula arising from the following grammar:

$$F(\vec{x}) ::= b + \sum_{i=1}^{k} c_i x_i \leq 0 \mid F(\vec{x}) \wedge F(\vec{x}) \mid F(\vec{x}) \vee F(\vec{x}) \mid (F(\vec{x}))$$

where $c_i, b \in \mathbb{Q}$.

Arbitrary linear constraints of the form $\sum_{i=1}^{k} c_i x_i \bowtie b$, where $\bowtie \in \{\leq, =, \geq\}$, as well as the boolean constants *true* and *false*, can be rewritten to follow the above definition.

The properties of interest typically impose conditions on the relation between inputs and output of the network, and as such can be easily encoded in LRA.

Definition 2.2 (Property). Given a neural network N, a *property* is any formula that can be written as follows:

$$P(\vec{x}, \vec{y}) := F_{pre}(\vec{x}) \Rightarrow F_{post}(\vec{x}, \vec{y}),$$

where $\vec{x}$ and $\vec{y}$ are respectively the inputs and outputs of the network, $F_{pre}(\vec{x})$ is a non-strict monotone formula giving the *precondition* on the inputs, and $F_{post}(\vec{x}, \vec{y})$ is a non-strict monotone formula giving the *postcondition* on the inputs and outputs of the network.

A neural network is *certified* with respect to a property if for all inputs respecting the precondition we have that the relation between inputs and outputs given by the postcondition is respected as well, as formally stated by the following definition.

Definition 2.3. Let N be a ReLU neural network with n inputs and m outputs, and $P(\vec{x}, \vec{y})$ a property. Then, N is *certified* for the property if for all inputs $\vec{\mathbf{x}} \in \mathbb{R}^n$ such that $F_{pre}(\vec{\mathbf{x}})$ is true, we have that $F_{post}(\vec{\mathbf{x}}, f_N(\vec{\mathbf{x}}))$ is true.

To give some examples, consider the following properties for the neural network N in Figure 1:

(1) *when the input is less than or equal to zero, the output is less than or equal to zero;*
(2) *when the input is greater than or equal to minus two, the output is less than or equal to the input minus six.*

Property (1) can be formalized by the formula $x \leq 0 \Rightarrow y \leq 0$. N is not certified for this property, since for $\vec{\mathbf{x}} = -3$ we have that $f_N(\vec{\mathbf{x}}) = 10$, and thus the postcondition $y \leq 0$ is false.

Property (2) can be formalized by the formula $x \geq -2 \Rightarrow y \leq x - 6$, that can be rewritten as $-x - 2 \leq 0 \Rightarrow y - x + 6 \leq 0$ to formalize the precondition and the postcondition in the form given by Definition 2.1. Notice that this property requires the use of both the input x and the output y in the postcondition to be encoded correctly. N is not certified for this property either, since for $\vec{\mathbf{x}} = -2$ we have that $f_N(\vec{\mathbf{x}}) = -2$, and thus the postcondition $y \leq x - 6$ is false.

We are usually interested in certifying a network for more than one property, and we would like to guarantee that all of them are respected at the same time. We can formalize this scenario by considering *sets of properties*:

$$\mathcal{P} = \{P_1(\vec{x}, \vec{y}), \ldots, P_h(\vec{x}, \vec{y})\},$$

where each $P_i(\vec{x}, \vec{y})$ is a property according to Definition 2.2. Then, Definition 2.3 of certified neural network can be extended to consider sets of properties as follows.

Definition 2.4. A ReLU neural network N is *certified* for a set of properties $\mathcal{P} = \{P_1(\vec{x}, \vec{y}), \ldots, P_h(\vec{x}, \vec{y})\}$ if it is certified for every $P_i(\vec{x}, \vec{y})$.

Figure 2 provides a geometrical interpretation of the situation under analysis. Figure 2 should be read as follows:

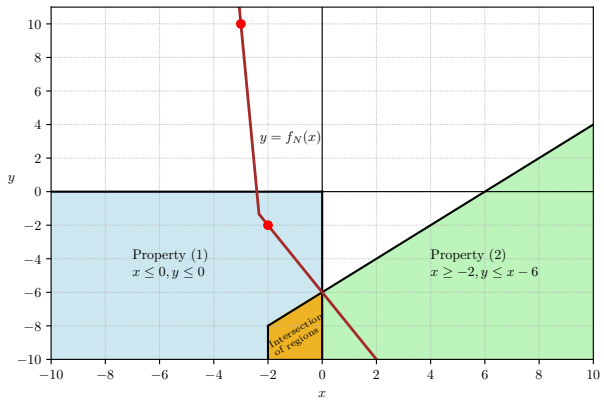


Fig. 2. Geometrical interpretation of neural network (brown line), constraint regions (shaded areas), and counterexamples (red dots).

(1) the piece-wise linear function in brown is the graphical representation of the neural network N form Figure 1;
(2) the regions where both precondition and postconditions of property (1) and of property (2) are true are the blue and the green ones, respectively. The orange region is the intersection of the two. Black thick lines highlighting the borders of the regions.
(3) Whenever a precondition of a property is false, that property holds trivially; so, in this example, N trivially satisfies property (1) for every $x > 0$, and (2) for every $x < -2$. Consequently, the picture shows that N satisfies both property (1) and property (2) for all $x \geq 0$ (where both postconditions are true), and also when $-\frac{41}{17} \leq x < -2$, where Property (2) is trivially true;
(4) Some counterexamples (red dots) can refer to more than (possibly all) properties; e.g., $(-3, 10)$ is a counterexample to both Property (1) and Property (2);
(5) Some counterexamples only refer to a single property; e.g., $(-2, -2)$ is a counterexample to Property (2) but not to Property (1).

The fact that counterexamples may impact different subsets of properties will play a central when we discuss our repair technique in Section 4.

2.3 Formal Verification of ReLU Neural Networks

Formal verification is the process of mathematically proving whether a system satisfies a given specification or not. If the system meets the specification, a proof of correctness is provided; if it does not, the method produces a *counterexample*, that is, a concrete scenario where the specification is violated.

When applied to ReLU neural networks, with specifications given as the properties defined in Section 2.2, the formal verification problem can be rephrased as a constraint satisfaction problem comprising two set of constraints:

(1) a set of constraints encoding the network;
(2) a set of constraints encoding violations of the property we would like to hold.

Solutions to such a CSP provide counterexamples to the property, that is, input-output pairs that do not respect the required conditions. An unsatisfiable CSP gives a proof that the network is certified for the property.

We will use the same theory of linear real arithmetic we used in Section 2.2 to formalize properties to build the CSP encoding the neural network verification problem described above.

Let N be a ReLU neural network with d layers (where each layer k had ℓ_k neurons), n inputs $\vec{x} = (x_1, \ldots, x_n)$, and m outputs $\vec{y} = (y_1, \ldots, y_m)$. To encode the network into an LRA formula we proceed as follows.

- For each neuron $v_{i,j}$ of N we add two auxiliary variables $v_{i,j}^{in}$ and $v_{i,j}^{out}$ that represent respectively the input and the output of the neuron. To improve readability, we use $x_1, \ldots, x_n$ instead of the auxiliary variables $v_{1,1}^{in}, \ldots, v_{1,n}^{in}$ for the input layer, and $y_1, \ldots, y_m$ instead of the auxiliary variables $v_{d,1}^{out}, \ldots, v_{d,m}^{out}$ for the output layer.
- Input neurons simply forward the input to the output. Hence, for each input neuron we add the conjunct:
$$x_j = v_{1,j}^{out}$$
- For each hidden neuron $v_{i,j}$, where $1 < i < d$, we add the following conjuncts:
$$v_{i,j}^{in} = b_{i,j} + \sum_{k=1}^{\ell_{i-1}} w(v_{i-1,k}, v_{i,j}) \cdot v_{i-1,k}^{out}$$
$$v_{i,j}^{out} = \max(0, v_{i,j}^{in})$$
where $a = \max(0, b)$ can be encoded in LRA as: $a \geq 0 \wedge a \geq b \wedge (a = 0 \vee a = b)$.
- For each output neuron $v_{d,j}$, we add the following conjuncts:
$$v_{d,j}^{in} = b_{d,j} + \sum_{k=1}^{\ell_{d-1}} w(v_{d-1,k}, v_{d,j}) \cdot v_{d-1,k}^{out}$$
$$y_j = v_{d,j}^{in}$$

The LRA formula $F_N(\vec{x}, \vec{y})$ encoding the network N is built by taking the conjunction of all the equations introduced above and then existentially quantifying all auxiliary variables $v_{i,j}^{in}$ and $v_{i,j}^{out}$. After this step, the only free variables in the formula are the inputs $\vec{x} = (x_1, \ldots, x_n)$ and the outputs $\vec{y} = (y_1, \ldots, y_m)$ of the network. Thus, for every pair $(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \mathbb{R}^n \times \mathbb{R}^m$, we have:

$$F_N(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \text{ if and only if } \vec{\mathbf{y}} = f_N(\vec{\mathbf{x}}).$$

To give an example, if we apply the LRA encoding to the network in Figure 1 we obtain the following formula:

$$\begin{aligned} F_N(x, y) := \exists v_{1,1}^{out}, v_{2,1}^{in}, v_{2,1}^{out}, v_{2,2}^{in}, v_{2,2}^{out}, v_{3,1}^{in}. \big(& \\ & x = v_{1,1}^{out} \wedge \\ & v_{2,1}^{in} = 8 + 2v_{1,1}^{out} \wedge v_{2,1}^{out} = \max(0, v_{2,1}^{in}) \wedge \\ & v_{2,2}^{in} = -7 - 3v_{1,1}^{out} \wedge v_{2,2}^{out} = \max(0, v_{2,2}^{in}) \wedge \\ & v_{3,1}^{in} = 2 - v_{2,1}^{out} + 5v_{2,2}^{out} \wedge \\ & y = v_{3,1}^{in} \big) \end{aligned}$$

Given a property $P(\vec{x}, \vec{y})$, to complete the encoding of the network verification problem we have to show how to encode violations of the property within an LRA formula. We can do this by conjoining the negation of the property with the formula $F_N(\vec{x}, \vec{y})$ encoding the network, as formally stated by the following theorem.

Theorem 2.5. *Let N be a ReLU neural network and $P(\vec{x}, \vec{y})$ a property. Then, N is* certified *for $P(\vec{x}, \vec{y})$ if and only if the following formula is unsatisfiable:*

$$F_{N,P}(\vec{x}, \vec{y}) := F_N(\vec{x}, \vec{y}) \wedge F_{pre}(\vec{x}) \wedge \neg F_{post}(\vec{x}, \vec{y}),$$

where $F_{pre}(\vec{x})$ and $F_{post}(\vec{x}, \vec{y})$ are the precondition and postcondition of the property, and $F_N(\vec{x}, \vec{y})$ is the formula encoding the network.

Proof. We prove the theorem by showing that $F_{N,P}(\vec{x}, \vec{y})$ is satisfiable if and only if the network N is not certified the property $P(\vec{x}, \vec{y})$.

($\Rightarrow$) Suppose that $F_{N,P}(\vec{x}, \vec{y})$ is satisfiable, and let $\vec{\mathbf{x}} \in \mathbb{R}^n$, $\vec{\mathbf{y}} \in \mathbb{R}^m$ be valuations for the free variables that make the formula true. By the conjunct $F_N(\vec{x}, \vec{y})$ we have that $\vec{\mathbf{y}} = f_N(\vec{\mathbf{x}})$. By the conjunct $F_{pre}(\vec{x})$ we have that the valuation $\vec{\mathbf{x}}$ satisfies the precondition of the property, while from the conjunct $\neg F_{post}(\vec{x}, \vec{y})$ we have that the postcondition is violated, thus proving that the network N is not certified for the property $P(\vec{x}, \vec{y})$.

($\Leftarrow$) Now suppose that N is not certified for $P(\vec{x}, \vec{y})$. Then there must exists a valuation $\vec{\mathbf{x}} \in \mathbb{R}^n$ for the inputs such that $F_{pre}(\vec{\mathbf{x}})$ is true, and $F_{post}(\vec{\mathbf{x}}, f_N(\vec{\mathbf{x}}))$ is false. We now show that if we take $\vec{\mathbf{y}} = f_N(\vec{\mathbf{x}})$, we obtain a valuation the free variables of $F_{N,P}(\vec{x}, \vec{y})$ that satisfies the formula. First of all, since $\vec{\mathbf{y}} = f_N(\vec{\mathbf{x}})$, we have that the conjunct $F_N(\vec{x}, \vec{y})$ is true. Then, by the choice of input valuation we have that the conjunct $F_{pre}(\vec{x})$ is true. Finally, since $F_{post}(\vec{\mathbf{x}}, f_N(\vec{\mathbf{x}}))$ is false, we have that the conjunct $\neg F_{post}(\vec{x}, \vec{y})$ is true.

□

Finally, we recall from Definition 2.4 that N is certified for a set of properties $\mathcal{P} = \{P_1(\vec{x}, \vec{y}), \ldots, P_h(\vec{x}, \vec{y})\}$ if it is certified for each one of them. This allows us to check separately unsatisfiability of the formulae $F_{N,P_1}(\vec{x}, \vec{y}), \ldots, F_{N,P_h}(\vec{x}, \vec{y})$.

Natural Extensions

The encoding presented above naturally extends to any activation function expressible in LRA. As an example, consider the *Leaky ReLU*, a commonly used piecewise-linear activation function that introduces a small positive gradient α for negative inputs:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

The output value $v_{i,j}^{out}$ of neuron $v_{i,j}$ can be encoded in LRA as:

$$(v_{i,j}^{in} \geq 0 \wedge v_{i,j}^{out} = v_{i,j}^{in}) \vee (v_{i,j}^{in} \leq 0 \wedge v_{i,j}^{out} = \alpha v_{i,j}^{in})$$

This extension also allows piecewise-linear activation functions to be associated with input and output neurons whenever needed.

Our encoding also supports non-linear activation functions in a limited setting, specifically when they appear only in the output layer. In this case, the non-linear activation functions of the output layer can be replaced with LRA-definable ones, and the property can be rewritten accordingly, ensuring that certification of the modified network with respect to the modified property implies certification of the original network with respect to the original property. As an example, consider the sigmoid activation function:

$$\sigma(x) := \frac{1}{1 + e^{-x}}$$

When the network output has a binary interpretation, where positive values correspond to "true" and negative ones to "false", $\sigma(x)$ can be replaced by the identity function $\mathrm{id}(x) = x$. This substitution is valid because both σ and id preserve the sign of the input: a positive input yields a positive output, and a negative input yields a negative output. Therefore, the binary interpretation of the network is preserved, and certification of the modified network implies certification of the original one.

Note that this approach is only valid for output neurons. For hidden neurons, the value of the activation function must be propagated to the next layer, which cannot be done exactly when the encoding language is restricted to linear constraints.

3 Realizability and Epsilon-Certification

In the previous sections, we introduced the formal language used to define properties as well as what it means for a neural network to be certified with respect to a property or a set of properties. In our setting, a neural network is a function from the input domain $\mathbb{R}^n$ to the output domain $\mathbb{R}^m$. Therefore, we focus on properties that are *realizable*, i.e., such that there exists at least one function that satisfies them.

Definition 3.1 (Realizability of properties). Let $P(\vec{x}, \vec{y})$ be a property. We say that $P(\vec{x}, \vec{y})$ is *realizable* if there exists a function $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ such that for all inputs $\vec{\mathbf{x}} \in \mathbb{R}^n$ where $F_{pre}(\vec{\mathbf{x}})$ is true, $F_{post}(\vec{\mathbf{x}}, f(\vec{\mathbf{x}}))$ is true. Furthermore, we say that a set of properties $\mathcal{P}$ is realizable if there exists a function $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ that simultaneously realizes all properties in $\mathcal{P}$.

From the universal approximation theorems for neural networks, we know that, given a closeness criterion $\varepsilon > 0$, if a property $P(\vec{x}, \vec{y})$ is realizable by a function $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, then there exists a neural network N with enough neurons to approximate f within ε (Park et al. 2021). Universal approximation theorems are limit theorems: they do not guarantee that a network computing exactly f is eventually found. As a result, they do not guarantee that a network certified for $P(\vec{x}, \vec{y})$ can be eventually found either, but only that we can get "arbitrary close" to the satisfaction of $P(\vec{x}, \vec{y})$.

To formalize this approximating nature of neural networks, and state precisely the problem that our framework addresses, we introduce the notion of *ε-certification* of a neural network with respect to a property. The intuition behind this notion is that we can relax the postcondition of a property to consider a given ε tolerance.

Definition 3.2 (ε-relaxation of a property). Let $P(\vec{x}, \vec{y})$ be a property and let $\varepsilon > 0$. The ε-relaxation of $P(\vec{x}, \vec{y})$, in symbols $P^{\varepsilon}(\vec{x}, \vec{y})$, is obtained by replacing every linear constraint $b + \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{m} d_i y_i \leq 0$ in the postcondition $F_{post}(\vec{x}, \vec{y})$ with $b + \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{m} d_i y_i < \varepsilon$.

Going back to the example properties of Section 2.2, let us recall the formulae encoding properties (1) and (2):

$$P_1(\vec{x}, \vec{y}) := \overbrace{x \leq 0}^{F_{pre}(x)} \Rightarrow \overbrace{y \leq 0}^{F_{post}(x,y)} \qquad P_2(\vec{x}, \vec{y}) := \overbrace{-x - 2 \leq 0}^{F_{pre}(x)} \Rightarrow \overbrace{y - x + 6 \leq 0}^{F_{post}(x,y)}$$

Given $\varepsilon > 0$, we have that the ε-relaxations of these two properties are:

$$P_1^{\varepsilon}(\vec{x}, \vec{y}) := \overbrace{x \leq 0}^{F_{pre}(x)} \Rightarrow \overbrace{y < \varepsilon}^{F^{\varepsilon}_{post}(x,y)} \qquad P_2^{\varepsilon}(\vec{x}, \vec{y}) := \overbrace{-x - 2 \leq 0}^{F_{pre}(x)} \Rightarrow \overbrace{y - x + 6 < \varepsilon}^{F^{\varepsilon}_{post}(x,y)}$$

When $\varepsilon \leq 10$, the network N fails to satisfy either ε-relaxation of the properties. Indeed, for input $\vec{\mathbf{x}} = (-3)$, the output $\vec{\mathbf{y}} = f_N(\vec{\mathbf{x}}) = (10)$ violates both postconditions.

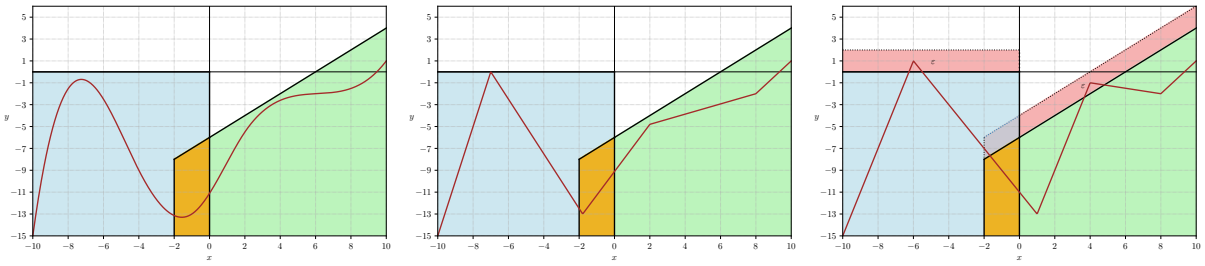
We say that a neural network is *ε-certified* for a property if it satisfies the ε-relaxation of the property.

Definition 3.3 (ε-certified neural network). Let N be a ReLU neural network, $P(\vec{x}, \vec{y})$ be a property, and let $\varepsilon > 0$. Then, N is *ε-certified* for $P(\vec{x}, \vec{y})$ if for all inputs $\vec{\mathbf{x}} \in \mathbb{R}^n$ such that $F_{pre}(\vec{\mathbf{x}})$ is true, we have that $F^{\varepsilon}_{post}(\vec{\mathbf{x}}, f_N(\vec{\mathbf{x}}))$ is true. N is ε-certified for a set of properties $\mathcal{P} = \{P_1(\vec{x}, \vec{y}), \ldots, P_h(\vec{x}, \vec{y})\}$, if it is ε-certified for every $P_i(\vec{x}, \vec{y})$.

We are now ready to state the (synthesis) problem we want to solve.

Definition 3.4 (Problem statement). Given:

- a dataset $\mathcal{D}$ of input-output pairs disjointly partitioned in training, validation, and test,



(a) Realizable set of properties. (b) Certified ReLU neural network. (c) ε-Certified ReLU neural network.

Fig. 3. Realizability, certification, and ε-certification.

- a realizable set of properties $\mathcal{P} = \{P_1(\vec{x}, \vec{y}), \ldots, P_h(\vec{x}, \vec{y})\}$,
- a closeness criterion $\varepsilon > 0$, and
- a loss function $\mathcal{L}$,

synthesize a ReLU neural network N such that:

- N is ε-certified for all properties in $\mathcal{P}$;
- $\mathcal{L}$ minimized on $\mathcal{D}$.

Since the ε-relaxation of a property is a rewrite of the property itself, we can easily extend Theorem 2.5 to provide a solution to the ε-certification problem for ReLU neural networks.

Theorem 3.5. *Let N be a ReLU neural network, $P(\vec{x}, \vec{y})$ be a property, and $\varepsilon > 0$ be a closeness criterion. Then, N is ε-certified for $P(\vec{x}, \vec{y})$ if and only if the following formula is unsatisfiable:*

$$F^{\varepsilon}_{N,P}(\vec{x}, \vec{y}) := F_N(\vec{x}, \vec{y}) \wedge F_{pre}(\vec{x}) \wedge \neg F^{\varepsilon}_{post}(\vec{x}, \vec{y}),$$

where $F_N(\vec{x}, \vec{y})$ and $F_{pre}(\vec{x})$ are the same of those in Theorem 2.5, whereas $F_{post}(\vec{x}, \vec{y})$ is the ε-relaxation of the postcondition of $P(\vec{x}, \vec{y})$.

The proof is the same of that given for Theorem 2.5 with respect to $F^{\varepsilon}_{post}(\vec{x}, \vec{y})$. Notice that $F^{\varepsilon}_{N,P}$ is an LRA formula whose set of solutions are counterexamples proving that N falsifies $P^{\varepsilon}(\vec{x}, \vec{y})$. More importantly, notice that the negation of a formula with strict inequalities — *here*, $F^{\varepsilon}_{post}(\vec{x}, \vec{y})$ — is a monotone formula with no strict inequalities, and thus $F^{\varepsilon}_{N,P}(\vec{x}, \vec{y})$ is a whole non-strict monotone formula.

Figure 3 provides a graphical illustration of the concepts of realizability, certification, and ε-certification (note that the examples therein are unrelated to the network in Figure 1). The colored areas represent the regions defined by the individual properties (cyan and green areas), their intersection (orange area), and the ε-relaxation (red areas):

- Figure 3a shows that there exists a function satisfying all properties (realizability);
- Figure 3b shows that the depicted ReLU neural network satisfies all properties (certification);
- Figure 3c shows that the depicted ReLU neural network satisfies the ε-relaxation of all properties (ε-certification).

4 Counter Example Guided Inductive Synthesis

The framework proposed in this paper is based on the Counter Example Guided Inductive Synthesis (CEGIS) workflow shown in Figure 4, originally proposed in (Solar-Lezama et al. 2006) for the automated synthesis of programs, which we adapt and extend here to the neural network setting.

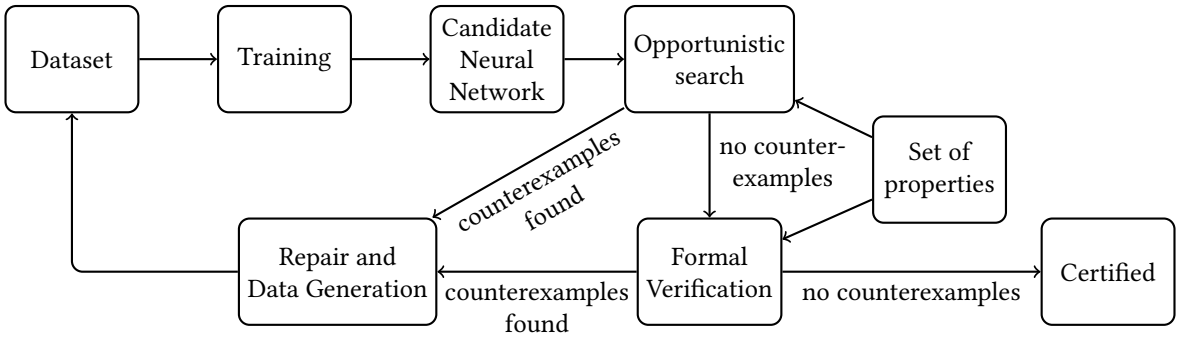


Fig. 4. The CEGIS loop employed for the synthesis of certified neural networks.

Our CEGIS workflow begins by generating a candidate neural network from the initial dataset. A subsequent opportunistic search phase attempts to find counterexamples by feeding random inputs to the network and checking whether the corresponding outputs satisfy the properties. If counterexamples are found, the workflow proceeds to the repair and data generation phase. Only when the opportunistic phase produces no counterexamples the network is formally verified. If formal verification finds no counterexamples either, the candidate network is certified. Otherwise, the counterexamples found, either by the opportunistic search or by formal verification, are used to augment the dataset and generate a new candidate network. This loop repeats until a candidate network is formally proved to satisfy all properties.

Given a set of properties $\mathcal{P}$ and a dataset $\mathcal{D}$ of input-output pairs $(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \mathbb{R}^n \times \mathbb{R}^m$, our CEGIS approach starts by partitioning $\mathcal{D}$ into training, validation, and test set. We then create a neural network N as a result of some training algorithm that works on train and validation sets only. After that, we verify N with respect to the ε-relaxation of each property $P_i(\vec{x}, \vec{y}) \in \mathcal{P}$, first opportunistically by generating a random batch of inputs. If the opportunistic search find no counterexamples, we formally verify the ε-relaxation of each property $P_i(\vec{x}, \vec{y}) \in \mathcal{P}$ in isolation by building the constraint satisfaction problem defined in Theorem 3.5. The output of the verification part can lead to two situations:

(1) all formulae are unsatisfiable; this means that N is ε-certified for all properties in $\mathcal{P}$;
(2) some formulae are satisfiable; this means that N violates one or more properties and that we have a counterexample for each violated property.

In the former case, the process is complete and we have proof that N respects all properties in $\mathcal{P}$ *for all possible values in the input domain*, and not only for the inputs contained in the *finite* dataset $\mathcal{D}$. In the latter case, we use the counterexamples provided by the verification phase to extend $\mathcal{D}$ with new input-output pairs $(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ that satisfy the desired properties and then we proceed with the next iteration of the CEGIS workflow. These new pairs are obtained by first replacing the component $\vec{\mathbf{y}}$ in the counterexample with a $\vec{\mathbf{y}}_{\mathbf{rep}}$ such that $(\vec{\mathbf{x}}, \vec{\mathbf{y}}_{\mathbf{rep}})$ satisfies all properties in $\mathcal{P}$, and then sampling around it to increase the number of pairs satisfying all properties that are added to $\mathcal{D}$. The following subsection details our data generation technique.

Counterexample Repair for Data Generation

Let N be a ReLU neural network and let $\mathcal{P}$ be a set of properties. Suppose that the verification phase of the CEGIS loop found the network not to be ε-certified for some property $P_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \mathcal{P}$, and let $(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ be the found counterexample.

To correct the behavior of the network, we want to build another input output pair $(\vec{\mathbf{x}}_{\mathbf{rep}}, \vec{\mathbf{y}}_{\mathbf{rep}})$ such that:

Algorithm 1 Neural Network verification and counterexample repair

Input: Feed-forward ReLU Neural Network N, set of properties $\mathcal{P}$, batch size $z > 0$ for opportunistic search, tolerance $\varepsilon > 0$.
Output: $\emptyset$ if N is ε-certified for $\mathcal{P}$, set of repaired counterexamples $\mathcal{S}$ otherwise.

$C \leftarrow \emptyset$
for each property $P(\vec{x}, \vec{y}) \in \mathcal{P}$ **do**
  opportunistic_search $\leftarrow$ *false*
  for $i = 1, \ldots, z$ **do** ▷ opportunistic verification phase
    Generate an input $\vec{\mathbf{x}}$ and compute $\vec{\mathbf{y}} = N(\vec{\mathbf{x}})$
    if $P^{\varepsilon}(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ is violated **then**
      $C \leftarrow C \cup \{(\vec{\mathbf{x}}, \vec{\mathbf{y}})\}$
      opportunistic_search $\leftarrow$ *true*
    end if
  end for
  if *opportunistic_search* = *false* **then** ▷ formal verification phase
    $(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \leftarrow \textsc{Verifier}(N, P(\vec{x}, \vec{y}), \varepsilon)$
    if $(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \neq$ NULL **then**
      $C \leftarrow C \cup \{(\vec{\mathbf{x}}, \vec{\mathbf{y}})\}$
    end if
  end if
end for
if $C = \emptyset$ **then**
  return $\emptyset$ ▷ No counterexamples, network is ε-certified.
end if
$\mathcal{S} \leftarrow \emptyset$
Let $\vec{s} := (s_1, \ldots, s_m)$ be a vector of fresh variables.
for each counterexample $(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in C$ **do**
  $\mathcal{R} \leftarrow \emptyset$ ▷ Initialize set of constraints
  for each property $P(\vec{x}, \vec{y}) \in \mathcal{P}$ **do** ▷ Find all properties whose preconditions are satisfied by the counterexample
    Let $P(\vec{x}, \vec{y}) = F_{pre}(\vec{x}) \Rightarrow F_{post}(\vec{x}, \vec{y})$
    if $F_{pre}(\vec{\mathbf{x}})$ is true **then**
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{F_{post}(\vec{\mathbf{x}}, \vec{\mathbf{y}})[\vec{\mathbf{x}}/\vec{x}, \vec{\mathbf{y}} \oplus \vec{s}/\vec{y}]\}$
    end if
  end for
  Let $\vec{\mathbf{s}}$ be the solution to the following mixed-integer quadratic programming (MIQP) problem: ▷ $\vec{\mathbf{s}}$ exists (Remark 1)
    Minimize $\sum_{i=1}^{m} s_i^2$
    subject to all constraints in $\mathcal{R}$.
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\vec{\mathbf{x}}, \vec{\mathbf{y}} \oplus \vec{\mathbf{s}})\}$
end for
return $\mathcal{S}$

(1) the value of the input variables do not change: that is, $\vec{\mathbf{x}}_{\mathbf{rep}} := \vec{\mathbf{x}}$;
(2) $(\vec{\mathbf{x}}_{\mathbf{rep}}, \vec{\mathbf{y}}_{\mathbf{rep}})$ satisfies *all properties in* $\mathcal{P}$.

Notice that in the second condition we require the new pair to respect not only the property $P_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \mathcal{P}$ from which the counterexample comes from, but also all the properties in $\mathcal{P}$ *in their non-relaxed form*. This is needed, on the one hand to prevent the repair phase from adding points to the dataset that satisfy $P_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \mathcal{P}$ but violate some other property $P_j(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \mathcal{P}$ with $i \neq j$, and on the other hand to make the learning more robust, and avoid adding to the dataset points that are just below the ε tolerance. When the neural network violates more than one property in $\mathcal{P}$, we repeat the repair process for each property that is violated.

Algorithm 1 shows how we can find counterexamples and repair them for all the properties that are violated in every iteration of the CEGIS loop. In the pseudocode, Verifier$(N, P(\vec{x}, \vec{y}), \varepsilon)$ executes the formal verification of N with respect to a single property $P(\vec{x}, \vec{y})$ and a tolerance $\varepsilon > 0$. This function returns NULL if the network is ε-certified for $P(\vec{x}, \vec{y})$, and a counterexample otherwise. In line 28, $F_{post}(\vec{\mathbf{x}}, \vec{\mathbf{y}})[\vec{\mathbf{x}}/\vec{x}, \vec{\mathbf{y}} \oplus \vec{s}/\vec{y}]$ is obtained from $F_{post}(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ by substituting $\mathbf{x_i} \in \vec{\mathbf{x}}$ for the corresponding input variable in $x_i \in \vec{x}$ and the expression $\mathbf{y}_i + s_i \in \vec{\mathbf{y}} \oplus \vec{s}$ for each output variable in $\vec{y}_i$, with $\oplus$ representing the element-wise vector addition.

Finally, to speed up the next training phase we generate further points to add to $\mathcal{D}$ by sampling around $(\vec{\mathbf{x}}_{\mathbf{rep}}, \vec{\mathbf{y}}_{\mathbf{rep}})$ and adding a sample $(\vec{\mathbf{x}}', \vec{\mathbf{y}}')$ to the dataset $\mathcal{D}$ only if $(\vec{\mathbf{x}}', \vec{\mathbf{y}}')$ satisfies all properties in $\mathcal{P}$. This way we avoid solving multiple mixed-integer quadratic programming (MIQP) problems to generate additional data. We recall that solving MIQP is NP-hard.

Remark 1. *The repair phase is always possible (=MIQP problems always admit a solution) because the set of properties $\mathcal{P}$ is realizable.*

As a result, the following invariant holds:

Invariant 1. *The repair phase generates input-output pairs that always satisfy all properties in $\mathcal{P}$.*

Recall from the end of Section 2.3 that, for the neural network in Figure 1, we provided counterexamples for both property (1) and (2). By following Algorithm 1, we show how the repairs are done.

Consider the counterexample $(\vec{\mathbf{x}}, \vec{\mathbf{y}}) = (-3, 10)$. Since -3 makes true the precondition of property (1) only, we repair it by solving:

$$\begin{array}{l} \min s^2 \\ \text{subject to} \\ y \leq 0[-3/x, 10 + s/y] \end{array} \quad \Rightarrow \quad \begin{array}{l} \min s^2 \\ \text{subject to} \\ 10 + s \leq 0 \end{array}$$

The optimal solution is $\vec{\mathbf{s}} = (-10)$, leading to the repaired example $(\vec{\mathbf{x}}_{\mathbf{rep}}, \vec{\mathbf{y}}_{\mathbf{rep}}) = (-3, 0)$.

Consider the counterexample $(\vec{\mathbf{x}}, \vec{\mathbf{y}}) = (-2, -2)$. Since -2 makes true both the preconditions of property (1) and property (2), we repair it by solving:

$$\begin{array}{l} \min s^2 \\ \text{subject to} \\ y \leq 0[-2/x, -2 + s/y] \\ y - x + 6 \leq 0[-2/x, -2 + s/y] \end{array} \Rightarrow \begin{array}{l} \min s^2 \\ \text{subject to} \\ -2 + s \leq 0 \\ (-2 + s) - (-2) + 6 \leq 0 \end{array} \Rightarrow \begin{array}{l} \min s^2 \\ \text{subject to} \\ -2 + s \leq 0 \\ 6 + s \leq 0 \end{array}$$

The optimal solution is $\vec{s} = (-6)$, leading to the repaired example $(\vec{\mathbf{x}}_{\mathbf{rep}}, \vec{\mathbf{y}}_{\mathbf{rep}}) = (-2, -8)$.

Figure 5 provides a geometric interpretation of these repairs. The repair phase starts from the red dots that depict the two counterexamples under analysis. The solutions to the related MIQP problems lead to the black dots that satisfy both property (1) and property (2). Finally, the gray dots are the result of selecting further points that satisfy the properties by sampling around the black ones.

Our property language represents a good balance between expressiveness and computational tractability. Indeed, the language is expressive enough to encode neural networks with ReLU and other piecewise-linear activation functions, making it at least as expressive as the network under certification. Furthermore, we support disjunctions in both preconditions and postconditions, and allow for defining properties over multiple input-output pairs. We therefore believe that the language has non-trivial expressiveness and covers the majority of realistic specifications for the network architectures we considered. Additionally, linear arithmetic is well-understood, decidable, and supported by a wide variety of solvers, making it both theoretically sound and practically convenient.

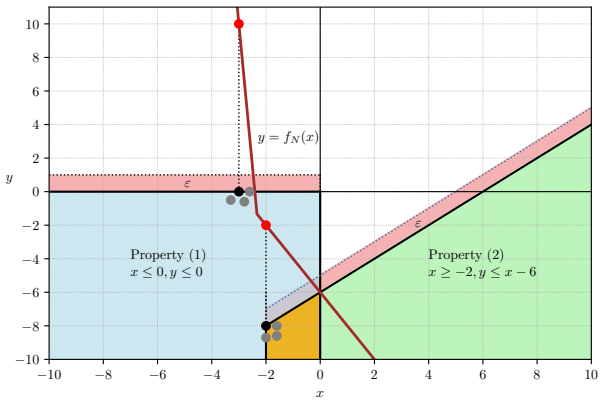


Fig. 5. Geometric interpretation of repair.

Besides non-linear properties, the most significant class of properties that our language cannot express are those involving strict inequalities. This is, however, a deliberate design choice driven by the repair phase: allowing strict inequalities would undermine the existence of optimal solutions to the MIQP solved during repair. To see why, consider the following property for Figure 1:

$$P_{strict}(x, y) := x < 0 \Rightarrow y > 0$$

Given the counterexample $(-2, -2)$, the repair phase would create the following MIQP problem:

$$\begin{array}{lcll} \min s^2 & & \min s^2 & & \min s^2 \\ \text{subject to} & \Rightarrow & \text{subject to} & \Rightarrow & \text{subject to} \\ y > 0[-2/x, -2 + s/y] & & -2 + s > 0 & & s > 2 \end{array}$$

which admits no optimal solution. Strict inequalities are therefore fundamentally incompatible with our optimization-based repair strategy.

Computational Complexity

We now summarize the computational complexity of the three main components of our framework.

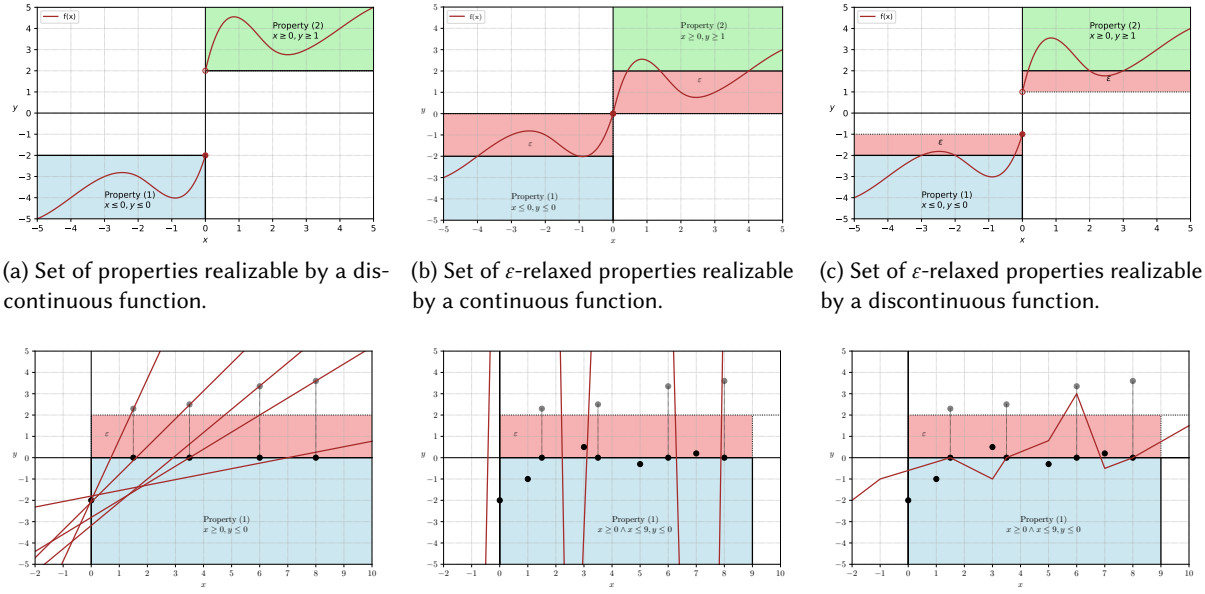
Training. The training phase grows linearly with the number of CEGIS iterations, as at each iteration a constant number of counterexamples is added to the dataset. This controlled growth ensures that the overhead introduced at each iteration remains predictable.

Formal Verification. Verification of ReLU neural networks is NP-complete, as already established in (Katz, Barrett, et al. 2017).

Repair. The repair phase is NP-hard, as it reduces to a Mixed-Integer Quadratic Program (MIQP), whose feasibility is at least as hard as that of MILP, a well-known NP-complete problem. Since both MILP and MIQP are optimization problems rather than decision problems, we characterize their complexity in terms of hardness only.

5 Termination

The overall goal of the CEGIS loop is to synthesize a function (in the form of a neural network) that respects a given set of properties. Since there are uncountably many possible inputs for the function, the CEGIS loop may go on forever, producing more and more counterexamples, even when a function respecting the property



(a) Set of properties realizable by a discontinuous function.

(b) Set of ε-relaxed properties realizable by a continuous function.

(c) Set of ε-relaxed properties realizable by a discontinuous function.

(d) Unbounded domain for a property.

(e) Learning a non-Lipschitz function.

(f) Non-zero error on repaired counterexamples.

Fig. 6. Possible situations that may break termination of the approach.

exists. Since the property is a quantifier-free LRA formula, we know we can restrict the search to piecewise-linear functions. We know that ReLU neural networks compute piecewise linear functions, and thus, from recent universal approximation results for ReLU neural networks (Park et al. 2021), we can conclude that a network big enough to respect the property exists. In this section we identify the conditions we need to prove termination of the approach we have discussed so far. Intuitively, we want to prove that after we process a finite number of counterexamples, our approach stops by returning a ReLU neural network N that is ε-certified for a set of properties $\mathcal{P}$. To do so, we need to make the following assumptions (we provide geometric interpretations in Figure 6, where again, the examples there are not related to the network in Figure 1).

(1) *The set of properties $\mathcal{P}$ is realizable by a continuous function.* Consider the properties $P_1(\vec{x}, \vec{y}) := x \leq 0 \implies y \leq -2$ and $P_2(\vec{x}, \vec{y}) := x \geq 0 \implies y \geq 2$. The geometric representation of the corresponding regions is depicted in Figure 6a. This set of properties is realizable only by functions that are discontinuous at $x = 0$. ReLU functions are continuous, therefore they are incapable of modeling discontinuities. Considering the ε-relaxations of the properties does not help in this case. For $\varepsilon > 2$ a continuous function satisfying the ε-relaxations of the properties exists (as shown in Figure 6b). However, our ε-certification approach assumes that ε may be arbitrarily small, and thus, since no continuous function can satisfies the properties for $\varepsilon < 2$ (as shown in Figure 6c), we restrict our attention to properties that are realizable by continuous function.
(2) *The input domain is bounded*. This is to avoid situations where increasingly large counterexamples are generated indefinitely due to the learning of a different neural network. Figure 6d shows such a situation.

(3) *The hypothesis space of the learning algorithm is a uniformly Lipschitz class of functions.* This means that there exists a single constant $\ell > 0$ such that every neural network function $f_N : \mathbb{R}^n \mapsto \mathbb{R}^m$ satisfies

$$\|f_N(\mathbf{x}) - f_N(\mathbf{y})\| \leq \ell \|\mathbf{x} - \mathbf{y}\| \quad \text{for every } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n.$$

If no such a uniform constant exists, we may end up in situations where at every iteration the learned function grows faster and faster, as depicted in Figure 6e. We acknowledge that this is a strong requirement: standard training procedures do not generally guarantee a uniform Lipschitz bound, particularly when training components such as batch normalization or dropout are used, and this therefore represents a practical limitation of this theoretical result. Nevertheless, this assumption can be satisfied in practice when Lipschitz control is explicitly enforced, for example through certified Lipschitz architectures (Araujo et al. 2023; Hu et al. 2024), making this result applicable in settings where such guarantees are available.

(4) *Zero learning error on the repaired counterexamples.* Typically, a reasonable assumption is that training outputs a network with zero error on all points in the training. Here, we need a weaker assumption to avoid situations where the training phase cannot update the weights and biases of the network such that the error is zero on the repaired counterexamples (as graphically depicted by Figure 6f).

Under the assumptions listed above, we can prove the following result.

Theorem 5.1. *Let $\mathcal{P} := \{P_1(\vec{x}, \vec{y}), \ldots, P_h(\vec{x}, \vec{y})\}$ be a set of properties, N be a feed-forward ReLU neural network, and let $\ell \in \mathbb{R}$. If:*

(1) $\mathcal{P}$ is realizable by a continuous function,
(2) the input domain is bounded,
(3) at every CEGIS iteration the training algorithm produces a ReLU neural network N so that:
  (a) the error of N on the repaired counterexamples is zero,
  (b) f_N is ℓ-Lipschitz,

then, the CEGIS approach discussed in Section 4 terminates after processing a finite number of counterexamples.

Proof. We start the proof by observing that we can assume, without loss of generality, that every property in $\mathcal{P}$ is of the special form

$$P(\vec{x}, \vec{y}) := F_{pre}(\vec{x}) \Rightarrow D_1 \vee \cdots \vee D_h,$$

where $D_1, \ldots, D_h$ are conjunctions of linear constraints (i.e., polyhedra). If this is not the case, we can take any property $F_{pre}(\vec{x}) \Rightarrow F_{post}(\vec{x}, \vec{y})$, convert the postcondition into disjunctive normal form (DNF) to obtain a formula is in the desired form. Notice that, since we do not have negations in our $F_{post}(\vec{x}, \vec{y})$, this conversion still outputs a monotone non-strict boolean formula over the same linear constraints of $F_{post}(\vec{x}, \vec{y})$.

This initial assumption would allow us to prove the theorem one property at a time, by showing that the CEGIS loop can generate only finitely many counterexamples violating a property. Consider now a property $P(\vec{x}, \vec{y}) := F_{pre}(\vec{x}) \Rightarrow D_1 \vee \cdots \vee D_h$, and suppose that at some iteration of the CEGIS loop the formal verification phase returns a counterexample $(\vec{\mathbf{x}}_{\mathtt{cex}}, \vec{\mathbf{y}}_{\mathtt{cex}})$ violating the property. By the implicative form of the property, we have that $\vec{\mathbf{x}}_{\mathtt{cex}}$ satisfies the precondition $F_{pre}(\vec{x})$, while $(\vec{\mathbf{x}}_{\mathtt{cex}}, \vec{\mathbf{y}}_{\mathtt{cex}})$ violates all disjuncts in the postcondition. The repaired counterexample $(\vec{\mathbf{x}}_{\mathtt{rep}}, \vec{\mathbf{y}}_{\mathtt{rep}})$ is such that $\vec{\mathbf{x}}_{\mathtt{rep}}$ still satisfies the precondition $F_{pre}(\vec{x}, \vec{y})$ (since input variables are not changed), while $(\vec{\mathbf{x}}_{\mathtt{rep}}, \vec{\mathbf{y}}_{\mathtt{rep}})$ must satisfy at least one of the disjunct in the postcondition. Let D be one the disjuncts satisfied by the repaired counterexample. Such a D has the following form:

$$D = \bigwedge_{k=1}^{z} \left(b_k + \sum_{i=1}^{n} c_{k,i} x_i + \sum_{i=1}^{m} d_{k,i} y_i \leq 0 \right)$$

where $z > 0$ is the number of linear constraints in it. Notice that we can assume that all variables $\vec{x}, \vec{y}$ appears in all the linear constraints. If some variable does not appear in the constraints, then we can add that variable to the linear expression by setting the corresponding coefficient to zero.

We show that there exists a value $\delta > 0$ that depends only on ε, the Lipschitz constant ℓ and the constants of the constraints in D, such that for all inputs in the hypercube of radius δ centered in $\vec{\mathbf{x}}_{\mathtt{rep}}$, the output of the network satisfies the relaxed property $P^{\epsilon}(\vec{x}, \vec{y})$. Let K be the maximum absolute value of all constants appearing in D and $\delta > 0$. Let $\vec{\mathbf{x}}_{\mathtt{rep}} = (\mathbf{x_1}, \ldots, \mathbf{x_n})$, $\vec{\mathbf{y}}_{\mathtt{rep}} = (\mathbf{y}_1, \ldots, \mathbf{y}_m)$ and let $\vec{\mathbf{x}}'$ be a point inside the hypercube of radius δ centered in $\vec{\mathbf{x}}_{\mathtt{rep}}$, that is, a point such that

$$|\mathbf{x_i} - \mathbf{x_i}'| < \delta \text{ for every } i \in \{1, \ldots, n\}.$$

In the following, we denote with $\mathrm{sgn}(c)$ the sign of a constant c and with $f_N(\vec{\mathbf{x}})_i$ the i-th component of the function f_N under input $\vec{\mathbf{x}}$. Given a constraint $b_k + \sum_{i=1}^{n} c_{k,i} x_i + \sum_{i=1}^{m} d_{k,i} y_i \leq 0$ in D, we have that the chain of following inequalities holds:

$$\begin{aligned}
& b_k + \sum_{i=1}^{n} c_{k,i} \mathbf{x_i}' + \sum_{i=1}^{m} d_{k,i} f_N(\vec{\mathbf{x}}')_i \\
& \leq b_k + \sum_{i=1}^{n} c_{k,i} (\mathbf{x_i} + \mathrm{sgn}(c_{k,i})\delta) + \sum_{i=1}^{m} d_{k,i} f_N(\vec{\mathbf{x}}')_i && \text{since } |\mathbf{x_i} - \mathbf{x_i}'| < \delta \\
& \leq b_k + \sum_{i=1}^{n} c_{k,i} (\mathbf{x_i} + \mathrm{sgn}(c_{k,i})\delta) + \sum_{i=1}^{m} d_{k,i} (\mathbf{y}_i + \mathrm{sgn}(d_{k,i})\ell\delta) && \text{since } f_N \text{ is } \ell\text{-Lipschitz} \\
& \leq b_k + \sum_{i=1}^{n} c_{k,i} \mathbf{x_i} + \sum_{i=1}^{n} |c_{k,i}|\delta + \sum_{i=1}^{m} d_{k,i} \mathbf{y}_i + \sum_{i=1}^{m} |d_{k,i}|\ell\delta \\
& \leq b_k + \Big(\sum_{i=1}^{n} c_{k,i} \mathbf{x_i} \Big) + nK\delta + \Big(\sum_{i=1}^{m} d_{k,i} \mathbf{y}_i \Big) + mK\ell\delta && \text{by definition of } K
\end{aligned}$$

Since $(\vec{\mathbf{x}}_{\mathtt{rep}}, \vec{\mathbf{y}}_{\mathtt{rep}})$ satisfies D, we have that $b_k + \sum_{i=1}^{n} c_{k,i} \mathbf{x}_i + \sum_{i=1}^{m} d_{k,i} \mathbf{y}_i \leq 0$, and thus that if $nK\delta + mK\ell\delta < \varepsilon$, then all points $(\vec{\mathbf{x}}', f_N(\vec{\mathbf{x}}'))$ satisfy the ε-relaxation of the linear constraint.

Thus, by choosing a $\delta < \frac{\varepsilon}{K(n+m\ell)}$ we are guaranteed that all points in the hypercube satisfy P as well. Since the value of δ is fixed and does not depend on the network function f_N, we have that this would be true also for all subsequent iterations of the CEGIS loop. Since the input domain is bounded, we have that the polyhedron D is bounded as well, and thus that after a finite number of iterations no counterexample can violate D anymore.

Since the number of properties in $\mathcal{P}$ is finite, and since every property has a finite number of disjuncts in the postcondition, we have that the CEGIS loop must terminate after processing a finite number of counterexamples. □

6 Properties on Multiple Input-Output Pairs

In the previous sections, we addressed the problem of automatically synthesizing certified ReLU neural networks for sets of properties defined over single input-output pairs. In this section we discuss a possible generalization for sets of properties defined over multiple input-output pairs.

First of all, why do we need properties on multiple input-output pairs? To answer this question, let us get back to our neural network in Figure 1. The language of properties defined in Definition 2.2 is incapable to formalize this third property:

(3) *for any two inputs-output pairs, if the difference between the first and the second input is less than or equal to one, then the first output is less than or equal to zero or the second output is greater than or equal to two.*

Property (3) can be formalized by the formula $x_1 - x_2 \leq 1 \Rightarrow y_1 \leq 0 \vee y_2 \geq 2$, where x_1 and y_1 are the input and the output of one copy of the neural network, while x_2 and y_2 are input and output of a second copy of the network. It turns out that the neural network in Figure 2 is not certified for Property (3). A counterexample is the pair of inputs $\mathbf{x_1} = -3$ and $\mathbf{x}_2 = -2$ that leads to $f_N(\mathbf{x_1}) = 10$ and $f_N(\mathbf{x}_2) = -2$ making the precondition true and the postcondition false.

Formulae

The language for specifying properties on multiple input-output pairs extends as follows.

Definition 6.1 (Property on multiple input-output pairs). Given a neural network N, a *property* over $k \geq 1$ input-output pairs $(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_k, \vec{y}_k)$ is any formula that can be written as follows:

$$P(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k) := F_{pre}(\vec{x}_1, \ldots, \vec{x}_k) \Rightarrow F_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$$

where $\vec{x}_1, \ldots, \vec{x}_k$ and $\vec{y}_1, \ldots, \vec{y}_k$ are k copies of the input and output vectors of the network, $F_{pre}(\vec{x}_1, \ldots, \vec{x}_k)$ is a non-strict monotone formula giving the *precondition* on inputs, and $F_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$ is a non-strict monotone formula giving the *postcondition* on input-output pairs.

Certified Networks

The notion of certification extends as follows.

Definition 6.2. Let N be a ReLU neural network with n inputs and m outputs, and $P(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$ a property over $k \geq 1$ input-output pairs. Then, N is *certified* for the property if for all inputs $\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_k \in \mathbb{R}^{k \times n}$ such that $F_{pre}(\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_k)$ is true, we have that $F_{post}(\vec{\mathbf{x}}_1, f_N(\vec{\mathbf{x}}_1), \ldots, \vec{\mathbf{x}}_k, f_N(\vec{\mathbf{x}}_k))$ is true.

We consider sets of properties

$$\mathcal{P} := \{P_1(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_{k_1}, \vec{y}_{k_1}), \ldots, P_h(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_{k_h}, \vec{y}_{k_h})\}$$

where each property P_i may be defined over a different number $k_i \geq 1$ of input-output pairs. A network is certified with respect to $\mathcal{P}$ if it is certified for all properties in $\mathcal{P}$.

Opportunistic Search

The opportunistic search phase follows the same structure as Algorithm 1, with one key difference: at each iteration, rather than generating a single input, it generates a batch of inputs of the same size as the number of input-output pairs on which the property is defined. The corresponding network outputs are then computed, and the ε-relaxation of the property under analysis is checked against the entire batch.

Formal Verification

Since a property can now be defined over $k \geq 1$ input-output pairs, we need to include k copies of the encoding of N in the encoding of the formal verification problem.

Theorem 6.3. *Let N be a ReLU neural network and $P(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$ a property over $k \geq 1$ input-output pairs. Then, N is* certified *for $P(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$ if and only if the following formula is unsatisfiable:*

$$F_{N,P}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k) := \bigwedge_{j=1}^{k} \Big(F_N(\vec{x}_j, \vec{y}_j)\Big) \wedge F_{pre}(\vec{x}_1, \ldots, \vec{x}_k) \wedge \neg F_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k),$$

where $F_{pre}(\vec{x}_1, \ldots, \vec{x}_k)$ *and* $F_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$ *are the precondition and postcondition of the property, and each* $F_N(\vec{x}_j, \vec{y}_j)$ *is the formula encoding the network with respect to the* j^{th} *input-output pair* $(\vec{x}_j, \vec{y}_j)$.

The proof is similar to that of Theorem 2.5. And once again, since we consider sets of properties $\mathcal{P} := \{P_1(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_{k_1}, \vec{y}_{k_1}), \ldots, P_h(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_{k_h}, \vec{y}_{k_h})\}$, we can run formal verification separately for each property to check if the network is certified for all properties.

Epsilon-Certification

The ε-certification part extends as follows. When we have to identify variables inside multiple vectors $\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k$ of input-output variables, we will use the notation $x_{i,j}$ and $y_{i,j}$ to denote respectively the j-th component of the input vector $\vec{x}_i$, and the j-th component of the output vector $\vec{y}_i$.

Definition 6.4 (ε-relaxation of a property). Let $P(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$ be a property over $k \geq 1$ pairs and let $\varepsilon > 0$. The ε-relaxation of $P(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$, in symbols $P^{\varepsilon}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$, is obtained by replacing every linear constraint $b + \sum_{j=1}^{k}\sum_{i=1}^{n} c_{j,i}x_{j,i} + \sum_{j=1}^{k}\sum_{i=1}^{m} d_{j,i}y_{j,i} \leq 0$ in the postcondition $F_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$ with $b + \sum_{j=1}^{k}\sum_{i=1}^{n} c_{j,i}x_{j,i} + \sum_{j=1}^{k}\sum_{i=1}^{m} d_{j,i}y_{j,i} < \varepsilon$.

The notion of ε-certified neural network remains the same of that given in Definition 3.3 with respect to properties over multiple input-output pairs. Finally, Theorem 6.3, can be extended to ε-certification as follows.

Theorem 6.5. *Let* N *be a ReLU neural network,* $P(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$ *be a property over* $k \geq 1$ *input-output pairs, and* $\varepsilon > 0$ *be a closeness criterion. Then,* N *is* ε*-certified for* $P(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$ *if and only if the following formula is unsatisfiable:*

$$F^{\varepsilon}_{N,P}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k) := \bigwedge_{j=1}^{k} \Big(F_N(\vec{x}_j, \vec{y}_j)\Big) \wedge F_{pre}(\vec{x}_1, \ldots, \vec{x}_k) \wedge \neg F^{\varepsilon}_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k),$$

where $F_{pre}(\vec{x}_1, \ldots, \vec{x}_k)$ *and each* $F_N(\vec{x}_j, \vec{y}_j)$ *are the same of those in Theorem 6.5, whereas* $F^{\varepsilon}_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$ *is the* ε*-relaxation of the postcondition* $F_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$.

As an example of the above definitions, given $\varepsilon > 0$, the ε-relaxation of property (3) is:

$$P^{\varepsilon}_3(\vec{x}_1, \vec{y}_1, \vec{x}_2, \vec{y}_2) := \overbrace{x_{1,1} - x_{2,1} - 1 \leq 0}^{F_{pre}(\vec{x}_1, \vec{x}_2)} \Rightarrow \overbrace{y_{1,1} < \varepsilon \vee -y_{2,1} + 2 < \varepsilon}^{F_{post}(\vec{x}_1, \vec{y}_1, \vec{x}_2, \vec{y}_2)}$$

It turns out that, if $\varepsilon \leq 4$, then N does not respect the ε-relaxation of $P_3(\vec{x}_1, \vec{y}_1, \vec{x}_2, \vec{y}_2)$, since for $\vec{\mathbf{x}}_1 = (-3)$ and $\vec{\mathbf{x}}_2 = (-2)$ we have that $\vec{\mathbf{y}}_1 = f_N(\vec{\mathbf{x}}_1) = 10$ and $\vec{\mathbf{y}}_2 = f_N(\vec{\mathbf{x}}_2) = -2$, violating the postcondition.

Repair

In this section, we show that when repairing counterexamples for properties on multiple input-output pairs following the method discussed in Section 4, Invariant 1 no longer holds in general. In what follows, we highlight a few possible scenarios and countermeasures. We continue the discussion with respect to Property (3).

Scenario 1. Consider this counterexample: $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1), (\vec{\mathbf{x}}_2, \vec{\mathbf{y}}_2) = (0, 1), (1, 1)$. A first attempt to repair such a counterexample needs to extend the approach in Section 4 to consider as many copies of s variables as the number of input-output pairs. This leads to the following MIQP problem:

$$\begin{array}{ll} \min s^2_{1,1} + s^2_{2,1} & \\ \text{subject to} & \Rightarrow \quad \min s^2_{1,1} + s^2_{2,1} \\ y_{1,1} \leq 0 \vee y_{2,1} \geq 2[0/x_{1,1}, 1 + s_{1,1}/y_{1,1}, 1/x_{2,1}, 1 + s_{2,1}/y_{2,1}] & \quad\quad\; \text{subject to} \\ & \quad\quad\; s_{1,1} \leq -1 \vee s_{2,1} \geq 1 \end{array}$$

Since the MIQP problem involves disjunctive constraints the optimal solution is not necessarily unique. Indeed, there are two optimal solutions: $(\vec{\mathbf{s}}_1, \vec{\mathbf{s}}_2) = (-1, 0)$ and $(\vec{\mathbf{s}}_1, \vec{\mathbf{s}}_2) = (0, 1)$. For the purpose of our method any of these solutions is fine. Let consider the first. Then,

- $\vec{\mathbf{x}}_{\mathbf{rep}_1} := \vec{\mathbf{x}}_1 = 0$ and $\vec{\mathbf{y}}_{\mathbf{rep}_1} := \vec{\mathbf{y}}_1 \oplus \vec{\mathbf{s}}_1 = (\mathbf{y}_{1,1} + \mathbf{s}_{1,1}) = 1 - 1 = 0$;
- $\vec{\mathbf{x}}_{\mathbf{rep}_2} := \vec{\mathbf{x}}_2 = 1$ and $\vec{\mathbf{y}}_{\mathbf{rep}_2} := \vec{\mathbf{y}}_2 \oplus \vec{\mathbf{s}}_2 = (\mathbf{y}_{2,1} + \mathbf{s}_{2,1}) = 1 + 0 = 1$;

and thus $(\vec{\mathbf{x}}_{\mathbf{rep}_1}, \vec{\mathbf{y}}_{\mathbf{rep}_1}) = (0, 0)$ and $(\vec{\mathbf{x}}_{\mathbf{rep}_2}, \vec{\mathbf{y}}_{\mathbf{rep}_2}) = (1, 1)$ are the repaired pairs that we add to the dataset whose current state becomes the following:

$$\mathcal{D} := \{\ldots, (0, 0), (1, 1)\}$$

Now, Invariant 1 is violated. By taking $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1) = (1, 1)$ and $(\vec{\mathbf{x}}_2, \vec{\mathbf{y}}_2) = (0, 0)$ the precondition of Property (3) is true while the postcondition is false.

The problem with this repair is caused by the combination of three factors:

(1) the counterexample for Property (3) is made of two input-output pairs instead of a single one;
(2) the MIQP problem finds a solution where Property (3) is respected only if $\vec{\mathbf{x}}_1 = 0$ and $\vec{\mathbf{x}}_2 = 1$;
(3) Property (3) must be true also when we exchange the two input-output pairs, that is, when $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1) = (1, 1)$ and $(\vec{\mathbf{x}}_2, \vec{\mathbf{y}}_2) = (0, 0)$.

This shows that to guarantee Invariant 1 we need to enforce the properties *for all possible permutations* of the input-output pairs in the counterexample.

Scenario 2. Let us continue the example discussed in above. Assume that, despite having introduced violations of Property (3) in the dataset, we retrained the network and that at the next verification phase we find this other counterexample: $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1), (\vec{\mathbf{x}}_2, \vec{\mathbf{y}}_2) = (2, 1), (4, 1)$. If we repair this counterexample as before, we need to solve:

$$\begin{array}{ll} \min s_{1,1}^2 + s_{2,1}^2 & \\ \text{subject to} & \Rightarrow \quad \min s_{1,1}^2 + s_{2,1}^2 \\ y_{1,1} \leq 0 \vee y_{2,1} \geq 2[2/x_{1,1}, 1 + s_{1,1}/y_{1,1}, 4/x_{2,1}, 1 + s_{2,1}/y_{2,1}] & \quad \text{subject to} \\ & \quad s_{1,1} \leq -1 \vee s_{2,1} \geq 1 \end{array}$$

whose optimal solutions are: $(\vec{\mathbf{s}}_1, \vec{\mathbf{s}}_2) = (-1, 0)$ and $(\vec{\mathbf{s}}_1, \vec{\mathbf{s}}_2) = (0, 1)$. Suppose we generate the repaired pair from the second solution. Then,

- $\vec{\mathbf{x}}_{\mathbf{rep}_1} := \vec{\mathbf{x}}_1 = 2$ and $\vec{\mathbf{y}}_{\mathbf{rep}_1} := \vec{\mathbf{y}}_1 \oplus \vec{\mathbf{s}}_1 = (\mathbf{y}_{1,1} + \mathbf{s}_{1,1}) = 1 + 0 = 1$;
- $\vec{\mathbf{x}}_{\mathbf{rep}_2} := \vec{\mathbf{x}}_2 = 4$ and $\vec{\mathbf{y}}_{\mathbf{rep}_2} := \vec{\mathbf{y}}_2 \oplus \vec{\mathbf{s}}_2 = (\mathbf{y}_{2,1} + \mathbf{s}_{2,1}) = 1 + 1 = 2$;

and thus we add to the dataset the pair $(\vec{\mathbf{x}}_{\mathbf{rep}_1}, \vec{\mathbf{y}}_{\mathbf{rep}_1}) = (2, 1)$ and $(\vec{\mathbf{x}}_{\mathbf{rep}_2}, \vec{\mathbf{y}}_{\mathbf{rep}_2}) = (4, 2)$:

$$\mathcal{D} = \{\ldots, (0, 0), (1, 1), (2, 1), (4, 2)\}$$

In this case we have that all permutations of $(2, 1)$ and $(4, 2)$ respect Property (3). However, Invariant 1 is now violated also by taking $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1) = (2, 1)$ and $(\vec{\mathbf{x}}_2, \vec{\mathbf{y}}_2) = (1, 1)$, since the precondition of Property (3) is true while the postcondition is false.

Hence, this scenario shows that subsequent verification phases may introduce violations to the property by combining the new counterexamples with the repaired counterexamples of the previous steps. This shows that to guarantee Invariant 1 we need to enforce the properties *for all possible permutations with repetitions* of the repaired counterexamples from *all previous steps*, and not only from the last one.

Generalized Verification-and-Repair Algorithm

Algorithm 2 presents a generalized verify-and-repair algorithm that works not only in the two scenarios presented before, but also considers arbitrary sets of properties, where each property may be defined on a different number of pairs. Beside the neural network N, the set of properties $\mathcal{P}$ and the tolerance ε, it also takes as input the set $\mathcal{S}_{\mathcal{D}}$ that contains all repaired counterexamples from the previous iterations of the CEGIS loop.

Algorithm 2 Generalized neural network verification and counterexample repair

Input: Feed-forward ReLU Neural Network N, set of properties $\mathcal{P}$ where each property is defined over an arbitrary number of pairs, batch size $z > 0$ for opportunistic search, previous set of repaired counterexamples $\mathcal{S}_{\mathcal{D}}$, tolerance $\varepsilon > 0$.
Output: $\emptyset$ if N is ε-certified for $\mathcal{P}$, set of repaired counterexamples $\mathcal{S}$ otherwise.

$\mathcal{S} \leftarrow \mathcal{S}_{\mathcal{D}}$
for each property $P \in \mathcal{P}$ **do**
  Let k be the number of pairs over which P is defined.
  opportunistic_search $\leftarrow$ *false*
  for $i = 1, \ldots, z$ **do** ▷ opportunistic verification phase
    Generate k inputs $\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_k$ and compute $\vec{\mathbf{y}}_1 = N(\vec{\mathbf{x}}_1), \ldots, \vec{\mathbf{y}}_k = N(\vec{\mathbf{x}}_k)$
    if $P^{\varepsilon}(\vec{\mathbf{x}}_1, \ldots \vec{\mathbf{x}}_k, \vec{\mathbf{y}}_1, \ldots \vec{\mathbf{y}}_k)$ is violated **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1), \ldots, (\vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k)\}$
      opportunistic_search $\leftarrow$ *true*
    end if
  end for
  if *opportunistic_search* = *false* **then** ▷ formal verification phase
    $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k) \leftarrow \textsc{Verifier}(N, P(\vec{x}_1, \ldots, \vec{x}_k, \vec{y}_1, \ldots, \vec{y}_k), \varepsilon)$
    if $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k) \neq$ NULL **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1), \ldots, (\vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k)\}$
    end if
  end if
end for
if $\mathcal{S} = \mathcal{S}_{\mathcal{D}}$ **then**
  return $\emptyset$ ▷ No new counterexamples, network is ε-certified.
end if
For each $(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \mathcal{S}$, let $\vec{s}_{(\vec{\mathbf{x}}, \vec{\mathbf{y}})} = (\vec{s}_{(\vec{\mathbf{x}}, \vec{\mathbf{y}}),1}, \ldots, \vec{s}_{(\vec{\mathbf{x}}, \vec{\mathbf{y}}),m})$
$\mathcal{R} \leftarrow \emptyset$
for each property $P \in \mathcal{P}$ **do**
  Let k be the number of pairs over which P is defined.
  Let $P(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k) = F_{pre}(\vec{x}_1, \ldots, \vec{x}_k) \Rightarrow F_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)$
  for each permutation $((\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1), \ldots, (\vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k)) \in \Pi(\mathcal{S}, k)$ **do**
    if $F_{pre}(\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_k)$ is true **then**
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{F_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k)[\vec{\mathbf{x}}_1/\vec{x}_1, \vec{\mathbf{y}}_1 \oplus \vec{s}_{(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1)}/\vec{y}_1, \ldots, \vec{\mathbf{x}}_k/\vec{x}_k, \vec{\mathbf{y}}_k \oplus \vec{s}_{(\vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k)}/\vec{y}_k]\}$
    end if
  end for
end for
Let $\vec{\mathbf{s}}$ be the solution to the following mixed-integer quadratic problem (MIQP): ▷ $\vec{\mathbf{s}}$ exists (Remark 1)
  Minimize $\sum_{(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \mathcal{S}} \sum_{i=1}^{m} \left(s_{(\vec{\mathbf{x}}, \vec{\mathbf{y}}),i}\right)^2$
  subject to all constraints in $\mathcal{R}$.
$\mathcal{S} \leftarrow \{(\vec{\mathbf{x}}, \vec{\mathbf{y}} \oplus \vec{\mathbf{s}}_{(\vec{\mathbf{x}}, \vec{\mathbf{y}})}) \mid (\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \mathcal{S}\}$
return $\mathcal{S}$

The operations involving $\textsc{Verifier}(N, P(\vec{x}, \vec{y}), \varepsilon)$ and substitutions in F_{post} formulae remain as in Algorithm 1, just extended to deal with multiple input-output pairs.

In contrast with Algorithm 1, Algorithm 2 generates a single MIQP problem that repairs all properties in a single shot and also may change previously repaired pairs. To do so, it first computes the set $\mathcal{S}$ by adding to $\mathcal{S}_{\mathcal{D}}$ the new counterexamples obtained by verifying every property in $\mathcal{P}$. If no new counterexamples are found, then the network is ε-certified and the algorithm terminates with success.

When new counterexamples are found, the algorithm generates the MIQP problem by adding a vector of fresh variables $\vec{s}_i = (s_{i,1}, \ldots, s_{i,m})$ for each $(\vec{\mathbf{x}}_i, \vec{\mathbf{y}}_i) \in \mathcal{S}$.

Given the set $\mathcal{S}$ of input-output pairs, $\Pi(\mathcal{S}, k)$ is the set of all permutations with repetitions of k elements taken from $\mathcal{S}$. For each property in $\mathcal{P}$, Algorithm 2 checks every permutation $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1), \ldots, (\vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k) \in \Pi(\mathcal{S}, k)$, where k is the number of input-output pairs of the properties. If the vector of values $(\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_k)$ satisfies the precondition of the property, then it adds a constraint to the MIQP problem that forces the values of the variables $\vec{s}_1, \ldots, \vec{s}_k$ to respect the postcondition of the property. The result consists in the set of repaired pairs $\mathcal{S}$, that will replace all the data points added so far by the CEGIS loop to the dataset, and will be the set $\mathcal{S}_{\mathcal{D}}$ for the next call of Algorithm 2.

Since the property is realizable, the MIQP problem built by Algorithm 2 admits a solution. However, the MIQP problem has size $\Theta(|\mathcal{S}|^k)$ that increases with the number of CEGIS iterations. This shows that, while correct, the general algorithm does not scale.

Sampling is also complicated in this context, since the sampled pairs may violate some property when considered in combination with some other pair in $\mathcal{S}$. A way to prevent this problem is to sample around the pairs in $\mathcal{S}$, add the corresponding pairs $(\vec{\mathbf{x}}', \vec{\mathbf{y}}')$ to $\mathcal{S}$, run again Algorithm 2 with $\mathcal{S}_{\mathcal{D}} = \mathcal{S}$, and use the resulting $\mathcal{S}$ for the next call of Algorithm 2.

The general algorithm applied to scenario 2. Let us get back to Scenario 2. The state of the dataset before doing verification and repair was:

$$\mathcal{D} = \{\ldots, (0,0), (1,1)\},$$

and the counterexample to Property (3) was $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1), (\vec{\mathbf{x}}_2, \vec{\mathbf{y}}_2) = (2,1), (4,1)$. Therefore, after the verification phase we have that:

$$\mathcal{S} := \{\underbrace{(0,0)}_{\vec{s}_1}, \underbrace{(1,1)}_{\vec{s}_2}, \underbrace{(2,1)}_{\vec{s}_3}, \underbrace{(4,1)}_{\vec{s}_4}\}$$

and the set of all permutations of two elements taken from $\mathcal{S}$ is:

$$\begin{aligned}\Pi(\mathcal{S}, 2) := \{&((0,0),(1,1)), \quad ((0,0),(2,1)), \quad ((0,0),(4,1)), \quad ((1,1),(0,0)), \quad ((1,1),(2,1)), \quad ((1,1),(4,1)),\\ &((2,1),(0,0)), \quad ((2,1),(1,1)), \quad ((2,1),(4,1)), \quad ((4,1),(0,0)), \quad ((4,1),(1,1)), \quad ((4,1),(2,1))\}.\end{aligned}$$

By removing all permutations that do not satisfy the precondition of Property (3) we obtain the following set of permutations:

$$\begin{aligned}\{&((0,0),(1,0)), \quad ((0,0),(2,1)), \quad ((0,0),(4,1)), \quad ((1,1),(0,0)), \quad ((1,1),(2,1)), \quad ((1,1),(4,1)),\\ &\cancel{((2,1),(0,0))}, \quad ((2,1),(1,1)), \quad ((2,1),(4,1)), \quad \cancel{((4,1),(0,0))}, \quad \cancel{((4,1),(1,1))}, \quad \cancel{((4,1),(2,1))}\}.\end{aligned}$$

Putting all together, Algorithm 2 builds the following MIQP problem:

$$
\begin{array}{lcl}
\min \sum_{i=1}^{|\mathcal{S}|} s_{i,1}^2 & & \min \sum_{i=1}^{|\mathcal{S}|} s_{i,1}^2 \\
\text{subject to} & \Rightarrow & \text{subject to} \\
y_{1,1} \leq 0 \vee y_{2,1} \geq 2[0/x_{1,1}, 0 + s_{1,1}/y_{1,1}, 1/x_{2,1}, 0 + s_{2,1}/y_{2,1}] & & s_{1,1} \leq 0 \vee s_{2,1} \geq 2 \\
y_{1,1} \leq 0 \vee y_{2,1} \geq 2[0/x_{1,1}, 0 + s_{1,1}/y_{1,1}, 2/x_{2,1}, 1 + s_{3,1}/y_{2,1}] & & s_{1,1} \leq 0 \vee s_{3,1} \geq 1 \\
y_{1,1} \leq 0 \vee y_{2,1} \geq 2[0/x_{1,1}, 0 + s_{1,1}/y_{1,1}, 4/x_{2,1}, 1 + s_{4,1}/y_{2,1}] & & s_{1,1} \leq 0 \vee s_{4,1} \geq 1 \\
y_{1,1} \leq 0 \vee y_{2,1} \geq 2[1/x_{1,1}, 1 + s_{2,1}/y_{1,1}, 0/x_{2,1}, 0 + s_{1,1}/y_{2,1}] & & s_{2,1} \leq -1 \vee s_{1,1} \geq 2 \\
y_{1,1} \leq 0 \vee y_{2,1} \geq 2[1/x_{1,1}, 1 + s_{2,1}/y_{1,1}, 2/x_{2,1}, 1 + s_{3,1}/y_{2,1}] & & s_{2,1} \leq -1 \vee s_{3,1} \geq 1 \\
y_{1,1} \leq 0 \vee y_{2,1} \geq 2[1/x_{1,1}, 1 + s_{2,1}/y_{1,1}, 4/x_{2,1}, 1 + s_{4,1}/y_{2,1}] & & s_{2,1} \leq -1 \vee s_{4,1} \geq 1 \\
y_{1,1} \leq 0 \vee y_{2,1} \geq 2[2/x_{1,1}, 1 + s_{3,1}/y_{1,1}, 1/x_{2,1}, 0 + s_{2,1}/y_{2,1}] & & s_{3,1} \leq -1 \vee s_{2,1} \geq 2 \\
y_{1,1} \leq 0 \vee y_{2,1} \geq 2[2/x_{1,1}, 1 + s_{3,1}/y_{1,1}, 4/x_{2,1}, 1 + s_{4,1}/y_{2,1}] & & s_{3,1} \leq -1 \vee s_{4,1} \geq 1
\end{array}
$$

for which an optimal solution is: $(\vec{\mathbf{s}}_1, \vec{\mathbf{s}}_2, \vec{\mathbf{s}}_3, \vec{\mathbf{s}}_4) = (0, -1, -1, 0)$. Then, the repaired pairs in $\mathcal{S}$ are as follows:

- $(0, 0)$: $\vec{\mathbf{x}}_{\mathbf{rep}_1} := \vec{\mathbf{x}}_1 = 0$ and $\vec{\mathbf{y}}_{\mathbf{rep}_1} := \vec{\mathbf{y}}_1 \oplus \vec{\mathbf{s}}_1 = (\mathbf{y}_{1,1} + \mathbf{s}_{1,1}) = 0 + 0 = 0$;
- $(1, 1)$: $\vec{\mathbf{x}}_{\mathbf{rep}_2} := \vec{\mathbf{x}}_1 = 1$ and $\vec{\mathbf{y}}_{\mathbf{rep}_2} := \vec{\mathbf{y}}_2 \oplus \vec{\mathbf{s}}_2 = (\mathbf{y}_{2,1} + \mathbf{s}_{2,1}) = 1 - 10$;
- $(2, 1)$: $\vec{\mathbf{x}}_{\mathbf{rep}_3} := \vec{\mathbf{x}}_1 = 2$ and $\vec{\mathbf{y}}_{\mathbf{rep}_3} := \vec{\mathbf{y}}_3 \oplus \vec{\mathbf{s}}_3 = (\mathbf{y}_{3,1} + \mathbf{s}_{3,1}) = 1 - 1 = 0$;
- $(4, 1)$: $\vec{\mathbf{x}}_{\mathbf{rep}_4} := \vec{\mathbf{x}}_1 = 4$ and $\vec{\mathbf{y}}_{\mathbf{rep}_3} := \vec{\mathbf{y}}_4 \oplus \vec{\mathbf{s}}_4 = (\mathbf{y}_{4,1} + \mathbf{s}_{4,1}) = 1 + 0 = 1$;

and thus we add update the dataset as follows:

$$\mathcal{D} = \{\ldots, \underbrace{(0,0), (1,0), (2,0), (4,1)}_{\text{Globally repaired pairs}}\}$$

And Invariant 1 holds again, since all possible permutations of two elements in $\{(0,0), (1,0), (2,0), (4,1)\}$ respect Property 3.

Termination

The discussion on the repair algorithm shows that when properties are defined over multiple input-output pairs we cannot analyze single input-ouput pairs in isolation. Hence, the proof strategy we used to prove termination in Section 5 cannot be applied anymore. Several aspects make a termination analysis non-trivial in the multiple input-output pairs setting. In the single input-output pair case, termination relies on the assumption of zero training error on the repaired counterexamples. Combined with the ε-tolerance, this guarantees that at every iteration a portion of the input space, of at least a fixed minimum size, is cleared of counterexamples. A key property of the single pair case is that repaired data persists in the dataset across all future CEGIS iterations, ensuring that previously cleared portions of the input space remain counterexample-free.

In the multiple input-output pairs case, however, this persistence is not guaranteed, as the repair phase can modify also input-output pairs that were added in previous phases (or even from the initial dataset). This means that we cannot guarantee anymore that previously cleared portions remains counterexample-free, potentially preventing termination. This fundamental mismatch makes it difficult to provide termination guarantees in the general multi-pair setting.

Computational Complexity

Single-pair properties are a special case of multi-pair properties, so the complexity lower bounds established earlier carries over directly. The extensions discussed in this section introduce additional overhead, which we summarize below.

Training The complexity remains the same as in the single-pair case. Even though part of the dataset is selectively replaced in the repair phase, the increase in the size of the dataset is still linear with the number of iterations.

Formal Verification The encoding of the property requires adding one copy of the neural network per input-output pair to the constraint satisfaction problem. However, the number of copies is a constant value that is bounded by the length of the property. Thus, the verification problem remains NP-complete.

Repair The set of constraints of the MIQP grows exponentially with the number of repaired pairs added so far, as it must account for all permutations of fixed length over those pairs. The complexity is therefore at least NP-hard, and a tighter characterization is left for future investigation.

Permutations In general we consider permutations with repetitions leading Algorithm 2 to iterate over $|\mathcal{S}|^k$ permutations for a given $\Pi(\mathcal{S}, k)$. This bound can sometimes be reduced if we consider properties for which reflexivity holds. For example, consider $P(\vec{x}_1, \vec{y}_1, \vec{x}_2, \vec{y}_2) := x_1 \leq x_2 \Rightarrow y_1 \leq y_2$ (i.e., non-decreasing monotonicity of the function implemented by the neural network in Figure 1). Clearly, taking two pairs $(\mathbf{x}_1, \mathbf{y}_1)$ and $(\mathbf{x}_2, \mathbf{y}_2)$ with $\mathbf{x}_1 = \mathbf{x}_2$ and thus $f_N(\mathbf{x}_1) = \mathbf{y}_1 = \mathbf{y}_2 = f_N(\mathbf{x}_2)$ would be useless for the repair phase. In such cases, Algorithm 2 can be adapted to consider only distinct pairs, reducing the number of permutations (without repetitions) to $\Theta\left(\frac{|\mathcal{S}|!}{(|\mathcal{S}|-k)!}\right)$ for a given $\Pi(\mathcal{S}, k)$.

7 Soft Constraint Acceleration

We discuss how we can reduce the total number of CEGIS iterations needed to synthesize a certified network. We proceed by injecting soft-constraints into the training phase to try to enforce the property as early as possible. The approach follows the usual idea of adding a term to the loss function that penalizes input-output pairs that violate the property. In this section we provide an automated method for the general computation of such a term. We start by defining the violation score (*VS*) of an arbitrary non-strict monotone formula.

Definition 7.1 (VS). Let $F(\vec{x})$ be any non-strict monotone formula. Let $\vec{\mathbf{x}}$ be any valuation of $\vec{x}$. The violation score of $F(\vec{x})$ with respect to $\vec{\mathbf{x}}$, in symbols $VS(F(\vec{x}), \vec{\mathbf{x}})$ is inductively defined as follows:

$$VS(F(\vec{x}), \vec{\mathbf{x}}) := \begin{cases} \text{ReLU}(b + \sum_{i=1}^{h} c_i \mathbf{x_i}) & \text{if } F(\vec{x}) = b + \sum_{i=1}^{h} c_i x_i \leq 0 \\ VS(F_1(\vec{x}), \vec{\mathbf{x}}) + VS(F_2(\vec{x}), \vec{\mathbf{x}}) & \text{if } F(\vec{x}) = F_1(\vec{x}) \wedge F_2(\vec{x}) \\ VS(F_1(\vec{x}), \vec{\mathbf{x}}) \cdot VS(F_2(\vec{x}), \vec{\mathbf{x}}) & \text{if } F(\vec{x}) = F_1(\vec{x}) \vee F_2(\vec{x}) \\ VS(F(\vec{x}), \vec{\mathbf{x}}) & \text{if } F(\vec{x}) = (F(\vec{x})) \end{cases}$$

In other words, the *VS* function provides a measure to tell how unsatisfied a formula $F(\vec{x})$ is with respect to a valuation $\vec{\mathbf{x}}$. The following lemma proves that $VS(F(\vec{x}), \vec{\mathbf{x}})$ is always a non-negative value, and that it is 0 if and only if $\vec{\mathbf{x}}$ satisfies the formula $F(\vec{x})$.

Lemma 7.2. *Let $F(\vec{x})$ be a non-strict monotone formula, and let $\vec{\mathbf{x}}$ be a valuation over $\vec{x}$. Then the following properties are respected:*

(1) $VS(F(\vec{x}), \vec{\mathbf{x}}) \geq 0$, and
(2) $VS(F(\vec{x}), \vec{\mathbf{x}}) = 0$ if and only if $F(\vec{\mathbf{x}})$ is true.

Proof. The proof is by structural induction on $F(\vec{x})$. Let $\vec{\mathbf{x}}$ be a valuation over $\vec{x}$.

- If $F(\vec{x})$ is a linear constraint $b + \sum_{i=1}^{n} c_i x_i \leq 0$, the violation score is strictly greater than 0 if the linear constraint is violated by $\vec{\mathbf{x}}$, and 0 if it is respected. Hence, both properties are respected.

- If $F(\vec{x}) = F_1(\vec{x}) \wedge F_2(\vec{x})$, then, by inductive hypothesis we have that $VS(F_1(\vec{x}), \vec{\mathbf{x}}) \geq 0$ and $VS(F_2(\vec{x}), \vec{\mathbf{x}}) \geq 0$. Since $VS(F(\vec{x}), \vec{\mathbf{x}}) = VS(F_1(\vec{x}), \vec{\mathbf{x}}) + VS(F_2(\vec{x}), \vec{\mathbf{x}})$, we have that $VS(F(\vec{x}), \vec{\mathbf{x}}) \geq 0$ as required by property 1. Moreover, we have that $VS(F(\vec{x}), \vec{\mathbf{x}}) = 0$ if and only if both $VS(F_1(\vec{x}), \vec{\mathbf{x}}) = 0$ and $VS(F_2(\vec{x}), \vec{\mathbf{x}}) = 0$. By inductive hypothesis this implies that $\vec{\mathbf{x}}$ satisfies both $F_1(\vec{x})$ and $F_2(\vec{x})$, and thus that is satisfies $F(\vec{x})$ as well.
- If $F(\vec{x}) = F_1(\vec{x}) \vee F_2(\vec{x})$, then, by inductive hypothesis we have that $VS(F_1(\vec{x}), \vec{\mathbf{x}}) \geq 0$ and $VS(F_2(\vec{x}), \vec{\mathbf{x}}) \geq 0$. Since $VS(F(\vec{x}), \vec{\mathbf{x}}) = VS(F_1(\vec{x}), \vec{\mathbf{x}}) \cdot VS(F_2(\vec{x}), \vec{\mathbf{x}})$, we have that $VS(F(\vec{x}), \vec{\mathbf{x}}) \geq 0$ as required by property 1. Moreover, we have that $VS(F(\vec{x}), \vec{\mathbf{x}}) = 0$ if and only if $VS(F_1(\vec{x}), \vec{\mathbf{x}}) = 0$ or $VS(F_2(\vec{x}), \vec{\mathbf{x}}) = 0$. By inductive hypothesis this implies that $\vec{\mathbf{x}}$ satisfies $F_1(\vec{x})$ or $F_2(\vec{x})$, and thus that is satisfies $F(\vec{x})$ as well.
- If $F(\vec{x}) = (F(\vec{x}))$, then the claim trivially follows from $VS(F(\vec{x}), \vec{\mathbf{x}}) = VS(F(\vec{x}), \vec{\mathbf{x}})$. □

Training and Loss Function

We first describe the approach for properties on single input-output pairs and then we show how to extend the approach to sets of properties defined on multiple input-output pairs.

Let N be a ReLU neural network and let $\mathcal{P} = \{P_1(\vec{x}, \vec{y}), \ldots, P_h(\vec{x}, \vec{y})\}$ be a set of properties defined on single pairs. Given an input-output pair $(\vec{\mathbf{x}}, \vec{\mathbf{y}})$, where $\vec{\mathbf{y}} = N(\vec{\mathbf{x}})$ is the value computed by the current network N under input $\vec{\mathbf{x}}$, let

$$\mathcal{F}_{post}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) := \{F_{post}(\vec{x}, \vec{y}) \mid F_{pre}(\vec{\mathbf{x}}) \Rightarrow F_{post}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \mathcal{P} \text{ is false}\}$$

be the set of all postconditions of the properties in $\mathcal{P}$ that are false under the valuation $(\vec{\mathbf{x}}, \vec{\mathbf{y}})$. Let $\mathcal{L}$ be the loss function from the problem statement. This loss function takes a batch B of input output pairs (where the outputs are computed from the inputs by using the current network) and returns a real value. To apply soft constraint acceleration we replace $\mathcal{L}$ with a new loss function defined as follows:

$$\mathcal{L}(B) := \mathcal{L}(B) + w \cdot \mathcal{L}_{\mathcal{S}}(\hat{B})$$

where $w > 0$ is a weight factor, and

$$\mathcal{L}_{\mathcal{S}}(\hat{B}) := \frac{\sum_{(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \in \hat{B}} \sum_{F_{post}(\vec{x}, \vec{y}) \in \mathcal{F}_{post}(\vec{\mathbf{x}}, \vec{\mathbf{y}})} VS(F_{post}(\vec{x}, \vec{y}), (\vec{\mathbf{x}}, \vec{\mathbf{y}}))}{|\hat{B}|}$$

for some randomly generated batch $\hat{B}$ of input-output pairs belonging to the same data distribution of that of the dataset. This new loss function is used in the training phase for both the training and the validation set. The term $w \cdot \mathcal{L}_{\mathcal{S}}(\hat{B})$ averages the violation score of the input-output pairs in the randomly generated batch $\hat{B}$. By Lemma 7.2, we have that the term is 0 when all pairs in the batch satisfy the property, and positive otherwise. Moreover, the value of the term increases as the number of pairs that violate the properties increases, or as the violation score of those pairs increases (i.e., when they are far from satisfying the property). Hence, minimizing the new term of the loss function corresponds to minimize the number of pairs that violate the property, and keeps the pairs violating the property as close as possible to satisfaction.

To consider sets of properties on multiple input-output pairs, we extend the approach as follows. Let $\mathcal{P} = \{P_1(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_{k_1}, \vec{y}_{k_1}), \ldots, P_h(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_{k_h}, \vec{y}_{k_h})\}$ be a set of properties, where each P_i is defined on an arbitrary number of pairs k_i. For every $k \geq 1$, given k input-output pairs $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1), \ldots, (\vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k)$, let

$$\mathcal{F}_{post}(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k) := \{F_{post}(\vec{x}_1, \vec{y}_1, \ldots, \vec{x}_k, \vec{y}_k) \mid F_{pre}(\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_k) \Rightarrow F_{post}(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k) \in \mathcal{P} \text{ is false}\}$$

be the set of all postconditions of the properties in $\mathcal{P}$ defined over k input-output pairs that are false under the valuation $(\vec{\mathbf{x}}_1, \vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{x}}_k, \vec{\mathbf{y}}_k)$. The loss extends as follows:

$$\mathcal{L}(B) := \mathcal{L}(B) + \sum_{i=1}^{k} w_i \cdot \mathcal{L}_{\mathcal{S}}(\hat{B}_{k_i})$$

Table 2. Input and outputs of the MANNERS-DB neural network.

Input	Description	Type	Range	Input	Description	Type	Range
x_1	operating mode	INT	$[0, 1]$	x_{15}	direction from closest human to robot	FLOAT	$[0, 360]$
x_2	number of people	INT	$[0, 9]$	x_{16}	robot facing closest human	INT	$[0, 1]$
x_3	number of people in group	INT	$[0, 5]$	x_{17}	robot facing 2nd closest human	INT	$[0, 1]$
x_4	group radius	FLOAT	$[0.5, 1]$	x_{18}	robot facing 3rd closest human	INT	$[0, 1]$
x_5	distance to group	FLOAT	$[0, 6]$	x_{19}	closest human facing robot	INT	$[0, 1]$
x_6	robot within group	INT	$[0, 1]$	x_{20}	2nd closest human facing robot	INT	$[0, 1]$
x_7	robot facing group	INT	$[0, 1]$	x_{21}	3d closest human facing robot	INT	$[0, 1]$
x_8	robot work radius	FLOAT	$[0, 3]$	x_{22}	number of children	INT	$[0, 2]$
x_9	distance to closest human	FLOAT	$[0.3, 6]$	x_{23}	distance to closest child	FLOAT	$[0.4, 6]$
x_{10}	distance to 2nd closest human	FLOAT	$[0.3, 6]$	x_{24}	number of animals	INT	$[0, 1]$
x_{11}	distance to 3rd closest human	FLOAT	$[0.3, 6]$	x_{25}	distance to closest animal	FLOAT	$[0.4, 6]$
x_{12}	direction to closest human	FLOAT	$[0, 360]$	x_{26}	number of people on sofa	INT	$[0, 2]$
x_{13}	direction to 2nd closest human	FLOAT	$[0, 360]$	x_{27}	music playing	INT	$[0, 1]$
x_{14}	direction to 3rd closest human	FLOAT	$[0, 360]$	x_{28}	number of agents in scene	INT	$[1, 11]$
Output	**Description**	**Type**	**Range**	**Output**	**Description**	**Type**	**Range**
y_1	vacuum cleaning	INT	$[1, 5]$	y_5	carry drinks	INT	$[1, 5]$
y_2	mopping the floor	INT	$[1, 5]$	y_6	carry small objects	INT	$[1, 5]$
y_3	carry warm food	INT	$[1, 5]$	y_7	carry big objects	INT	$[1, 5]$
y_4	carry cold food	INT	$[1, 5]$	y_8	cleaning or starting conversation	INT	$[1, 5]$

where each $w_i > 0$ is a weight factor, and

$$\mathcal{S}_{\mathcal{L}}(\hat{B}_{k_i}) := \frac{\sum_{(\vec{\mathbf{x}}_1,\vec{\mathbf{y}}_1,\ldots,\vec{\mathbf{x}}_{k_i},\vec{\mathbf{y}}_{k_i})\in\hat{B}_{k_i}} \sum_{F_{post}(\vec{x}_1,\vec{y}_1,\ldots,\vec{x}_{k_i},\vec{y}_{k_i})\in\mathcal{F}_{post}(\vec{\mathbf{x}}_1,\vec{\mathbf{y}}_1,\ldots,\vec{\mathbf{x}}_{k_i},\vec{\mathbf{y}}_{k_i})} VS(F_{post}(\vec{x}_1,\vec{y}_1,\ldots,\vec{x}_{k_i},\vec{y}_{k_i}),(\vec{\mathbf{x}}_1,\vec{\mathbf{y}}_1,\ldots,\vec{\mathbf{x}}_{k_i},\vec{\mathbf{y}}_{k_i}))}{|\hat{B}_{k_i}|}$$

for some randomly generated batch $\hat{B}_{k_i}$ of valuations $(\vec{\mathbf{x}}_1,\vec{\mathbf{y}}_1,\ldots,\vec{\mathbf{x}}_{k_i},\vec{\mathbf{y}}_{k_i})$, where each pair $(\vec{\mathbf{x}},\vec{\mathbf{y}})$ in the valuation belongs to the same data distribution of that of the dataset.

8 Mind Your Manners, Robot!

The MANNERS-DB was introduced in (Tjomsland et al. 2022). The case study involves a robot moving in a room where humans and animals are present. The controller is implemented through a neural network. The description of the inputs and outputs are given in Table 2. In particular, the outputs of the neural network provide numbers (scores) representing the adequacy of the possible actions executable by the robot. Each adequacy is a score from 1 to 5, where the bigger the number, the more adequate the action. The adequacy scores have been computed as the average across human judgments obtained via crowd sourcing; see (Tjomsland et al. 2022). The original dataset comprises 11050 input-output pairs.

We considered 2 properties on single pairs in isolation, 2 properties on multiple pairs in isolation, and a set of properties containing those on single pairs. Notice that the properties P_3, P_4 as well as the set of properties $\mathcal{P} = \{P_1, P_4\}$ were not considered in (Zavatteri et al. 2024). All properties are examples of behavioral constraints on the manners of the robot.

In what follows, for each input feature x_i we write $LB(x_i)$ and $UB(x_i)$ for its lower and upper bound given in Table 2, respectively. Moreover, we write

$$F_{Bounds}(\vec{x}) := \bigwedge_{1\le i\le 28} LB(x_i) \le x_i \le UB(x_i)$$

to restrict the application of the precondition within the allowed input bounds.

Property P_1: If the distance to the closest child is within 0.6 meters, then mopping the floor must have minimal score. The encoding of P_1 is:

$$P_1(\vec{x},\vec{y}) := \overbrace{\Big(F_{Bounds}(\vec{x}) \wedge x_{23} \leq 0.6\Big)}^{F_{pre}(\vec{x})} \Rightarrow \overbrace{\Big(\bigwedge_{\substack{1\leq i\leq 8\\ i\neq 2}} y_2 \leq y_i\Big)}^{F_{post}(\vec{x},\vec{y})}$$

The ε-relaxation of P_1 is:

$$P_1^{\varepsilon}(\vec{x},\vec{y}) := \overbrace{\Big(F_{Bounds}(\vec{x}) \wedge x_{23} \leq 0.6\Big)}^{F^{\varepsilon}_{pre}(\vec{x})} \Rightarrow \overbrace{\Big(\bigwedge_{\substack{1\leq i\leq 8\\ i\neq 2}} y_2 < y_i + \varepsilon\Big)}^{F^{\varepsilon}_{post}(\vec{x},\vec{y})}$$

The counterexamples to P_1 are all valuations $(\vec{\mathbf{x}},\vec{\mathbf{y}})$ satisfying the formula:

$$F_N(\vec{x},\vec{y}) \wedge \overbrace{F_{Bounds}(\vec{x}) \wedge x_{23} \leq 0.6}^{F^{\varepsilon}_{pre}(\vec{x})} \wedge \overbrace{\Big(\bigvee_{\substack{1\leq i\leq 8\\ i\neq 2}} y_2 \geq y_i + \varepsilon\Big)}^{\neg F^{\varepsilon}_{post}(\vec{x},\vec{y})}$$

The repair phase creates a vector of 8 fresh variables $\vec{s} = (s_1,\ldots,s_8)$ and solves the following MIQP problem:

$$\begin{aligned} &\min \sum_{i=1}^{8} s_i^2 \\ &\text{subject to} \\ &\bigwedge_{i=1}^{8} \mathbf{y}_2 + s_2 \leq \mathbf{y}_i + s_i \end{aligned}$$

to obtain a new data point $(\vec{\mathbf{x}},\vec{\mathbf{y}} \oplus \vec{\mathbf{s}})$ to add to the dataset.

Property P_2: If the distance to the closest human is within 0.6 meters, then the adequateness of mopping the floor must be less than or equal to the adequateness of mopping the floor in all possible scenarios in which the distance to the closest animal is within 0.6 meters.

To express this kind of property we need to consider the output of the network on two different valuations of the input vector: $\vec{\mathbf{x}}_h$ that represents a scenario where the closest human is within 0.6 meters and $\vec{\mathbf{x}}_a$ that represents a scenario where the closest animal is within 0.6 meters. This means that F_{pre} is defined on $\vec{x}_h$ and $\vec{x}_a$ and F_{post} on $\vec{x}_h,\vec{y}_h,\vec{x}_a,\vec{y}_a$. We used the subscript h,a to distinguish the inputs that satisfies the precondition for human and animal, respectively. The encoding of P_2 is:

$$P_2(\vec{x}_h,\vec{y}_h,\vec{x}_a,\vec{y}_a) := \overbrace{\Big(F_{Bounds}(\vec{x}_h) \wedge F_{Bounds}(\vec{x}_a) \wedge x_{h,9} \leq 0.6 \wedge x_{a,25} \leq 0.6\Big)}^{F_{pre}(\vec{x}_h,\vec{x}_a)} \Rightarrow \overbrace{y_{h,2} \leq y_{a,2}}^{F_{post}(\vec{x}_h,\vec{y}_h,\vec{x}_a,\vec{y}_a)}$$

The ε-relaxation of P_2 is:

$$P_2^{\varepsilon}(\vec{x}_h, \vec{y}_h, \vec{x}_a, \vec{y}_a) := \overbrace{\left(F_{Bounds}(\vec{x}_h) \wedge F_{Bounds}(\vec{x}_a) \wedge x_{h,9} \leq 0.6 \wedge x_{a,25} \leq 0.6\right)}^{F^{\varepsilon}_{pre}(\vec{x}_h,\vec{x}_a)} \Rightarrow \overbrace{y_{h,2} < y_{a,2} + \varepsilon}^{F^{\varepsilon}_{post}(\vec{x}_h,\vec{y}_h,\vec{x}_a,\vec{y}_a)}$$

The counterexamples to P_2 are all pairs of valuations $(\mathbf{x_h}, \mathbf{y_h})$, $(\mathbf{x_a}, \mathbf{y_a})$ satisfying:

$$F_N(\vec{x}_h, \vec{y}_h) \wedge F_N(\vec{x}_a, \vec{y}_a) \wedge \overbrace{F_{Bounds}(\vec{x}_h) \wedge F_{Bounds}(\vec{x}_a) \wedge x_{h,9} \leq 0.6 \wedge x_{a,25} \leq 0.6}^{F^{\varepsilon}_{pre}(\vec{x}_h,\vec{x}_a)} \wedge \overbrace{y_{h,2} \geq y_{a,2} + \varepsilon}^{\neg F^{\varepsilon}_{post}(\vec{x}_h,\vec{y}_h,\vec{x}_a,\vec{y}_a)}$$

where $(\vec{x}_h, \vec{y}_h)$ is the input-output pair of the first copy of the network (for the human) and $(\vec{x}_a, \vec{y}_a)$ is the input-output pair of the second copy of the network (for the animal).

The repair phase creates 2 vectors of 8 fresh variables $\vec{s}_h = (s_{h,1}, \ldots, s_{h,8})$ and $\vec{s}_a = (s_{a,1}, \ldots, s_{a,8})$ and solves the following MIQP problem:

$$\begin{aligned} &\min \left(\sum_{i=1}^{8} s_{h,i}^2 + \sum_{i=1}^{8} s_{a,i}^2 \right) \\ &\text{subject to} \\ &\mathbf{y_{h,2}} + s_{h,2} \leq \mathbf{y_{a,2}} + s_{a,2} \end{aligned}$$

to obtain two new data points $(\vec{\mathbf{x}}, \vec{\mathbf{y}} \oplus \vec{\mathbf{s}}_\mathbf{h})$ and $(\vec{\mathbf{x}}, \vec{\mathbf{y}} \oplus \vec{\mathbf{s}}_\mathbf{a})$ to add to the dataset.

Property P_3: If the distance to the closest animal is within 0.6 meters, then the adequateness of mopping the floor must be less than or equal to the adequateness of mopping the floor in all possible scenarios in which the distance to the closest human is within 0.6 meters.

Notice that P_3 is the same as P_2 with human and animal swapped. The encoding of P_3 is:

$$P_3(\vec{x}_a, \vec{y}_a, \vec{x}_h, \vec{y}_h) := \overbrace{\left(F_{Bounds}(\vec{x}_a) \wedge F_{Bounds}(\vec{x}_h) \wedge x_{a,25} \leq 0.6 \wedge x_{h,9} \leq 0.6\right)}^{F_{pre}(\vec{x}_a,\vec{x}_h)} \Rightarrow \overbrace{y_{a,2} \leq y_{h,2}}^{F_{post}(\vec{x}_a,\vec{y}_a,\vec{x}_h,\vec{y}_h)}$$

The ε-relaxation of P_3 is:

$$P_3^{\varepsilon}(\vec{x}_a, \vec{y}_a, \vec{x}_h, \vec{y}_h) := \overbrace{\left(F_{Bounds}(\vec{x}_a) \wedge F_{Bounds}(\vec{x}_h) \wedge x_{a,9} \leq 0.6 \wedge x_{h,25} \leq 0.6\right)}^{F^{\varepsilon}_{pre}(\vec{x}_a,\vec{x}_h)} \Rightarrow \overbrace{y_{a,2} < y_{h,2} + \varepsilon}^{F^{\varepsilon}_{post}(\vec{x}_a,\vec{y}_a,\vec{x}_h,\vec{y}_h)}$$

The counterexamples to P_3 are all pairs of valuations $(\mathbf{x_a}, \mathbf{y_a})$, $(\mathbf{x_h}, \mathbf{y_h})$ satisfying:

$$F_N(\vec{x}_a, \vec{y}_a) \wedge F_N(\vec{x}_h, \vec{y}_h) \wedge \overbrace{F_{Bounds}(\vec{x}_a) \wedge F_{Bounds}(\vec{x}_h) \wedge x_{a,9} \leq 0.6 \wedge x_{h,25} \leq 0.6}^{F^{\varepsilon}_{pre}(\vec{x}_a,\vec{x}_h)} \wedge \overbrace{y_{a,2} \geq y_{h,2} + \varepsilon}^{\neg F^{\varepsilon}_{post}(\vec{x}_a,\vec{y}_a,\vec{x}_h,\vec{y}_h)}$$

where $(\vec{x}_a, \vec{y}_a)$ is the input-output pair of the first copy of the network (for the animal) and $(\vec{x}_h, \vec{y}_h)$ is the input-output pair of the second copy of the network (for the human).

The repair phase creates 2 vectors of 8 fresh variables $\vec{s}_a = (s_{a,1}, \ldots, s_{a,8})$ and $\vec{s}_h = (s_{h,1}, \ldots, s_{h,8})$ and solves the following MIQP problem:

$$\min \left(\sum_{i=1}^{8} s_{a,i}^2 + \sum_{i=1}^{8} s_{h,i}^2 \right)$$
$$\text{subject to}$$
$$\mathbf{y_{a,2}} + s_{a,2} \leq \mathbf{y_{h,2}} + s_{h,2}$$

to obtain two new data points $(\vec{\mathbf{x}}, \vec{\mathbf{y}} \oplus \vec{\mathbf{s}}_\mathbf{a})$ and $(\vec{\mathbf{x}}, \vec{\mathbf{y}} \oplus \vec{\mathbf{s}}_\mathbf{h})$ to add to the dataset.

Property P_4: If the distance to the closest child is at least 4.5 meters then carry warm food or carry big objects has maximum score.
The encoding of P_4 is:

$$P_4(\vec{x}, \vec{y}) := \left(\overbrace{F_{Bounds}(\vec{x}) \wedge x_{23} \geq 4.5}^{F_{pre}(\vec{x})} \right) \Rightarrow \overbrace{\left(\left(\bigwedge_{\substack{1 \leq i \leq 8 \\ i \neq 3}} y_3 \geq y_i \right) \vee \left(\bigwedge_{\substack{1 \leq i \leq 8 \\ i \neq 7}} y_7 \geq y_i \right) \right)}^{F_{post}(\vec{x}, \vec{y})}$$

The ε-relaxation of P_4 is:

$$P_4^\varepsilon(\vec{x}, \vec{y}) := \left(\overbrace{F_{Bounds}(\vec{x}) \wedge x_{23} \geq 4.5}^{F_{pre}^\varepsilon(\vec{x})} \right) \Rightarrow \overbrace{\left(\left(\bigwedge_{\substack{1 \leq i \leq 8 \\ i \neq 3}} y_3 + \varepsilon > y_i \right) \vee \left(\bigwedge_{\substack{1 \leq i \leq 8 \\ i \neq 7}} y_7 + \varepsilon > y_i \right) \right)}^{F_{post}^\varepsilon(\vec{x}, \vec{y})}$$

The counterexamples are all valuations $(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ satisfying:

$$F_N(\vec{x}, \vec{y}) \wedge \overbrace{F_{Bounds}(\vec{x}) \wedge x_{23} \geq 4.5}^{F_{pre}^\varepsilon(\vec{x})} \wedge \overbrace{\left(\bigvee_{\substack{1 \leq i \leq 8 \\ i \neq 3}} y_3 + \varepsilon \leq y_i \right) \wedge \left(\bigvee_{\substack{1 \leq i \leq 8 \\ i \neq 7}} y_7 + \varepsilon \leq y_i \right)}^{\neg F_{post}^\varepsilon(\vec{x}, \vec{y})}$$

The repair phase creates a vector of 8 fresh variables $\vec{s} = (s_1, \ldots, s_8)$ and solves the following MIQP problem:

$$\min \sum_{i=1}^{8} s_i^2$$
$$\text{subject to}$$
$$\left(\bigwedge_{i=1}^{8} \mathbf{y}_3 + s_3 \geq \mathbf{y}_i + s_i \right) \vee \left(\bigwedge_{i=1}^{8} \mathbf{y}_7 + s_7 \geq \mathbf{y}_i + s_i \right)$$

to obtain a new data point $(\vec{\mathbf{x}}, \vec{\mathbf{y}} \oplus \vec{\mathbf{s}})$ to add to the dataset.

Implementation

We implemented our approach in Python 3, using PyTorch 2.1.1 as the deep learning framework, Marabou 2.0.0 (Katz, Huang, et al. 2019) as the neural network verifier, and Gurobi 13.0.1 (Gurobi Optimization, LLC 2024) for repair (both with default parameters). We ran our software on a High Performance Computing (HPC) center using 3 compute nodes, each one equipped with 2 Intel(R) Xeon(R) CPU E5-2650 v3 2.30GHz, 1 GPU Nvidia T4, 160GB of RAM, and running Ubuntu 20.04.3 LTS. The HPC is handled by the SLURM workload manager (Jette

and Wickberg 2023). Each job was scheduled on one of these 3 nodes and assigned 20 CPU cores, 150GB of RAM, and the whole GPU of the node.

We tested our workflow on a ReLU network with 28 inputs, 8 outputs, 10 hidden layers and 6 neurons per hidden layer. We normalized each input and output to make it fall in $[0, 1]$, where given any pair $(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ in the dataset, $(\vec{\mathbf{x}}_{\mathbf{normalized}}, \vec{\mathbf{y}}_{\mathbf{normalized}})$ is built as follows:

$$\vec{\mathbf{x}}_{\mathbf{normalized}} := \left(\frac{\mathbf{x_1}}{\mathbf{x}_{\mathtt{max}}}, \ldots, \frac{\mathbf{x_n}}{\mathbf{x}_{\mathtt{max}}}\right) \qquad \vec{\mathbf{y}}_{\mathbf{normalized}} := \left(\frac{\mathbf{y}_1}{\mathbf{y}_{\mathtt{max}}}, \ldots, \frac{\mathbf{y}_m}{\mathbf{y}_{\mathtt{max}}}\right)$$

When some inputs are not applicable, such as distance to the closest animal when no animal is present or group distance when no group exists, we normalized their value to 1. Moreover, the software rewrites the properties we gave above according to this normalization too.

For each property, we randomly partitioned[1] the dataset in train, validation, and test sets so that they contain 8840 (i.e., 80%), 1105 (i.e., 10%), and 1105 (i.e., 10%) pairs, respectively. We used the Adam optimizer (Kingma and Ba 2015) with learning rates of 10^{-2}, 10^{-3}, and 10^{-4} and mean square error (MSE) loss function. We trained the networks on the normalized data for at most 100 epochs by dividing the data in batches of size 100 each. We kept the network with minimal loss on the validation set among the epochs, by applying an early stopping criterion of 20 epochs and we evaluate its accuracy (at the end) on the test set. Every CEGIS iteration starts over with a new network whose weights are randomly initialized. We used $\varepsilon = 10^{-4}$. For soft acceleration we always considered a batch $\hat{B}$ as big as the batch of training data coming from the dataset and set the weights of the soft loss part to $w = 1$.

For each property P detailed above, we set up the verification phase as follows. The opportunistic phase samples 1000 input points inside the region of the input space satisfying the precondition, computes the outputs of the network under analysis, and checks for the existence of counterexamples. If any are found, it returns the corresponding set of counterexamples. If the opportunistic phase fails, the formal verification phase searches for a single counterexample. After repairing a counterexample, a small number of additional input-output pairs are generated by perturbing both the continuous inputs and outputs with a noise vector. Specifically, we sample a tensor from a normal distribution with mean 0 and variance 1, scale it by a factor of 0.1, and add it to both the input and output components of the counterexample. Only the pairs satisfying the postcondition of the property under analysis are kept. We used Algorithm 1 to repair counterexamples to properties $P_1, P_4, \mathcal{P}$ and Algorithm 2 for P_2 and P_3. Each experiment is defined by a property, a learning rate, a maximum number of counterexamples, a maximum number of sampled pairs, and whether soft-acceleration is used or not. We run each configuration 3 times with a timeout of 5 hours per single experiment. Table 3 reports the results aggregated according to the following two metrics.

(1) **Varying the number of counterexamples:** the maximum number of counterexamples is varied across $\{25, 50, 75\}$, while the maximum number of sampled points per counterexample is fixed at 20.
(2) **Varying the number of sampled points:** the maximum number of counterexamples is fixed at 50, while the maximum number of sampled points per counterexample is varied across $\{10, 20, 30\}$.

The data shows that the only experiments that hit the timeout are those involving conjunction of properties on single pairs when using soft-acceleration constraints. Notice that the use of soft-acceleration must be seen as a tradeoff: it helps reduce the CEGIS iterations by paying the price of making training hevier as more computation is needed when computing the loss function. We plot the average formal verification time vs average repair time in Figure 7. The data clearly shows that in general, repair is harder when applied to multipair properties or to disjunctive properties or to sets of properties. Finally, we also show the convergence curves for these two studies in Table 4 and Table 5. Regardless of the study, we adopt the following convention: solid lines report on

[1]Each partition corresponds to a specific seed in the experimental evaluation.

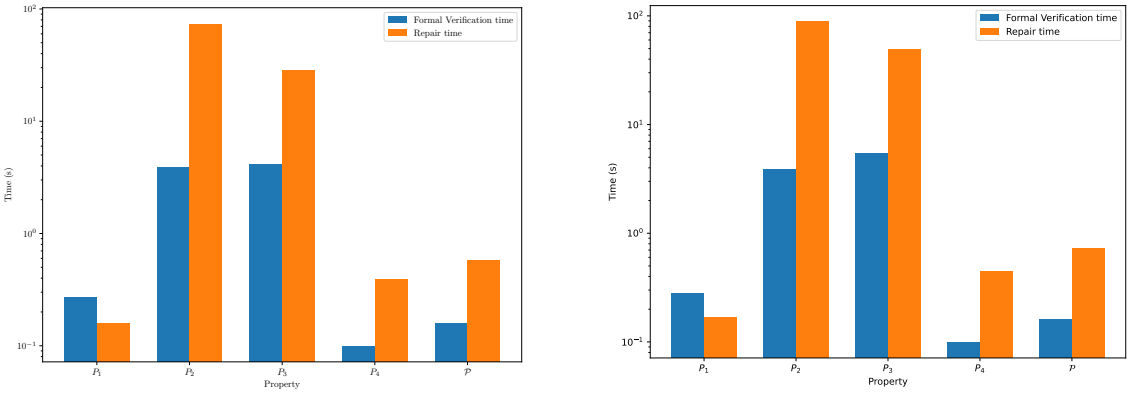
Table 3. MANNERS-DB: Each line shows average value and standard deviation of at most 9 experiments (3 experiments for each learning rate) by excluding those experiments that hit the timeout. Unshaded rows reports on data without soft-acceleration, whereas shaded rows reports on data with soft-acceleration. **P**, **MaxCE**, **MaxS**, **TO**, **FV** are shorts for property, max number of counterexamples, max number of sampled points, timeout, and formal verification, respectively.

(a) Results of the experiments by varying the maximum number of counterexamples with respect to maximum of 20 sampled pairs.

			Average value ± standard deviation												
			Generic synthesis process		Generic CEGIS iteration										
	MaxCE	TO	CEGIS iter.	Total time (s)	Training time (s)	Opportun. search (s)	FV time (s)	Repair time (s)	Initial loss on valid	Final loss on valid	Initial loss on test	Final loss on test	CounterEX (opportun.)	FV calls	New pairs
P_1	25	0	33 ± 5	2632.72 ± 790.29	78.64 ± 34.87	0.0 ± 0.01	0.08 ± 0.47	0.13 ± 0.1	0.065 ± 0.003	0.068 ± 0.002	0.063 ± 0.003	0.067 ± 0.0	24 ± 5	0 ± 0	277 ± 54
		0	3 ± 3	244.36 ± 225.84	74.1 ± 22.85	0.0 ± 0.0	1.44 ± 1.04	0.06 ± 0.09	0.07 ± 0.017	0.074 ± 0.015	0.069 ± 0.017	0.072 ± 0.016	0 ± 1	2 ± 2	9 ± 8
	50	0	25 ± 5	2119.71 ± 451.36	82.89 ± 40.03	0.0 ± 0.01	0.11 ± 0.54	0.18 ± 0.1	0.065 ± 0.004	0.068 ± 0.002	0.064 ± 0.003	0.067 ± 0.0	45 ± 14	0 ± 0	516 ± 163
		0	4 ± 7	407.8 ± 783.54	95.0 ± 21.82	0.0 ± 0.0	1.31 ± 0.87	0.08 ± 0.1	0.065 ± 0.004	0.068 ± 0.003	0.064 ± 0.003	0.067 ± 0.002	0 ± 1	2 ± 5	10 ± 6
	75	0	15 ± 4	1335.4 ± 659.53	87.17 ± 46.38	0.01 ± 0.02	0.18 ± 0.69	0.24 ± 0.12	0.061 ± 0.003	0.068 ± 0.002	0.061 ± 0.002	0.067 ± 0.0	66 ± 23	0 ± 0	760 ± 265
		0	2 ± 1	155.49 ± 134.58	75.42 ± 25.59	0.01 ± 0.01	1.99 ± 1.37	0.05 ± 0.09	0.065 ± 0.003	0.073 ± 0.015	0.064 ± 0.003	0.072 ± 0.015	0 ± 0	1 ± 1	7 ± 7
P_2	25	0	3 ± 1	291.09 ± 229.15	58.59 ± 22.18	0.02 ± 0.03	2.99 ± 8.58	34.32 ± 33.4	0.063 ± 0.004	0.068 ± 0.002	0.062 ± 0.003	0.067 ± 0.0	13 ± 13	0 ± 0	174 ± 168
		0	1 ± 1	121.02 ± 124.2	75.85 ± 40.97	0.0 ± 0.0	3.07 ± 2.93	3.86 ± 6.08	0.067 ± 0.002	0.068 ± 0.002	0.066 ± 0.002	0.067 ± 0.0	0 ± 1	0 ± 1	5 ± 11
	50	0	4 ± 2	1155.77 ± 973.77	64.17 ± 22.04	0.02 ± 0.02	1.01 ± 1.82	142.18 ± 178.75	0.062 ± 0.003	0.068 ± 0.002	0.061 ± 0.002	0.067 ± 0.0	33 ± 24	0 ± 0	419 ± 307
		0	3 ± 3	381.14 ± 456.92	105.29 ± 46.37	0.0 ± 0.0	9.59 ± 29.88	15.39 ± 17.18	0.067 ± 0.003	0.068 ± 0.002	0.065 ± 0.002	0.067 ± 0.0	1 ± 2	1 ± 2	12 ± 19
	75	0	2 ± 2	803.57 ± 1339.49	62.9 ± 28.01	0.04 ± 0.05	3.95 ± 11.21	188.03 ± 345.31	0.065 ± 0.004	0.068 ± 0.002	0.064 ± 0.003	0.067 ± 0.0	38 ± 39	0 ± 0	498 ± 513
		0	2 ± 3	257.26 ± 419.16	92.98 ± 44.01	0.0 ± 0.0	3.21 ± 6.41	17.92 ± 20.3	0.067 ± 0.003	0.068 ± 0.002	0.065 ± 0.003	0.067 ± 0.0	1 ± 3	1 ± 2	11 ± 25
P_3	25	0	3 ± 2	358.06 ± 367.19	55.74 ± 18.77	0.02 ± 0.03	1.32 ± 2.16	50.77 ± 59.4	0.063 ± 0.004	0.068 ± 0.002	0.062 ± 0.003	0.067 ± 0.0	15 ± 12	0 ± 0	204 ± 173
		0	2 ± 2	269.12 ± 320.46	104.86 ± 61.19	0.0 ± 0.0	2.04 ± 1.8	6.49 ± 5.8	0.067 ± 0.003	0.068 ± 0.002	0.065 ± 0.002	0.067 ± 0.0	0 ± 1	1 ± 1	10 ± 13
	50	0	2 ± 1	264.12 ± 353.53	56.5 ± 19.04	0.03 ± 0.03	1.75 ± 1.5	44.37 ± 71.92	0.068 ± 0.008	0.07 ± 0.006	0.067 ± 0.007	0.068 ± 0.005	19 ± 25	0 ± 0	258 ± 344
		0	1 ± 0	140.38 ± 156.13	80.23 ± 47.91	0.0 ± 0.0	21.01 ± 62.54	2.7 ± 4.89	0.067 ± 0.003	0.068 ± 0.002	0.065 ± 0.002	0.067 ± 0.0	0 ± 0	0 ± 0	3 ± 5
	75	0	2 ± 1	300.58 ± 404.68	60.47 ± 17.78	0.04 ± 0.03	2.08 ± 1.51	50.93 ± 81.19	0.066 ± 0.004	0.068 ± 0.002	0.064 ± 0.003	0.067 ± 0.0	21 ± 35	0 ± 0	296 ± 484
		0	2 ± 2	242.67 ± 355.63	108.48 ± 48.65	0.0 ± 0.0	2.94 ± 5.63	7.91 ± 11.12	0.067 ± 0.004	0.069 ± 0.005	0.065 ± 0.003	0.068 ± 0.004	1 ± 2	0 ± 1	15 ± 33
P_4	25	0	43 ± 12	3295.01 ± 821.29	76.23 ± 34.87	0.01 ± 0.02	0.07 ± 0.43	0.3 ± 0.35	0.065 ± 0.004	0.068 ± 0.002	0.063 ± 0.003	0.067 ± 0.0	24 ± 4	0 ± 0	195 ± 34
		0	46 ± 6	4875.57 ± 1959.36	105.81 ± 47.15	0.0 ± 0.0	0.06 ± 0.39	0.28 ± 0.28	0.066 ± 0.009	0.068 ± 0.002	0.066 ± 0.009	0.067 ± 0.0	24 ± 4	0 ± 0	194 ± 35
	50	0	24 ± 6	2194.53 ± 940.52	90.16 ± 43.65	0.01 ± 0.02	0.12 ± 0.59	0.51 ± 0.64	0.063 ± 0.004	0.068 ± 0.002	0.062 ± 0.003	0.067 ± 0.0	47 ± 10	0 ± 0	379 ± 86
		0	23 ± 3	2489.52 ± 869.21	105.08 ± 48.45	0.0 ± 0.0	0.11 ± 0.53	0.41 ± 0.16	0.065 ± 0.004	0.068 ± 0.002	0.064 ± 0.003	0.067 ± 0.0	47 ± 11	0 ± 0	375 ± 90
	75	0	17 ± 3	1434.87 ± 564.01	80.97 ± 38.93	0.01 ± 0.02	0.16 ± 0.65	0.57 ± 0.25	0.065 ± 0.004	0.068 ± 0.002	0.064 ± 0.003	0.066 ± 0.002	69 ± 20	0 ± 0	553 ± 161
		0	17 ± 2	1754.01 ± 678.69	102.64 ± 48.19	0.01 ± 0.06	0.16 ± 0.63	0.54 ± 0.21	0.066 ± 0.003	0.068 ± 0.002	0.065 ± 0.003	0.067 ± 0.0	69 ± 19	0 ± 0	557 ± 154
$\mathcal{P}$	25	0	73 ± 17	11007.63 ± 3234.61	150.57 ± 84.09	0.01 ± 0.01	0.06 ± 0.49	0.43 ± 0.33	0.064 ± 0.004	0.068 ± 0.002	0.063 ± 0.003	0.067 ± 0.0	45 ± 10	0 ± 0	439 ± 111
		2	60 ± 23	9963.74 ± 5058.3	166.01 ± 69.38	0.0 ± 0.0	0.21 ± 0.73	0.4 ± 0.67	0.067 ± 0.005	0.074 ± 0.016	0.066 ± 0.003	0.073 ± 0.016	23 ± 10	0 ± 0	184 ± 92
	50	0	42 ± 16	6373.66 ± 2269.55	149.29 ± 84.63	0.01 ± 0.01	0.09 ± 0.58	0.65 ± 0.22	0.064 ± 0.005	0.068 ± 0.002	0.064 ± 0.003	0.067 ± 0.0	84 ± 26	0 ± 0	809 ± 275
		0	45 ± 12	9658.41 ± 4329.14	215.17 ± 102.74	0.01 ± 0.0	0.27 ± 0.85	0.62 ± 0.72	0.066 ± 0.003	0.071 ± 0.008	0.065 ± 0.003	0.07 ± 0.008	44 ± 20	1 ± 1	352 ± 169
	75	0	32 ± 10	4875.99 ± 1267.9	149.54 ± 78.45	0.01 ± 0.01	0.13 ± 0.68	0.94 ± 0.82	0.064 ± 0.004	0.069 ± 0.002	0.063 ± 0.003	0.067 ± 0.0	116 ± 45	0 ± 0	1118 ± 466
		1	32 ± 12	7303.62 ± 3493.98	226.19 ± 114.62	0.01 ± 0.01	0.33 ± 0.98	0.67 ± 0.39	0.065 ± 0.004	0.073 ± 0.011	0.064 ± 0.003	0.071 ± 0.012	60 ± 34	1 ± 1	484 ± 293

(b) Results of the experiments by varying the sampling batch with respect to a maximum 50 counterexamples.

			Average value ± standard deviation												
			Generic synthesis process		Generic CEGIS iteration										
	MaxS	TO	CEGIS iter.	Total time (s)	Training time (s)	Opportun. search (s)	FV time (s)	Repair time (s)	Initial loss on valid	Final loss on valid	Initial loss on test	Final loss on test	CounterEX (opportun.)	FV calls	New pairs
P_1	10	0	39 ± 5	3124.21 ± 595.58	79.73 ± 37.38	0.0 ± 0.01	0.07 ± 0.41	0.18 ± 0.11	0.064 ± 0.003	0.068 ± 0.002	0.063 ± 0.003	0.067 ± 0.0	47 ± 11	0 ± 0	294 ± 69
		0	3 ± 2	195.46 ± 158.66	73.85 ± 25.06	0.01 ± 0.01	2.34 ± 2.54	0.05 ± 0.08	0.065 ± 0.003	0.076 ± 0.022	0.063 ± 0.003	0.074 ± 0.022	0 ± 1	1 ± 2	4 ± 4
	20	0	25 ± 5	2119.71 ± 451.36	82.89 ± 40.03	0.0 ± 0.01	0.11 ± 0.54	0.18 ± 0.1	0.065 ± 0.004	0.068 ± 0.002	0.064 ± 0.003	0.067 ± 0.0	45 ± 14	0 ± 0	516 ± 163
		0	4 ± 7	407.8 ± 783.54	95.0 ± 21.82	0.0 ± 0.0	1.31 ± 0.87	0.08 ± 0.1	0.065 ± 0.004	0.068 ± 0.003	0.064 ± 0.003	0.067 ± 0.002	0 ± 1	2 ± 5	10 ± 6
	30	0	15 ± 7	1197.52 ± 572.83	79.35 ± 36.74	0.01 ± 0.01	0.17 ± 0.65	0.17 ± 0.11	0.066 ± 0.008	0.07 ± 0.005	0.065 ± 0.007	0.068 ± 0.005	44 ± 15	0 ± 0	735 ± 248
		0	3 ± 3	283.48 ± 319.8	80.52 ± 23.16	0.0 ± 0.0	1.49 ± 0.67	0.08 ± 0.1	0.066 ± 0.004	0.072 ± 0.014	0.065 ± 0.003	0.07 ± 0.013	0 ± 0	2 ± 3	12 ± 9
P_2	10	0	3 ± 2	779.75 ± 982.47	65.39 ± 21.96	0.02 ± 0.02	2.29 ± 5.82	153.34 ± 193.8	0.064 ± 0.004	0.068 ± 0.002	0.063 ± 0.003	0.067 ± 0.0	29 ± 25	0 ± 1	221 ± 189
		0	2 ± 1	181.4 ± 193.18	89.02 ± 46.84	0.0 ± 0.0	2.13 ± 1.31	9.91 ± 15.16	0.067 ± 0.002	0.068 ± 0.002	0.066 ± 0.002	0.067 ± 0.0	2 ± 4	0 ± 1	10 ± 24
	20	0	4 ± 2	1155.77 ± 973.77	64.17 ± 22.04	0.02 ± 0.02	1.01 ± 1.82	142.18 ± 178.75	0.062 ± 0.003	0.068 ± 0.002	0.061 ± 0.002	0.067 ± 0.0	33 ± 24	0 ± 0	419 ± 307
		0	3 ± 3	381.14 ± 456.92	105.29 ± 46.37	0.0 ± 0.0	9.59 ± 29.88	15.39 ± 17.18	0.067 ± 0.003	0.068 ± 0.002	0.065 ± 0.002	0.067 ± 0.0	1 ± 2	1 ± 2	12 ± 19
	30	0	2 ± 2	914.55 ± 1384.73	67.88 ± 26.71	0.03 ± 0.03	1.32 ± 1.57	127.43 ± 187.48	0.064 ± 0.004	0.068 ± 0.002	0.063 ± 0.003	0.067 ± 0.0	27 ± 25	0 ± 0	501 ± 468
		0	2 ± 2	265.86 ± 454.07	92.52 ± 49.02	0.0 ± 0.0	8.06 ± 25.37	12.03 ± 20.12	0.067 ± 0.003	0.068 ± 0.002	0.066 ± 0.002	0.067 ± 0.0	3 ± 10	1 ± 1	58 ± 191
P_3	10	0	2 ± 1	386.39 ± 596.2	55.4 ± 19.75	0.04 ± 0.04	3.79 ± 10.37	106.52 ± 199.33	0.065 ± 0.004	0.069 ± 0.002	0.065 ± 0.004	0.067 ± 0.002	24 ± 26	0 ± 0	187 ± 201
		0	3 ± 3	342.76 ± 455.55	105.22 ± 61.04	0.0 ± 0.0	2.65 ± 4.74	9.39 ± 7.04	0.067 ± 0.003	0.068 ± 0.002	0.065 ± 0.002	0.067 ± 0.0	1 ± 2	1 ± 2	9 ± 11
	20	0	2 ± 1	264.12 ± 353.53	56.5 ± 19.04	0.03 ± 0.03	1.75 ± 1.5	44.37 ± 71.92	0.068 ± 0.008	0.07 ± 0.006	0.067 ± 0.007	0.068 ± 0.005	19 ± 25	0 ± 0	258 ± 344
		0	1 ± 0	140.38 ± 156.13	80.23 ± 47.91	0.0 ± 0.0	21.01 ± 62.54	2.7 ± 4.89	0.067 ± 0.003	0.068 ± 0.002	0.065 ± 0.002	0.067 ± 0.0	0 ± 0	0 ± 0	3 ± 5
	30	0	2 ± 1	788.48 ± 852.74	60.46 ± 19.69	0.04 ± 0.04	3.06 ± 7.41	110.57 ± 204.7	0.063 ± 0.004	0.068 ± 0.002	0.062 ± 0.003	0.067 ± 0.0	29 ± 25	0 ± 0	544 ± 484
		0	2 ± 1	215.67 ± 207.85	100.77 ± 45.4	0.0 ± 0.0	6.85 ± 19.59	5.21 ± 5.74	0.067 ± 0.003	0.068 ± 0.002	0.065 ± 0.003	0.067 ± 0.0	0 ± 1	1 ± 1	8 ± 19
P_4	10	0	42 ± 5	3619.15 ± 1248.21	84.04 ± 39.94	0.0 ± 0.01	0.07 ± 0.44	0.45 ± 0.17	0.064 ± 0.004	0.068 ± 0.002	0.063 ± 0.003	0.067 ± 0.0	49 ± 8	0 ± 0	219 ± 37
		0	44 ± 7	4612.93 ± 1380.32	103.61 ± 46.51	0.0 ± 0.01	0.06 ± 0.37	0.46 ± 0.33	0.064 ± 0.004	0.068 ± 0.002	0.063 ± 0.003	0.067 ± 0.0	48 ± 9	0 ± 0	215 ± 40
	20	0	24 ± 6	2194.53 ± 940.52	90.16 ± 43.65	0.01 ± 0.02	0.12 ± 0.59	0.51 ± 0.64	0.063 ± 0.004	0.068 ± 0.002	0.062 ± 0.003	0.067 ± 0.0	47 ± 10	0 ± 0	379 ± 86
		0	23 ± 3	2489.52 ± 869.21	105.08 ± 48.45	0.0 ± 0.0	0.11 ± 0.53	0.41 ± 0.16	0.065 ± 0.004	0.068 ± 0.002	0.064 ± 0.003	0.067 ± 0.0	47 ± 11	0 ± 0	375 ± 90
	30	0	17 ± 3	1339.01 ± 407.06	77.18 ± 35.21	0.01 ± 0.02	0.17 ± 0.68	0.44 ± 0.19	0.065 ± 0.005	0.068 ± 0.002	0.064 ± 0.004	0.067 ± 0.0	46 ± 13	0 ± 0	529 ± 146
		0	19 ± 5	2211.64 ± 1413.95	117.38 ± 58.11	0.0 ± 0.0	0.15 ± 0.63	0.42 ± 0.19	0.064 ± 0.003	0.068 ± 0.002	0.063 ± 0.003	0.067 ± 0.0	46 ± 13	0 ± 0	523 ± 148
$\mathcal{P}$	10	0	75 ± 23	10860.82 ± 2976.03	144.53 ± 84.13	0.01 ± 0.01	0.05 ± 0.45	0.68 ± 0.46	0.064 ± 0.006	0.068 ± 0.002	0.064 ± 0.006	0.067 ± 0.0	88 ± 23	0 ± 0	472 ± 133
		3	71 ± 16	11606.27 ± 1545.64	163.28 ± 69.05	0.01 ± 0.0	0.16 ± 0.69	0.56 ± 0.5	0.065 ± 0.004	0.068 ± 0.002	0.064 ± 0.003	0.067 ± 0.0	47 ± 18	0 ± 0	211 ± 94
	20	0	42 ± 16	6373.66 ± 2269.55	149.29 ± 84.63	0.01 ± 0.01	0.09 ± 0.58	0.65 ± 0.22	0.064 ± 0.005	0.068 ± 0.002	0.064 ± 0.003	0.067 ± 0.0	84 ± 26	0 ± 0	809 ± 275
		0	45 ± 12	9658.41 ± 4329.14	215.17 ± 102.74	0.01 ± 0.0	0.27 ± 0.85	0.62 ± 0.72	0.066 ± 0.003	0.071 ± 0.008	0.065 ± 0.003	0.07 ± 0.008	44 ± 20	1 ± 1	352 ± 169
	30	0	30 ± 8	4704.73 ± 1675.89	155.29 ± 90.59	0.01 ± 0.02	0.14 ± 0.76	0.69 ± 0.53	0.064 ± 0.005	0.068 ± 0.002	0.064 ± 0.003	0.067 ± 0.0	82 ± 27	0 ± 0	1135 ± 412
		0	34 ± 18	7941.84 ± 4735.28	230.98 ± 117.34	0.01 ± 0.01	0.36 ± 1.0	1.32 ± 12.28	0.084 ± 0.059	0.071 ± 0.006	0.084 ± 0.06	0.07 ± 0.006	44 ± 22	1 ± 2	517 ± 291



(a) Data of Table 3a aggregated per property.

(b) Data of Table 3b aggregated per property.

Fig. 7. MANNERS-DB: Average formal verification time vs average repair time (log y scale).

data without soft-acceleration, whereas dashed lines report on data with soft-acceleration. Hence, each curve corresponds to a row a table in Table 3a or Table 3b and it is an estimation of the number of violations to the property under analysis. Since the 3 repeating experiments for all learning rates (in each row) may terminate in different numbers of CEGIS iterations, then, for each experiment, we estimate the number of violations at specific iterations that lie at (or are close to) the 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100% of the whole CEGIS synthesis process. This way we collect 11 values for each experiment we consider, regardless of how many CEGIS iterations it took (if an experiment terminates in less than 11 iterations some of these values will be equal). For each experiment, we compute the values of the plot as follows. For each network that lies at (or is close to) the 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100% of the whole CEGIS synthesis process (of that experiment), we generate a random sample of 1000 input points that satisfy to precondition of the property under analysis and we compute their corresponding outputs. After that we count the number of violations. Each curve averages the values of all experiments involving the same network structure, with (solid) or without (dashed) soft-acceleration.

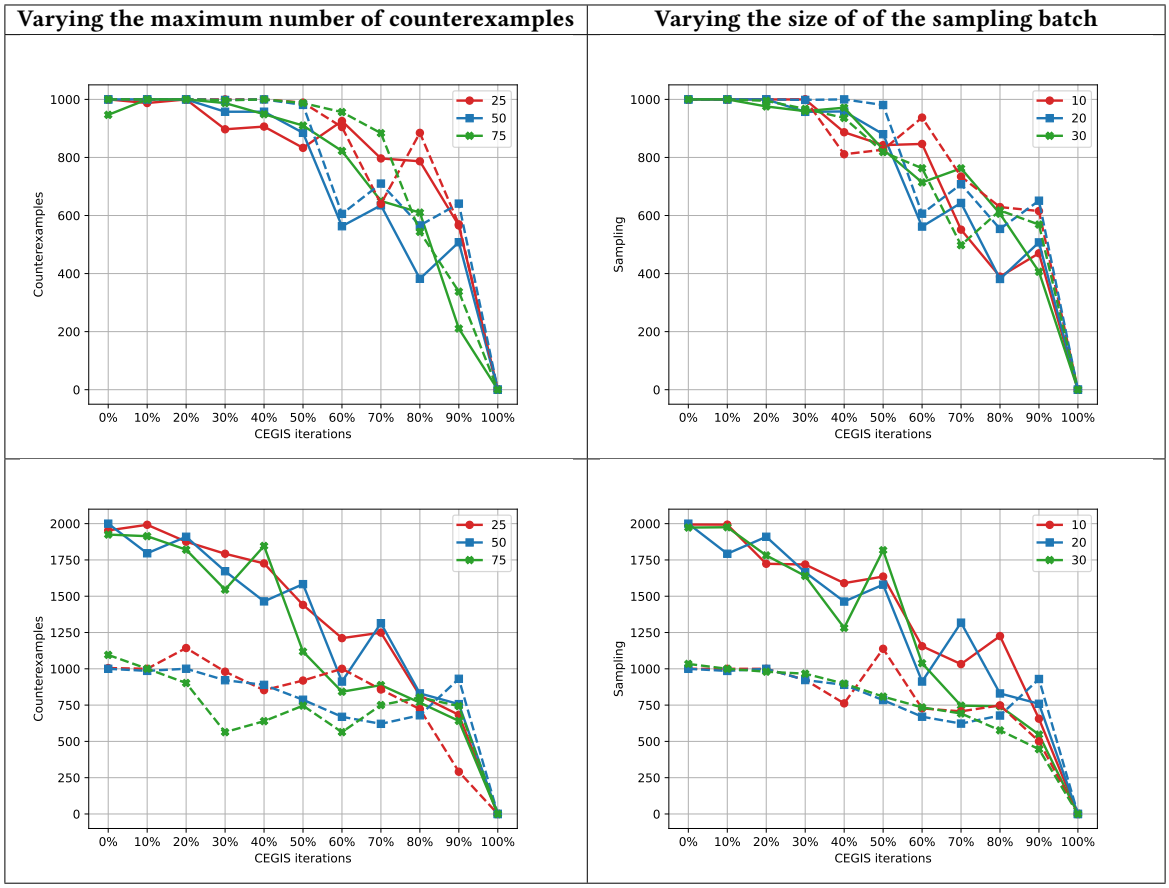
We conducted experiments for all properties and all learning rates discussed above. Each experiment (=choice of specific hyper parameters) was repeated three times in order to compute mean and standard deviation (according to the same timeout used before). What emerges from the data is that soft-acceleration constraints tend to reduce the number of CEGIS iterations in those synthesis processes that, without soft-acceleration, require many CEGIS iterations. The opportunistic counterexample search does indeed countain the number of formal verification calls. Synthesizing more counterexamples boosts convergence of the approach. However, this should be handled carefully as the more counterexamples we synthesize (along with the sampled points) the bigger the dataset at the next training phase. The code and the results (synthesized networks) discussed in this section are available in (Zavatteri 2026d).

9 Horizontal Collision Avoidance System

The Horizontal Collision Avoidance System (HCAS) is a policy for aircraft midair collision avoidance (Julian and Kochenderfer 2019). HCAS is derived from the Airborne Collision Avoidance System X for unmanned aircraft (ACAS Xu), where an ownship aircraft needs to make decisions about the possible maneuvers to execute upon encountering an intruder aircraft, according to some preconditions that involve the geometry of the two aircraft.

Table 4. Convergence curves for Table 3a (left column) and Table 3b (right column). Each curve shows average value of all related experiments that terminated. Solid lines report on data with soft-acceleration. Dashed lines report on data without soft-acceleration. Rows 1-3 correspond to properties $P_1 - P_3$.

Table 5. Convergence curves for Table 3a (left column) and Table 3b (right column). Each curve shows average value of all related experiments that terminated. Solid lines report on data with soft-acceleration. Dashed lines report on data without soft-acceleration. The first row corresponds to P_4, whereas the last row corresponds to $\mathcal{P} = \{P_1, P_4\}$.



Neural networks were proved to be effective in compressing the original policy table. Therefore, neural networks can also act as controllers for advising aircraft maneuvers.

Table 6 reports on inputs and outputs. Inputs describe the position of the intruder aircraft with respect to ownship assumed to lie at coordinates (0,0), whereas outputs provide scores to five possible maneuvers for ownship: clear of conflict, weak left, weak right, strong left, strong right, where the bigger the score the more advised the maneuver. Moreover, the dataset is split in 40 sub-datasets that are parametrized for all combinations of two more features (a_{prev}, τ), where a_{prev} represents the previous advised action and τ represents the time until loss of vertical separation, measured in seconds and discretized for the values 0, 5, 10, 15, 20, 30, 40, 60. As a result, HCAS consists of 40 neural networks[2], one for each combination (a_{prev}, τ).

[2]For each previous action a_{prev} = `X` and discretization of τ = `Y`, the dataset corresponding to the combination (a_{prev}, τ) is the file `HCAS_polar_networks/HCAS_polar_TrainingData_v6_praX_tauY.h5`, whereas the data of all experiments related to this dataset are contained in the folder `experiments/praX-tauY`.

Table 6. Input-output pairs of the HCAS dataset.

Type	Name	Index	Description	Values	Unit
Input	x_ρ	0	Range to intruder (ρ)	$[0, 56000]$	ft
	x_θ	1	Bearing angle to intruder (θ)	$[-\pi, \pi]$	rad
	x_ψ	2	Relative angle of intruder (ψ)	$[-\pi, \pi]$	rad
Output	$y_{\mathbf{coc}}$	0	Clear of Conflict (COC)	(from score table) $[-17.4792, -5.4293 \times 10^{-15}]$ mean: -0.2428 range: 17.48	(real)
	$y_{\mathbf{wl}}$	1	Weak Left (WL)		(real)
	$y_{\mathbf{wr}}$	2	Weak Right (WR)		(real)
	$y_{\mathbf{sl}}$	3	Strong Left (SL)		(real)
	$y_{\mathbf{sr}}$	4	Strong Right (SR)		(real)

We considered 13 properties for this dataset. The first 10 properties are an adaption of the well-known properties for ACAS Xu given in (Katz, Barrett, et al. 2017) and typically used in the neural network verification competition. We adapted them in order to:

- make them coherent with the bounds of input variables in HCAS;
- remove conditions on speed of ownship and intruder (assumed constant in HCAS);
- align them with the fact that in HCAS the advised action is the action with maximal score;
- inject a small $\Delta > \varepsilon$ to compensate the ε-relaxation and guarantee that the output action with maximal value is also unique.

The last 3 properties are conjunctions of the former 10, where each conjunction corresponds is a subset of properties that are realizable together.

Once again, let $F_{Bounds}(\vec{x})$ be a formula restricting the application of the precondition within the allowed input bounds according to Table 6.

Property P_1: If the intruder is distant, then the output score of a COC advisory will always be at least a certain threshold.

$$P_1(\vec{x}, \vec{y}) := \Big(F_{Bounds}(\vec{x}) \land x_\rho \geq 55947.691\Big) \Rightarrow \Big(y_{\mathbf{coc}} \geq -1 + \Delta\Big)$$

Property P_2: If the intruder is distant, then the score of a clear of conflict advisory will never be minimal.

$$P_2(\vec{x}, \vec{y}) := \Big(F_{Bounds}(\vec{x}) \land x_\rho \geq 55947.691\Big) \Rightarrow \Big(\bigvee_{\mathbf{a} \neq \mathbf{coc}} y_{\mathbf{coc}} \geq y_{\mathbf{a}} + \Delta\Big)$$

Property P_3: If the intruder is directly ahead and is moving towards the ownship, the score of a clear of conflict advisory will not be maximal.

$$P_3(\vec{x}, \vec{y}) := \Big(F_{Bounds}(\vec{x}) \land x_\rho \geq 1500 \land x_\rho \leq 1800 \land x_\theta \geq -0.06 \land x_\theta \leq 0.06 \land x_\psi \geq 3.10\Big) \Rightarrow \Big(\bigvee_{\mathbf{a} \neq \mathbf{coc}} y_{\mathbf{a}} \geq y_{\mathbf{coc}} + \Delta\Big)$$

Property P_4: If the intruder is directly ahead and is moving away from the ownship, the score of a clear of conflict advisory will not be maximal.

$$P_4(\vec{x}, \vec{y}) := \Big(F_{Bounds}(\vec{x}) \land x_\rho \geq 1500 \land x_\rho \leq 1800 \land x_\theta \geq -0.06 \land x_\theta \leq 0.06 \land x_\psi = 0\Big) \Rightarrow \Big(\bigvee_{\mathbf{a} \neq \mathbf{coc}} y_{\mathbf{a}} \geq y_{\mathbf{coc}} + \Delta\Big)$$

Property P_5: If the intruder is near and approaching from the left, then strong right is advised.

$$P_5(\vec{x},\vec{y}) := \Big(F_{Bounds}(\vec{x}) \land x_\rho \geq 250 \land x_\rho \leq 400 \land x_\theta \geq 0.2 \land x_\theta \leq 0.4 \land x_\psi \geq -3.141592 \land x_\psi \leq -3.141592 + 0.005\Big)$$
$$\Rightarrow \Big(\bigwedge_{\mathbf{a}\neq\mathbf{sr}} y_{\mathbf{sr}} \geq y_{\mathbf{a}} + \Delta\Big)$$

Property P_6: If the intruder is sufficiently far away, the network advises clear of conflict.

$$\begin{aligned} P_6(\vec{x},\vec{y}) := \Big(&F_{Bounds}(\vec{x}) \land x_\rho \geq 12000 \land x_\rho \leq 56000 \land \\ &((x_\theta \geq 0.7 \land x_\theta \leq 3.141592) \lor (x_\theta \geq -3.141592 \land x_\theta \leq -0.7)) \land \\ &x_\psi \geq -3.141592 \land x_\psi \leq -3.141592 + 0.005\Big) \\ &\Rightarrow \Big(\bigwedge_{\mathbf{a}\neq\mathbf{coc}} y_{\mathbf{coc}} \geq y_{\mathbf{a}} + \Delta\Big) \end{aligned}$$

Property P_7: If vertical separation is large, the network will never advise a strong turn (i.e., neither the score of SR nor that of SL is maximal).

$$P_7(\vec{x},\vec{y}) := \Big(F_{Bounds}(\vec{x}) \land x_\rho \geq 0 \land x_\rho \leq 56000 \land x_\theta \geq -3.141592 \land x_\theta \leq 3.141592 \land x_\psi \geq -3.141592 \land x_\psi \leq 3.141592\Big)$$
$$\Rightarrow \Big(\big(\bigvee_{\substack{\mathbf{a}\neq\mathbf{sr}\\ \mathbf{a}\neq\mathbf{sl}}} y_{\mathbf{sr}} + \Delta \leq y_{\mathbf{a}}\big) \land \big(\bigvee_{\substack{\mathbf{a}\neq\mathbf{sl}\\ \mathbf{a}\neq\mathbf{sr}}} y_{\mathbf{sl}} + \Delta \leq y_{\mathbf{a}}\big)\Big)$$

Property P_8: For a large vertical separation and a previous "weak left" advisory, the network will output COC or continue advising WL. (i.e., the score of COC is maximal or the score of WL is maximal).

$$P_8(\vec{x},\vec{y}) := \Big(F_{Bounds}(\vec{x}) \land x_\rho \geq 0 \land x_\rho \leq 56000 \land x_\theta \geq -3.141592 \land x_\theta \leq -0.75 \cdot 3.141592 \land x_\psi \geq -0.1 \land x_\psi \leq 0.1\Big)$$
$$\Rightarrow \Big(\big(\bigwedge_{\substack{\mathbf{a}\neq\mathbf{coc}\\ \mathbf{a}\neq\mathbf{wl}}} y_{\mathbf{coc}} \geq y_{\mathbf{a}} + \Delta\big) \lor \big(\bigwedge_{\substack{\mathbf{a}\neq\mathbf{wl}\\ \mathbf{a}\neq\mathbf{coc}}} y_{\mathbf{wl}} \geq y_{\mathbf{a}} + \Delta\big)\Big)$$

Property P_9: If the previous advisory was "weak right", the presence of a nearby intruder will cause the network to output a "strong left" advisory instead.

$$P_9(\vec{x},\vec{y}) := \Big(F_{Bounds}(\vec{x}) \land x_\rho \geq 2000 \land x_\rho \leq 7000 \land x_\theta \geq -0.4 \land x_\theta \leq -0.14 \land x_\psi \geq -3.141592 \land x_\psi \leq -3.141592 + 0.01\Big)$$
$$\Rightarrow \Big(\bigwedge_{\mathbf{a}\neq\mathbf{sl}} y_{\mathbf{sl}} \geq y_{\mathbf{a}} + \Delta\Big)$$

Property P_{10}: For a far away intruder, the network advises clear of conflict.

$$P_{10}(\vec{x},\vec{y}) := \Big(F_{Bounds}(\vec{x}) \land x_\rho \geq 36000 \land x_\rho \leq 56000 \land x_\theta \geq 0.7 \land x_\theta \leq 3.141592 \land x_\psi \geq -3.141592 \land x_\psi \leq -3.141592 + 0.01\Big)$$
$$\Rightarrow \Big(\bigwedge_{\mathbf{a}\neq\mathbf{coc}} y_{\mathbf{coc}} \geq y_{\mathbf{a}} + \Delta\Big)$$

We recall that the conjunctions of two realizable properties may not be realizable. This happens whenever the conjunction of their preconditions is satisfiable by some input, while the conjunction of their postconditions is unsatisfiable for the same input and corresponding output. In our set, we have that the sets of properties $\{P_5, P_7\}$ and $\{P_8, P_9\}$ are not realizable. Hence, we selected the set of properties $\mathcal{P}_1 = \{P_1, P_2, P_3, P_4, P_5, P_6, P_{10}\}$,

$\mathcal{P}_2 = \{P_1, P_2, P_3, P_4, P_5, P_6, P_8, P_{10}\}$, and $\mathcal{P}_3 = \{P_1, P_2, P_3, P_4, P_5, P_6, P_9, P_{10}\}$ for our experiments, since they are among the largest realizable combination of properties.

We also excluded from the experiments the test cases where the property (or the set of property) is trivially satisfied by a network. This leads to the following final test cases:

- $P_1, P_2, P_3, P_4, P_6, P_{10}$, and the set of properties $\mathcal{P}_1$ are tested on all 40 networks;
- P_7 is tested on networks where $\tau = 60$ (5 networks);
- P_8 and the set of properties $\mathcal{P}_2$ are tested on the single network where $a_{prev} = \mathbf{WL}$ and $\tau = 60$;
- P_9 and the set of properties $\mathcal{P}_3$ are tested on networks where $a_{prev} = \mathbf{WR}$ (8 networks).

Implementation

We used the same hardware and software versions described in Section 8. We considered a single neural network structure **6x50**: 6 layers with 50 neurons per layer, as this is the size of the networks used in the VNN-COMP. We only considered one learning rate. We ran each experiment once for 3 combinations of ε and Δ, by setting a timeout of 12 hours for each experiment.

All values in the datasets are normalized as follows. Given a value $\mathbf{v}$ (no matter if input or output), the normalization $\mathbf{v_{normalized}}$ of $\mathbf{v}$ is done by computing:

$$\mathbf{v_{normalized}} := \frac{\mathbf{v} - \mathbf{v_{mean}}}{\mathbf{v_{max}} - \mathbf{v_{min}}}$$

For each property, we randomly partitioned the dataset under analysis according to the proportion 80-10-10% for train, validation, and test so that these partitions contain 43000, 5396, and 5396 pairs, respectively. We used the Adam optimizer (Kingma and Ba 2015) with learning rate 10^{-3} and the asymmetric mean square error (MSE) loss function defined in (Julian and Kochenderfer 2019). We trained the networks on the normalized data for at most 1000 epochs by dividing the data in batches of size 512 each. We kept the network with minimal loss on the validation set among the epochs, by applying an early stopping criterion of 50 epochs and we evaluate its accuracy (at the end) on the test set. Every CEGIS iteration starts over with a new network whose weights are randomly initialized. We assigned to both ε and Δ values from the set $\{10^{-2}, 10^{-3}, 10^{-4}\}$ such that $\Delta > \varepsilon$ (such values are not affected by the normalization of the other constants). For soft acceleration we always considered a batch of random points $\hat{B}$ as big as the batch of training data coming from the dataset.

For each property P detailed above, we set up the verification phase as follows: the opportunistic phase samples 1000 input points inside the region of the input space that enables the precondition, then computes the predictions of the current network under analysis and checks if any counterexample exists. If so, it returns a set of counterexamples. Instead, if the opportunistic phase fails, then the formal verification phase looks for one counterexample.

After repairing a single counterexample, we sampled around 20 additional pairs by adding to all continuous inputs and outputs a noise vector. Concretely, we generate a tensor filled with random numbers from a normal distribution with mean 0 and variance 1 and we multiply it by 0.01 before adding it to the input part of the counterexample. We do the same for the output part. We keep a new computed pair only if it satisfies the post-condition of the property under analysis.

Table 7 shows the results of the experimental evaluation on all experiments with and without soft acceleration, respectively. For each property (row of the table) we have aggregated the data of all possible a_{prev} and τ (where the property is applicable) with respect to the specific choice of ε and Δ. In general, the data shows that most experiments terminated within the timeout limit. Notable exceptions include property P_7, where 2 to 3 timeouts out of 5 experiments (per row) were observed depending on the case, and the set of properties $\mathcal{P}_1$, where 2 to 6 timeouts out of 40 experiments (per row) were observed. The former is attributable to the fact that the precondition of P_7 coincides with the whole input space (i.e., the precondition is effectively `true`), while the

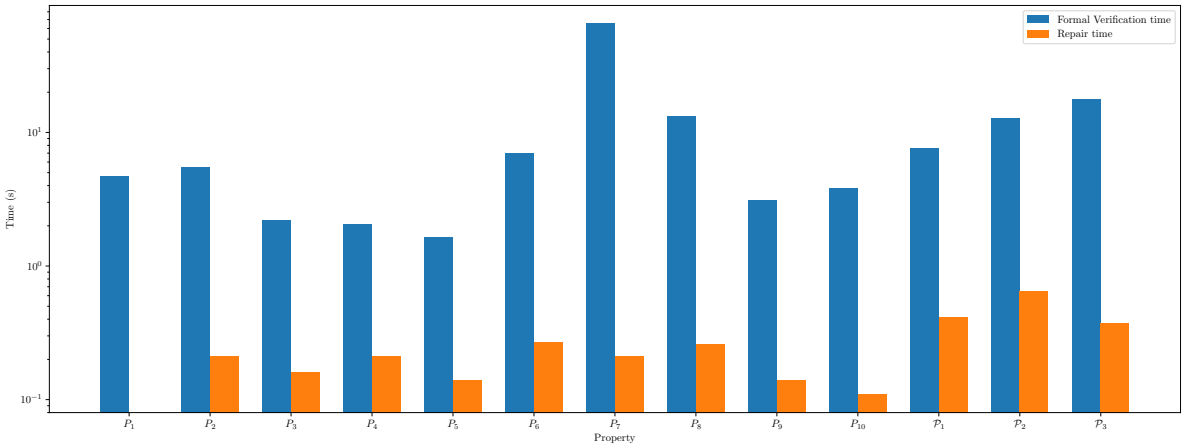


Fig. 8. HCAS: Average formal verification time vs average repair time (log y scale).

latter is due to the simultaneous verification of multiple properties. Figure 8 summarizes the data from Table 7 by plotting the average formal verification time against the average repair time.

Table 8 shows the convergence curves of the corresponding synthesis processes for each property in Table 7. Again, for each network structure, we used a different color according so that solid lines report on data without soft-acceleration, whereas dashed lines report on data with soft-acceleration. Like the MANNERS_DB example, each curve averages the estimations of the number of violations at the 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100% of the whole CEGIS synthesis process according to all experiments belonging to a specific row of the corresponding table. What emerges from the data is the combination $\Delta = 10^{-3}$ and $\varepsilon = 10^{-4}$ seems the one that converges faster. In Appendix A we provide several images for the graphical HCAS policy interpretation of the networks before and after being certified with respect to the discussed properties. The code and the results (synthesized networks) discussed in this section are available in (Zavatteri 2026c).

10 Comparison with DeepSADE and SpecREPAIR

The two closest frameworks to compare our work with are DeepSADE (Goyal et al. 2024) and SpecREPAIR (Bauer-Marquart et al. 2022). For this experimental evaluation, we used the same High Performance Computing (HPC) center using one node equipped with 2 x Intel(R) Xeon(R) CPU E5-2630L v3 (1.80GHz), 1 GPU Nvidia V100, 192GB of RAM, and running Ubuntu 20.04.3 LTS. We run a single job on this node with 16 assigned CPU cores, 100GB of RAM, and the whole GPU of the node.

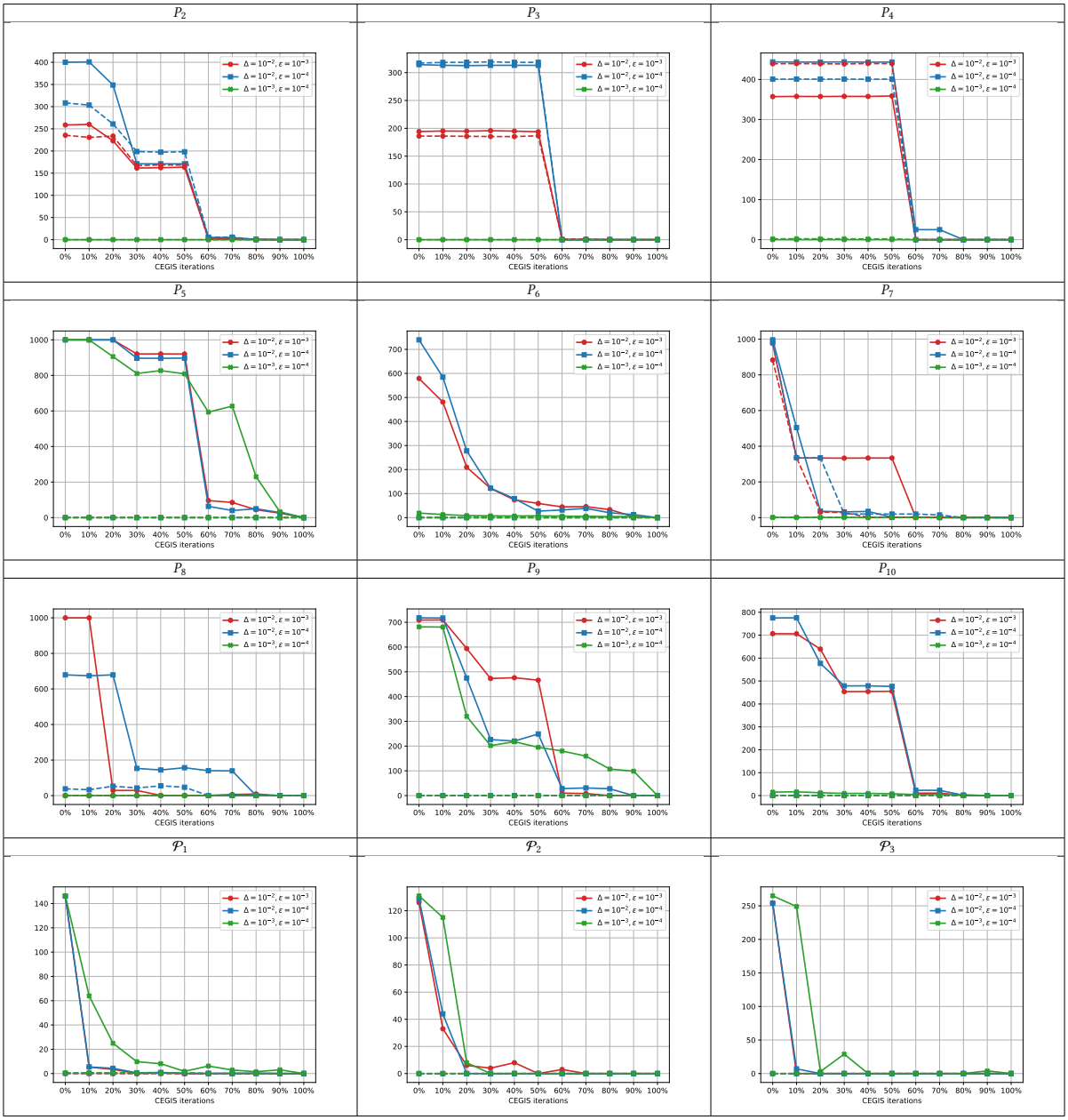
DeepSADE (Goyal et al. 2024) provides an experimental evaluation on 5 usecases (UC1-UC5), of which UC1 is a regression problem, UC2-UC4 are classification problems, and UC5 is a preference learning problem. Since our work targets regression problems, we compared with UC1 only. UC1 is a multi-target regression problem to predict 5 household expenses (All Food Expenses, Going Out Expenses, Shopping Expenses, Necessary Expenses, Miscellaneous Expenses) given 13 features (Number of Television, Number of Car, Jeep, Van, Number of Stove with Oven/Gas Range, House Floor Area, House Age, Number of bedrooms, Electricity, Total Number of Family members, Total number of family members employed, Household Head Sex, Household Head Age, Agricultural Household indicator, Total Household Income). The dataset contains 41417 entries. We considered the following properties:

- P_1: *the sum of all the expenses must be less than or equal to the total household income*

Table 7. Results of the experiments on HCAS ($P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$). Unshaded rows reports on data without soft-acceleration, whereas shaded rows reports on data with soft-acceleration. **P**, **TO**, and **FV** are shorts for property, timeout, and formal verification, respectively. Each line shows average value and standard deviation of all terminated experiments with respect to all applicable networks.

			Average value ± standard deviation													
			Whole process		CEGIS iteration											
	ε	Δ	TO	CEGIS iter.	Total time (s)	Training time (s)	Opportun. search (s)	FV time (s)	Repair time (s)	Initial loss on valid	Final loss on valid	Initial loss on test	Final loss on test	CounterEX (opportun.)	FV calls	New pairs
P_1	10^{-3}	10^{-2}	0	1 ± 0	275.11 ± 115.86	268.86 ± 115.41	0.0 ± 0.0	4.77 ± 1.61	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
			0	1 ± 0	292.35 ± 81.77	286.1 ± 81.75	0.0 ± 0.0	4.81 ± 1.07	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
	10^{-4}	10^{-2}	0	1 ± 0	277.35 ± 92.27	271.44 ± 91.84	0.0 ± 0.0	4.72 ± 1.24	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.003	0 ± 0	1 ± 0	0 ± 0
			0	1 ± 0	300.41 ± 113.5	294.46 ± 113.14	0.0 ± 0.0	4.79 ± 1.37	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.003	0 ± 0	1 ± 0	0 ± 0
		10^{-3}	0	1 ± 0	266.89 ± 95.8	261.19 ± 95.71	0.0 ± 0.0	4.32 ± 1.18	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
			0	1 ± 0	321.71 ± 90.55	315.64 ± 90.64	0.0 ± 0.0	4.69 ± 1.14	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
P_2	10^{-3}	10^{-2}	0	2 ± 1	458.01 ± 263.86	252.98 ± 90.15	0.01 ± 0.01	5.79 ± 5.65	0.26 ± 0.34	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.002	16 ± 23	1 ± 0	292 ± 398
			0	2 ± 1	613.77 ± 325.78	343.83 ± 144.77	0.01 ± 0.01	5.3 ± 5.15	0.24 ± 0.3	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.002	16 ± 22	1 ± 0	283 ± 390
	10^{-4}	10^{-2}	0	2 ± 1	526.74 ± 257.28	236.45 ± 75.99	0.01 ± 0.01	4.02 ± 4.81	0.28 ± 0.43	0.002 ± 0.003	0.002 ± 0.003	0.002 ± 0.003	0.002 ± 0.003	17 ± 22	1 ± 0	299 ± 397
			0	2 ± 1	754.46 ± 371.83	340.65 ± 140.05	0.01 ± 0.01	4.64 ± 6.01	0.29 ± 0.31	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	18 ± 22	1 ± 0	311 ± 396
		10^{-3}	0	1 ± 0	310.88 ± 96.49	301.07 ± 95.97	0.01 ± 0.01	8.45 ± 3.57	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
			0	1 ± 0	325.48 ± 118.03	316.71 ± 117.54	0.0 ± 0.0	7.54 ± 2.25	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
P_3	10^{-3}	10^{-2}	0	1 ± 0	330.79 ± 150.0	260.74 ± 104.51	0.01 ± 0.01	2.11 ± 1.09	0.21 ± 0.63	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	10 ± 20	1 ± 0	204 ± 413
			0	1 ± 1	480.6 ± 250.46	365.81 ± 114.4	0.0 ± 0.01	2.15 ± 1.13	0.28 ± 0.99	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.003	9 ± 19	1 ± 0	179 ± 392
	10^{-4}	10^{-2}	0	1 ± 0	356.41 ± 159.48	255.49 ± 91.35	0.01 ± 0.01	1.93 ± 1.21	0.19 ± 0.32	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	13 ± 22	1 ± 0	265 ± 449
			0	1 ± 0	445.68 ± 208.96	320.46 ± 113.53	0.01 ± 0.01	1.94 ± 1.22	0.2 ± 0.34	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	14 ± 22	1 ± 0	279 ± 460
		10^{-3}	0	1 ± 0	275.06 ± 101.69	271.08 ± 101.69	0.0 ± 0.0	2.66 ± 0.23	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
			0	1 ± 0	325.16 ± 95.57	321.27 ± 95.53	0.0 ± 0.0	2.62 ± 0.23	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
P_4	10^{-3}	10^{-2}	0	1 ± 1	347.64 ± 145.65	240.52 ± 79.8	0.01 ± 0.01	1.94 ± 1.3	0.21 ± 0.33	0.002 ± 0.003	0.002 ± 0.003	0.002 ± 0.003	0.002 ± 0.003	14 ± 23	1 ± 0	291 ± 465
			0	1 ± 1	541.2 ± 223.24	363.55 ± 124.26	0.01 ± 0.01	1.87 ± 1.32	0.24 ± 0.35	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	16 ± 24	1 ± 0	331 ± 484
	10^{-4}	10^{-2}	0	2 ± 1	425.47 ± 181.48	266.91 ± 90.51	0.01 ± 0.01	1.67 ± 1.29	0.25 ± 0.34	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.003	17 ± 24	1 ± 0	355 ± 484
			0	1 ± 1	497.22 ± 212.53	345.26 ± 139.68	0.01 ± 0.01	1.92 ± 1.29	0.43 ± 1.27	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	15 ± 23	1 ± 0	301 ± 467
		10^{-3}	0	1 ± 0	267.11 ± 95.54	261.95 ± 96.25	0.0 ± 0.0	2.81 ± 0.52	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
			0	1 ± 0	347.31 ± 110.48	334.9 ± 97.58	0.0 ± 0.0	2.59 ± 0.48	0.02 ± 0.1	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	1 ± 8	1 ± 0	25 ± 161
P_5	10^{-3}	10^{-2}	0	2 ± 1	626.96 ± 335.21	263.31 ± 72.19	0.01 ± 0.01	1.34 ± 1.32	0.16 ± 0.17	0.002 ± 0.002	0.003 ± 0.003	0.002 ± 0.002	0.003 ± 0.003	23 ± 25	1 ± 1	475 ± 520
			0	1 ± 0	336.42 ± 117.56	332.22 ± 117.39	0.0 ± 0.0	2.82 ± 0.27	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
	10^{-4}	10^{-2}	0	2 ± 1	634.09 ± 488.81	261.18 ± 90.42	0.01 ± 0.01	1.29 ± 1.25	0.16 ± 0.15	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.003	0.003 ± 0.003	21 ± 25	1 ± 2	452 ± 518
			0	1 ± 0	308.89 ± 94.8	304.7 ± 94.73	0.0 ± 0.0	2.9 ± 0.45	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
		10^{-3}	0	4 ± 3	1025.53 ± 898.81	269.06 ± 95.44	0.01 ± 0.01	1.05 ± 1.32	0.21 ± 0.43	0.002 ± 0.003	0.002 ± 0.003	0.002 ± 0.003	0.003 ± 0.003	27 ± 25	2 ± 3	574 ± 520
			0	1 ± 0	334.01 ± 114.94	329.91 ± 114.99	0.0 ± 0.0	2.85 ± 0.42	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
P_6	10^{-3}	10^{-2}	0	6 ± 4	1722.83 ± 1433.9	276.34 ± 91.35	0.01 ± 0.01	6.06 ± 8.84	0.34 ± 1.09	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	17 ± 23	3 ± 4	362 ± 464
			0	1 ± 0	339.55 ± 70.76	321.49 ± 70.66	0.0 ± 0.0	16.79 ± 3.17	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
	10^{-4}	10^{-2}	0	5 ± 2	1526.42 ± 643.29	279.54 ± 87.54	0.01 ± 0.02	5.63 ± 10.47	0.37 ± 1.56	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	21 ± 23	2 ± 1	438 ± 480
			0	1 ± 0	389.28 ± 185.59	325.6 ± 89.76	0.0 ± 0.0	18.71 ± 4.78	0.03 ± 0.1	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.003	0 ± 0	1 ± 0	2 ± 7
		10^{-3}	0	9 ± 7	2573.17 ± 2294.08	279.21 ± 89.07	0.0 ± 0.01	4.9 ± 10.79	0.25 ± 0.79	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	8 ± 10	3 ± 4	161 ± 211
			0	1 ± 0	333.22 ± 119.11	317.46 ± 118.8	0.0 ± 0.0	14.28 ± 2.37	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
P_7	10^{-3}	10^{-2}	2	76 ± 65	23891.15 ± 21127.98	206.9 ± 84.13	0.0 ± 0.0	108.45 ± 188.25	0.24 ± 0.49	0.0 ± 0.0	0.004 ± 0.003	0.0 ± 0.0	0.004 ± 0.003	3 ± 8	37 ± 33	39 ± 92
			2	80 ± 67	28071.72 ± 22530.56	273.39 ± 105.3	0.0 ± 0.0	77.09 ± 175.91	0.21 ± 0.17	0.0 ± 0.0	0.007 ± 0.008	0.0 ± 0.0	0.007 ± 0.008	4 ± 10	24 ± 19	53 ± 108
	10^{-4}	10^{-2}	3	74 ± 100	19514.98 ± 26559.5	223.3 ± 80.62	0.0 ± 0.0	38.17 ± 198.83	0.35 ± 0.98	0.0 ± 0.0	0.009 ± 0.012	0.0 ± 0.0	0.009 ± 0.012	5 ± 11	16 ± 22	68 ± 136
			2	82 ± 69	23993.9 ± 20229.09	246.97 ± 92.74	0.0 ± 0.0	45.31 ± 132.17	0.19 ± 0.13	0.0 ± 0.0	0.007 ± 0.007	0.0 ± 0.0	0.007 ± 0.007	4 ± 9	27 ± 31	53 ± 107
		10^{-3}	2	51 ± 45	11251.17 ± 11096.37	167.36 ± 72.88	0.0 ± 0.0	51.4 ± 82.42	0.17 ± 0.23	0.0 ± 0.0	0.001 ± 0.001	0.0 ± 0.0	0.001 ± 0.001	1 ± 1	25 ± 22	23 ± 15
			2	71 ± 60	19720.46 ± 16345.31	223.15 ± 79.59	0.0 ± 0.0	55.58 ± 106.24	0.16 ± 0.11	0.0 ± 0.0	0.002 ± 0.001	0.0 ± 0.0	0.002 ± 0.001	1 ± 1	36 ± 31	22 ± 15
P_8	10^{-3}	10^{-2}	0	5 ± 0	857.32 ± 0	168.17 ± 0	0.01 ± 0	2.64 ± 0	0.29 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	17 ± 0	1 ± 0	192 ± 0
			0	1 ± 0	341.08 ± 0	315.24 ± 0	0.0 ± 0	24.64 ± 0	0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0 ± 0	1 ± 0	0 ± 0
	10^{-4}	10^{-2}	0	3 ± 0	575.26 ± 0	159.69 ± 0	0.01 ± 0	31.15 ± 0	0.45 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	33 ± 0	1 ± 0	415 ± 0
			0	2 ± 0	639.09 ± 0	310.34 ± 0	0.01 ± 0	8.26 ± 0	0.31 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	20 ± 0	1 ± 0	274 ± 0
		10^{-3}	0	1 ± 0	406.19 ± 0	392.1 ± 0	0.04 ± 0	12.95 ± 0	0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0 ± 0	1 ± 0	0 ± 0
			0	1 ± 0	190.32 ± 0	179.0 ± 0	0.0 ± 0	10.2 ± 0	0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0 ± 0	1 ± 0	0 ± 0
P_9	10^{-3}	10^{-2}	0	2 ± 1	643.93 ± 111.61	267.44 ± 72.92	0.01 ± 0.02	2.86 ± 3.41	0.16 ± 0.16	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.002	0.002 ± 0.003	23 ± 25	1 ± 0	478 ± 501
			0	1 ± 0	351.77 ± 159.01	344.89 ± 158.23	0.0 ± 0.0	5.69 ± 1.47	0 ± 0.0	0.002 ± 0.001	0.002 ± 0.001	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
	10^{-4}	10^{-2}	0	3 ± 1	818.03 ± 317.67	294.65 ± 95.75	0.01 ± 0.01	2.05 ± 2.8	0.19 ± 0.19	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	25 ± 24	1 ± 0	506 ± 491
			0	1 ± 0	295.13 ± 92.69	287.71 ± 90.98	0.0 ± 0.0	6.23 ± 2.56	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
		10^{-3}	0	4 ± 1	1153.87 ± 541.13	268.79 ± 91.26	0.01 ± 0.01	2.08 ± 3.97	0.2 ± 0.16	0.002 ± 0.003	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.002	32 ± 23	1 ± 0	646 ± 466
			0	1 ± 0	319.89 ± 81.27	313.27 ± 81.16	0.0 ± 0.0	5.42 ± 1.76	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
P_{10}	10^{-3}	10^{-2}	0	2 ± 1	684.51 ± 299.01	278.09 ± 106.03	0.01 ± 0.01	2.92 ± 3.87	0.16 ± 0.18	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	22 ± 24	1 ± 0	449 ± 481
			0	1 ± 0	349.49 ± 105.78	341.56 ± 105.31	0.01 ± 0.0	6.16 ± 2.23	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.003	0 ± 0	1 ± 0	0 ± 0
	10^{-4}	10^{-2}	0	3 ± 1	742.18 ± 255.47	266.22 ± 92.45	0.01 ± 0.01	2.53 ± 3.83	0.17 ± 0.18	0.002 ± 0.003	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.002	21 ± 24	1 ± 0	419 ± 472
			0	1 ± 0	324.6 ± 96.89	316.89 ± 97.37	0.01 ± 0.0	5.89 ± 1.26	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
		10^{-3}	0	2 ± 1	542.34 ± 400.28	276.5 ± 78.8	0.02 ± 0.01	3.78 ± 4.35	0.12 ± 0.14	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	11 ± 17	1 ± 0	216 ± 351
			0	1 ± 0	338.11 ± 126.89	331.33 ± 126.66	0.01 ± 0.0	4.95 ± 1.16	0 ± 0.0	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0 ± 0	1 ± 0	0 ± 0
$\mathcal{P}_1$	10^{-3}	10^{-2}	2	21 ± 8	6342.01 ± 2560.11	289.62 ± 87.16	0.01 ± 0.01	6.51 ± 15.1	0.39 ± 0.73	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	39 ± 58	4 ± 2	39 ± 57
			0	3 ± 1	1827.03 ± 1096.22	677.42 ± 229.82	0.02 ± 0.01	17.2 ± 20.7	0.85 ± 2.44	0.002 ± 0.003	0.002 ± 0.003	0.002 ± 0.003	0.003 ± 0.003	32 ± 43	1 ± 0	33 ± 43
	10^{-4}	10^{-2}	6	29 ± 15	8550.13 ± 4089.96	289.23 ± 97.56	0.01 ± 0.01	6.26 ± 15.49	0.42 ± 0.96	0.002 ± 0.002	0.003 ± 0.005	0.002 ± 0.002	0.003 ± 0.003	39 ± 54	5 ± 4	65 ± 128
			0	4 ± 2	2497.09 ± 1463.84	647.2 ± 196.65	0.02 ± 0.01	13.15 ± 18.6	0.62 ± 0.82	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.003	0.003 ± 0.003	32 ± 42	1 ± 1	33 ± 42
		10^{-3}	0	13 ± 5	3838.81 ± 2046.41	280.63 ± 94.14	0.01 ± 0.01	6.16 ± 12.23	0.32 ± 0.71	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	0.002 ± 0.002	27 ± 32	3 ± 2	28 ± 32
			0	1 ± 0	589.95 ± 186.06	517.24 ± 140.63	0.03 ± 0.01	30.3 ± 8.67	0.04 ± 0.15	0.002 ± 0.002	0.002 ± 0.003	0.002 ± 0.003	0.002 ± 0.003	2 ± 11	1 ± 0	2 ± 11
$\mathcal{P}_2$	10^{-3}	10^{-2}	0	9 ± 0	2165.2 ± 0	226.99 ± 0	0.02 ± 0	11.95 ± 0	0.77 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	109 ± 0	1 ± 0	109 ± 0
			0	4 ± 0	2522.68 ± 0	571.14 ± 0	0.02 ± 0	57.48 ± 0	1.25 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	38 ± 0	2 ± 0	38 ± 0
	10^{-4}	10^{-2}	0	26 ± 0	6582.18 ± 0	243.53 ± 0	0.01 ± 0	6.71 ± 0	0.63 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	53 ± 0	2 ± 0	53 ± 0
			0	3 ± 0	2439.3 ± 0	786.1 ± 0	0.02 ± 0	21.91 ± 0	0.83 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	55 ± 0	1 ± 0	55 ± 0
		10^{-3}	0	19 ± 0	3550.41 ± 0	177.18 ± 0	0.01 ± 0	7.91 ± 0	0.52 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	29 ± 0	3 ± 0	29 ± 0
			0	2 ± 0	645.74 ± 0	282.9 ± 0	0.03 ± 0	38.23 ± 0	0.13 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0 ± 0	2 ± 0	0 ± 0
$\mathcal{P}_3$	10^{-3}	10^{-2}	0	32 ± 0	7140.9 ± 0	212.91 ± 0	0.01 ± 0	9.0 ± 0	0.42 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	65 ± 0	5 ± 0	66 ± 0
			0	3 ± 0	1756.86 ± 0	570.18 ± 0	0.02 ± 0	13.56 ± 0	0.54 ± 0	0.001 ± 0	0.001 ± 0	0.001 ± 0	0.001 ± 0	24 ± 0	1 ± 0	24 ± 0
	10^{-4}	10^{-2}	0	60 ± 0	13315.13 ± 0	215.38 ± 0	0.01 ± 0	6.1 ± 0	0.31 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	44 ± 0	7 ± 0	44 ± 0
			0	2 ± 0	2176.31 ± 0	470.27 ± 0	0.02 ± 0	616.12 ± 0	0.91 ± 0	0.001 ± 0	0.001 ± 0	0.001 ± 0	0.001 ± 0	50 ± 0	1 ± 0	50 ± 0
		10^{-3}	0	18 ± 0	3523.77 ± 0	185.87 ± 0	0.01 ± 0	3.85 ± 0	0.41 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	68 ± 0	2 ± 0	68 ± 0
			0	1 ± 0	913.39 ± 0	863.6 ± 0	0.03 ± 0	41.06 ± 0	0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0 ± 0	1 ± 0	0 ± 0

Table 8. Convergence curves for Table 7 (excluding P_1). Each curve shows average value of all corresponding experiments that terminated. Solid lines report on data with soft-acceleration. Dashed lines report on data without soft-acceleration.



- P_2: *going out expense must be less than or equal to 5% of the household income*
- $\mathcal{P} := \{P_1, P_2\}$

Table 9. Results for the Expense prediction case study (soft-acceleration only).

		Average value ± standard deviation												
		Generic synthesis process		Generic CEGIS iteration										
	TO	CEGIS iter.	Total time (s)	Training time (s)	Opportun. search (s)	FV time (s)	Repair time (s)	Initial loss on valid	Final loss on valid	Initial loss on test	Final loss on test	CounterEX (opportun.)	FV calls	New pairs
P_1	0	1 ± 0	107.76 ± 24.63	104.51 ± 24.43	0.0 ± 0.0	2.74 ± 0.28	0 ± 0.0	0.001 ± 0.0	0.001 ± 0.0	0.001 ± 0.0	0.001 ± 0.0	0 ± 0	1 ± 0	0 ± 0
P_2	0	5 ± 5	1224.37 ± 1167.86	226.11 ± 76.15	0.0 ± 0.0	3.01 ± 4.0	0.13 ± 0.11	0.0 ± 0.0	0.001 ± 0.0	0.0 ± 0.0	0.001 ± 0.0	0 ± 0	5 ± 5	7 ± 4
$\mathcal{P}$	0	4 ± 2	1008.6 ± 664.33	270.27 ± 76.33	0.0 ± 0.0	4.45 ± 3.46	0.16 ± 0.12	0.0 ± 0.0	0.001 ± 0.0	0.0 ± 0.0	0.001 ± 0.0	0 ± 0	4 ± 2	7 ± 5

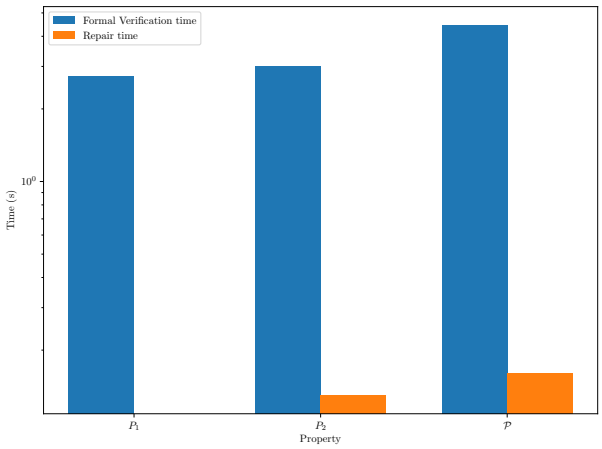


Fig. 9. Expense prediction: Average formal verification time vs average repair time aggregated per property (log y scale).

The original experiments from (Goyal et al. 2024) targeted only the conjunction of the two properties $\mathcal{P}$. In our experiments we also targeted P_1 and P_2 in isolation. We tested our workflow on a ReLU network with 13 inputs, 5 outputs, 10 hidden layers and 6 neurons per hidden layer. We normalized each input and output by following a min-max scale. For each property, we randomly partitioned the dataset in train, validation, and test according to the division 70%, 10%, 20%. We used the Adam optimizer with learning rate of 10^{-3} and mean square error (MSE) loss function. We trained the networks on the normalized data for at most 100 epochs by dividing the data in batches of size 100 each. We kept the network with minimal loss on the validation set among the epochs, by applying an early stopping criterion of 20 epochs and we evaluate its accuracy (at the end) on the test set. We used soft-acceleration constraints, $\varepsilon = 0.002$ and $\Delta = 0.004$ in the formulation of the properties (roughly, 0.002 is the normalization of 1 currency unit). The opportunistic phase generates 1000 points in order to find 50 counterexamples and each counterexample is repaired and strengthened by adding up to 20 points around. We run our experimental evaluation with a 24-hour timeout, running each experiment 3 times. Table 9 reports the results, whereas Figure 9 report formal verification vs repair time. Notably, DeepSADE timed out in all runs, which is consistent with the original paper, where the authors reported a runtime of 28 hours for the same experiment. The code and the results (synthesized networks) for the Expense Prediction case study are available in (Zavatteri 2026b).

The last case study that we consider is the original ACAS Xu benchmark from (Katz, Barrett, et al. 2017). As noted earlier, while the 45 networks of the benchmark are publicly available, the dataset used to train them is not. We therefore generated a synthetic dataset by uniformly sampling the publicly available networks, producing 45 datasets of 10,000 entries each, one per network. Therefore, all results reflect the fidelity to the original neural

Table 10. ACAS Xu results (soft-acceleration only).

		Average value ± standard deviation												
		Generic synthesis process		Generic CEGIS iteration										
	TO	CEGIS iter.	Total time (s)	Training time (s)	Opportun. search (s)	FV time (s)	Repair time (s)	Initial loss on valid	Final loss on valid	Initial loss on test	Final loss on test	CounterEX (opportun.)	FV calls	New pairs
P_1	0	1 ± 0	132.42 ± 33.05	129.09 ± 32.89	0.01 ± 0.0	2.95 ± 0.24	0 ± 0.0	0.013 ± 0.014	0.013 ± 0.014	0.012 ± 0.013	0.012 ± 0.013	0 ± 0	1 ± 0	0 ± 0
P_2	1	5 ± 11	1391.42 ± 4090.11	265.74 ± 167.43	0.0 ± 0.01	25.91 ± 165.34	0.99 ± 0.82	0.013 ± 0.014	0.013 ± 0.014	0.012 ± 0.013	0.013 ± 0.013	32 ± 23	1 ± 1	548 ± 392
P_3	0	5 ± 7	1349.11 ± 3014.57	267.85 ± 174.82	0.01 ± 0.02	21.63 ± 82.77	0.83 ± 0.63	0.013 ± 0.014	0.013 ± 0.014	0.013 ± 0.014	0.013 ± 0.014	34 ± 23	1 ± 1	710 ± 474
P_4	0	5 ± 6	1171.51 ± 2252.34	247.67 ± 143.28	0.0 ± 0.0	5.33 ± 37.81	0.74 ± 0.47	0.013 ± 0.013	0.013 ± 0.014	0.013 ± 0.014	0.013 ± 0.014	34 ± 23	1 ± 0	713 ± 472
P_5	0	1 ± 0	142.32 ± 37.48	136.41 ± 36.87	0.01 ± 0.0	5.49 ± 2.22	0 ± 0.0	0.013 ± 0.014	0.013 ± 0.014	0.013 ± 0.014	0.013 ± 0.014	0 ± 0	1 ± 0	0 ± 0
P_6	5	3 ± 3	2215.01 ± 1989.07	165.38 ± 56.49	0.01 ± 0.01	639.49 ± 853.15	0.11 ± 0.12	0.014 ± 0.015	0.015 ± 0.015	0.013 ± 0.013	0.013 ± 0.013	4 ± 13	2 ± 1	100 ± 265
P_7	4	1 ± 0	363.18 ± 0	81.48 ± 0	0.02 ± 0	281.3 ± 0	0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0 ± 0	1 ± 0	0 ± 0
P_8	1	-	-	-	-	-	-	-	-	-	-	-	-	-
P_9	0	1 ± 0	142.65 ± 46.12	137.54 ± 43.76	0.01 ± 0.0	4.71 ± 2.7	0 ± 0.0	0.012 ± 0.014	0.012 ± 0.014	0.018 ± 0.021	0.018 ± 0.021	0 ± 0	1 ± 0	0 ± 0
P_{10}	1	2 ± 4	419.05 ± 577.49	142.75 ± 56.54	0.01 ± 0.01	24.28 ± 22.4	0.16 ± 0.48	0.013 ± 0.014	0.013 ± 0.014	0.013 ± 0.014	0.013 ± 0.014	0 ± 2	2 ± 3	20 ± 41

networks, not to any ground-truth phenomenon such as retraining from genuine real flight data augmented with repaired counterexamples.

We analyzed the original 10 properties from (Katz, Barrett, et al. 2017), following a certification process analogous to that described for HCAS, with the exception that a sampling batch of 100 was used. The parameters were set to $\varepsilon = 10^{-4}$ and $\Delta = 10^{-3}$. As in the previous case studies, we synthesize up to 50 counterexamples per CEGIS iteration, which are then repaired and augmented with 20 additional points sampled in their neighborhood. We set a timeout of 12 hour per experiment. The results are reported in Table 10, whereas the formal verification vs repair time is given in Figure 10. With respect to the HCAS case study we observe that we get more timeouts than Table 7. This is reasonable to expect by keeping the same timeout as the ACAS Xu case study has a bigger input space than HCAS: 5 inputs instead of 3 (i.e., the same inputs of Table 6 augmented with the velocities of ownship and intruder).

For comparison with SpecREPAIR, we attempted to reproduce the authors' environment following their instructions[3]. While the installation succeeded, the software crashed when running the ACAS Xu test case, making a direct comparison unfeasible. We recovered several "Illegal instruction" errors from the log of the HPC job, which are typically indicative of a hardware incompatibility between the CPU architecture of the HPC we used and one or more of the libraries employed by SpecREPAIR.

The code and synthesized networks for this case study are publicly available at (Zavatteri 2026a).

11 Conclusions and Future Work

We proposed a method based on the CEGIS workflow for the automated synthesis of certified neural networks. The certification part is based on formal verification of neural networks, whereas the overall synthesis exploits information on counterexamples to repair them and build new input-output pairs to add to the current dataset. An important point of our approach is that it is modular, meaning that our approach can be applied off-the-shelf to other kinds of neural networks and properties provided there exist a formal verification and a counterexample repair part able to handle the corresponding constraints. With respect to the work in (Zavatteri et al. 2024), in this paper we extended the discussion by considering larger neural networks, providing the notion of ε-certification, considering more complex properties, opportunistic search and soft-acceleration constraints, and proving termination for a whole class of properties under reasonable assumptions. Last but not least, our approach could certify neural networks for 4 different case studies. Our software prototypes, data, and instructions are available on github to ease reproducibility of our experiments. Furthermore, our software is already prepared to run many more experimental evaluations with respect to different combinations of hyperparameters that can be specified in input. In our experimental setup, we deliberately limited the number of trials to 3 repetitions per configuration (beside HCAS and ACASXu that average on many networks) in order to explore a wide

[3] https://github.com/sen-uni-kn/specrepair

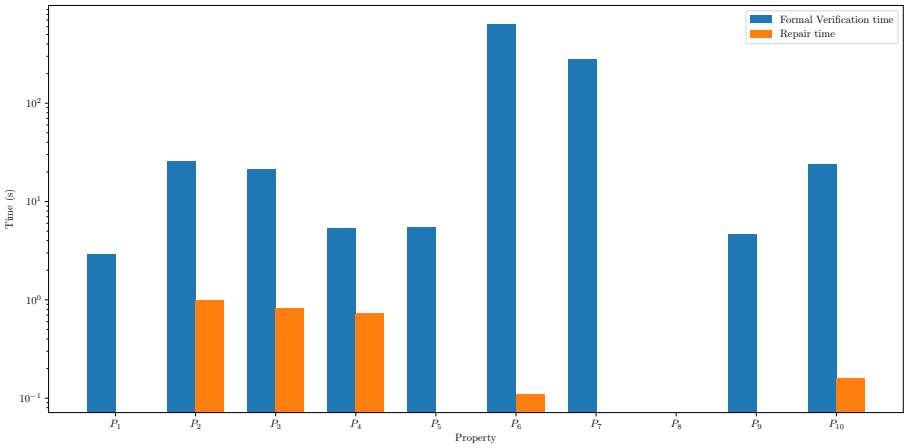


Fig. 10. ACAS Xu: Average formal verification time vs average repair time aggregated per property (log y scale).

range of hyperparameter combinations within a reasonable amount of time. With such a small sample size per configuration, applying statistical tests would not yield meaningful conclusions and could give a misleading impression of statistical rigor. We therefore consider this a trade-off between breadth and depth of the experimental analysis. We also point out that while soft-acceleration constraints tend to reduce the number of CEGIS iterations, a statistical evaluation of the empirical computational improvement provided by the adoption of soft-acceleration is outside the scope of the paper. A more statistically rigorous evaluation is left for future work.

As future work, there are several directions to consider.

Generalized Repair. Besides the scalability issues of formal verification, our generalized repair approach currently requires handling all permutations of the pairs found in each counterexample together with those repaired in previous iterations. As a result, the corresponding MIQPs become bigger and bigger at every CEGIS iteration. An important open question is whether approximation techniques can be devised to mitigate this issue.

Strict Inequalities. The property language considered in this paper is restricted to non-strict inequalities, primarily because MIQP solvers require non-strict constraints to guarantee the existence of optimal solutions. Extending the framework to support strict inequalities would require either the design of alternative MIQP formulations, or the development of a different approach for counterexample repair.

Expressiveness. We plan to investigate neural networks with activation functions beyond ReLU, as well as property languages involving non-linear constraints. A further direction concerns the termination guarantees of Algorithm 2, at least for specific classes of multi-pair properties.

Safe RL. We will explore more complex settings, such as reinforcement learning, to extend this approach towards the synthesis of certified neural controllers.

Fairness We will investigate whether the proposed framework can be extended to enforce *fairness constraints*, and to what extent the augmented dataset generated to certify a specific network architecture can be transferred to other architectures.

Physics Informed Neural Networks. A further research direction concerns the application of this method to the certification of physics-informed neural networks (PINN) (Raissi et al. 2019). Since training in PINN incorporates physical laws as constraints, any data generated during the repair process must be verified to comply with such physical laws.

Acknowledgments

This work was supported by the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.5 - Call for tender No. 3277 of 30 December 2021 of Italian Ministry of University and Research funded by the European Union – NextGenerationEU; project code: ECS00000043, Concession Decree No. 1058 of June 23, 2022, CUP C43C22000340006, project title "iNEST: Interconnected Nord-Est Innovation Ecosystem", and by Mission 4 Component 2 Investment 1.3 - Call for tender No. 341 of March 15, 2022 of Italian Ministry of University and Research – NextGenerationEU; project code PE0000013, Concession Decree No. 1555 of October 11, 2022, CUP C63C22000770006, project title "Future AI Research (FAIR) - Spoke 2 Integrative AI - Symbolic conditioning of Graph Generative Models (SymboliG)".

This work was also supported by the Project iNEST Young Researchers "Neuro-Symbolic AI in digital twins", and by the INdAM-GNCS 2024 Project "Certificazione, monitoraggio, ed interpretabilità in sistemi di intelligenza artificiale" CUP_E53C23001670001.

References

A. Abate, D. Ahmed, M. Giacobbe, and A. Peruffo. 2021. "Formal Synthesis of Lyapunov Neural Networks." *IEEE Control. Syst. Lett.*, 5, 3, 773–778.

A. Albarghouthi. 2021. "Introduction to Neural Network Verification." *Found. Trends Program. Lang.*, 7, 1-2, 1–157.

M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. 2018. "Safe Reinforcement Learning via Shielding." In: *AAAI 2018*. AAAI Press, 2669–2678.

A. Araujo, A. J. Havens, B. Delattre, A. Allauzen, and B. Hu. 2023. "A Unified Algebraic Perspective on Lipschitz Neural Networks." *CoRR*, abs/2303.03169. doi:10.48550/ARXIV.2303.03169.

F. Bauer-Marquart, D. Boetius, S. Leue, and C. Schilling. 2022. "SpecRepair: Counter-Example Guided Safety Repair of Deep Neural Networks." In: *Model Checking Software*. Ed. by O. Legunsen and G. Rosu. Springer International Publishing, 79–96.

D. Boetius, S. Leue, and T. Sutter. 2023. "A robust optimisation perspective on counterexample-guided repair of neural networks." In: *Proceedings of the 40th International Conference on Machine Learning* (ICML'23) Article 114. JMLR.org, 26 pages.

C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu. 2023. "First three years of the international verification of neural networks competition (VNN-COMP)." *Int. J. Softw. Tools Technol. Transf.*, 25, 3, 329–339.

Y. Chang, N. Roohi, and S. Gao. 2019. "Neural Lyapunov Control." In: *NeurIPS*, 3240–3249.

K. Chatterjee, T. A. Henzinger, M. Lechner, and D. Zikelic. 2023. "A Learner-Verifier Framework for Neural Network Controllers and Certificates of Stochastic Systems." In: *TACAS (1)* (Lecture Notes in Computer Science). Vol. 13993. Springer, 3–25.

A. Church. 1964. "Logic, Arithmetic, and Automata." *Journal of Symbolic Logic*, 29, 4, 210–210. doi:10.2307/2270398.

C. Cornelio, J. Stuehmer, S. X. Hu, and T. M. Hospedales. 2023. "Learning where and when to reason in neuro-symbolic inference." In: *ICLR 2023*. OpenReview.net.

A. Daniele and L. Serafini. 2022. "Knowledge Enhanced Neural Networks for Relational Domains." In: *AIxIA 2022* (LNCS). Vol. 13796. Springer, 91–109.

J. Deng, N. Ding, Y. Jia, A. Frome, K. Murphy, S. Bengio, Y. Li, H. Neven, and H. Adam. 2014. "Large-Scale Object Classification Using Label Relation Graphs." In: *ECCV 2014* (LNCS). Vol. 8689. Springer, 48–64.

P. Dragone, S. Teso, and A. Passerini. 2021. "Neuro-Symbolic Constraint Programming for Structured Prediction." In: *(IJCLR 2021)* (CEUR Workshop Proceedings). Vol. 2986. CEUR-WS.org, 6–14.

EASA and Collins Aerospace. Apr. 2023. *Formal Methods use for Learning Assurance (ForMuLA)*. Tech. rep. (Apr. 2023). https://www.easa.europa.eu/en/downloads/137878/en.

European Parliament. 2024. *Artificial intelligence act.* (2024). https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:52021PC0206.

E. Giunchiglia and T. Lukasiewicz. 2021. "Multi-Label Classification Neural Networks with Hard Logical Constraints." *JAIR*, 72, 759–818.

E. Giunchiglia, M. C. Stoian, and T. Lukasiewicz. 2022. "Deep Learning with Logical Constraints." In: *IJCAI 2022*. ijcai.org, 5478–5485.

B. Goldberger, G. Katz, Y. Adi, and J. Keshet. 2022. "Minimal Modifications of Deep Neural Networks using Verification." In: *Proc. of LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning* (EPiC Series in Computing). Vol. 73, 260–240. doi:10.29007/699q.

K. Goyal, S. Dumancic, and H. Blockeel. 2024. "DeepSaDe: learning neural networks that guarantee domain constraint satisfaction." In: *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence* (AAAI'24/IAAI'24/EAAI'24) Article 1361. AAAI Press, 9 pages. ISBN: 978-1-57735-887-9. doi:10.1609/aaai.v38i11.29109.

Gurobi Optimization, LLC. 2024. *Gurobi Optimizer Reference Manual*. (2024). https://www.gurobi.com.
N. Hoernle, R. Karampatsis, V. Belle, and K. Gal. 2022. "MultiplexNet: Towards Fully Satisfied Logical Constraints in Neural Networks." In: *AAAI 2022*. AAAI Press, 5700–5709.
K. Hu, K. Leino, Z. Wang, and M. Fredrikson. 2024. "A Recipe for Improved Certifiable Robustness." In: *The Twelfth International Conference on Learning Representations, ICLR 2024*. OpenReview.net. https://openreview.net/forum?id=qz3mcn99cu.
M. A. Jette and T. Wickberg. 2023. "Architecture of the Slurm Workload Manager." In: *Job Scheduling Strategies for Parallel Processing*. Springer Nature Switzerland, Cham, 3–23.
K. D. Julian and M. J. Kochenderfer. 2019. "Guaranteeing safety for neural network-based aircraft collision avoidance systems." In: *DASC*.
G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. 2017. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks." In: *CAV 2017* (LNCS). Vol. 10426. Springer, 97–117.
G. Katz, D. A. Huang, et al.. 2019. "The Marabou Framework for Verification and Analysis of Deep Neural Networks." In: *CAV 2019* (LNCS). Vol. 11561. Springer, 443–452.
D. P. Kingma and J. Ba. 2015. "Adam: A Method for Stochastic Optimization." In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. http://arxiv.org/abs/1412.6980.
T. Li, V. Gupta, M. Mehta, and V. Srikumar. 2019. "A Logic-Driven Framework for Consistency of Neural Models." In: *EMNLP-IJCNLP 2019*. Association for Computational Linguistics, 3922–3933.
T. Li and V. Srikumar. 2019. "Augmenting Neural Networks with First-order Logic." In: *ACL 2019*. Association for Computational Linguistics, 292–302.
X. Liu, X. Han, N. Zhang, and Q. Liu. 2020. "Certified Monotonic Neural Networks." In: *NeurIPS 2020*.
P. Minervini and S. Riedel. 2018. "Adversarially Regularising Neural NLI Models to Integrate Logical Background Knowledge." In: *CoNLL 2018*. Association for Computational Linguistics, 65–74.
S. Park, C. Yun, J. Lee, and J. Shin. 2021. "Minimum Width for Universal Approximation." In: *International Conference on Learning Representations*. https://openreview.net/forum?id=O-XJwyoIF-k.
A. Peruffo, D. Ahmed, and A. Abate. 2021. "Automated and Formal Synthesis of Neural Barrier Certificates for Dynamical Models." In: *TACAS (1)* (Lecture Notes in Computer Science). Vol. 12651. Springer, 370–388.
M. Raissi, P. Perdikaris, and G. Karniadakis. 2019. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." *Journal of Computational Physics*, 378, 686–707. doi:https://doi.org/10.1016/j.jcp.2018.10.045.
A. Sivaraman, G. Farnadi, T. D. Millstein, and G. V. den Broeck. 2020. "Counterexample-Guided Learning of Monotonic Neural Networks." In: *NeurIPS 2020*.
A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, and V. A. Saraswat. 2006. "Combinatorial sketching for finite programs." In: *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS*. ACM, 404–415.
A. Tacchella, L. Pulina, D. Guidotti, and S. Demarchi. 2022. *VNN-LIB, https://www.vnnlib.org*. (2022).
J. Tjomsland, S. Kalkan, and H. Gunes. 2022. "Mind Your Manners! A Dataset and a Continual Learning Approach for Assessing Social Appropriateness of Robot Actions." *Frontiers Robotics AI*, 9, 669420.
W. Wang and S. J. Pan. 2020. "Integrating Deep Learning with Logic Fusion for Information Extraction." In: *AAAI 2020*. AAAI Press, 9225–9232.
Y. Xie, Z. Xu, K. S. Meel, M. S. Kankanhalli, and H. Soh. 2019. "Embedding Symbolic Knowledge into Deep Networks." In: *NeurIPS 2019*, 4235–4245.
J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. V. den Broeck. 2018. "A Semantic Loss Function for Deep Learning with Symbolic Knowledge." In: *ICML 2018*. Vol. 80. PMLR, 5498–5507.
M. Zavatteri. 2026a. *ACAS Xu experiments*. (2026). https://github.com/matteozavatteri/Certified-NN-JAIR-2026-ACASXu.
M. Zavatteri. 2026b. *Expense Prediction experiments*. (2026). https://github.com/matteozavatteri/Certified-NN-JAIR-2026-EP.
M. Zavatteri. 2026c. *Horizontal CAS experiments*. (2026). https://github.com/matteozavatteri/Certified-NN-JAIR-2026-HCAS.
M. Zavatteri. 2026d. *MIND YOUR MANNERS experiments*. (2026). https://github.com/matteozavatteri/Certified-NN-JAIR-2026-MANNERSDB.
M. Zavatteri, D. Bresolin, and N. Navarin. 2024. "Automated Synthesis of Certified Neural Networks." In: *ECAI 2024 - 27th European Conference on Artificial Intelligence* (Frontiers in Artificial Intelligence and Applications). Vol. 392. IOS Press, 1341–1348. doi:10.3233/FAIA240633.
H. Zhang, T. Weng, P. Chen, C. Hsieh, and L. Daniel. 2018. "Efficient Neural Network Robustness Certification with General Activation Functions." In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, 4944–4953.
H. Zhu, Z. Xiong, S. Magill, and S. Jagannathan. 2019. "An inductive synthesis framework for verifiable reinforcement learning." In: *ACM 2019*. ACM, 686–701.

A Policy Visualization

One advantage of the HCAS dataset is that the advising policy can be visualized on a polar plane once we fix ψ to some value and feed the neural network to compute the scores for the actions, and finally we use a different color for each action and draw a corresponding scatter point.

Each image shows a control policy for a network that we interpret as follows:

- Ownship is at the origin $(0, 0)$ heading right in the image;
- The color at any other point (ρ, θ) is the action that ownship is advised to take should the intruder be at (ρ, θ) according to the fixed ψ;
- The preconditions areas of the properties (mostly given as intervals on ρ and θ) are shown as red perimeters.

Tables 11,12,13,14,15,16,17,18,19,20 show the graphical representation of the policies for $P_1 - P_{10}$ according to these conventions (we omit $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ because drawing the union of their preconditions would make the images quite hard to understand). For each property (table), we show the first network indexed by (a_{prev}, τ) where the property is applicable with respect to all ε, Δ. Specifically, in each table:

- the images in the first (top) row always refer to the combination $\varepsilon = 10^{-3}$ and $\Delta = 10^{-2}$;
- the images in the second (middle) row always refer to the combination $\varepsilon = 10^{-4}$ and $\Delta = 10^{-2}$;
- the images in the third (bottom) row always refer to the combination $\varepsilon = 10^{-4}$ and $\Delta = 10^{-3}$.

Table 11. HCAS certified policy synthesis for P_1. The networks are synthesized for $a_{prev} = \text{COC}$, $\tau = 0$. In all inputs we fix $\psi = -\pi$.

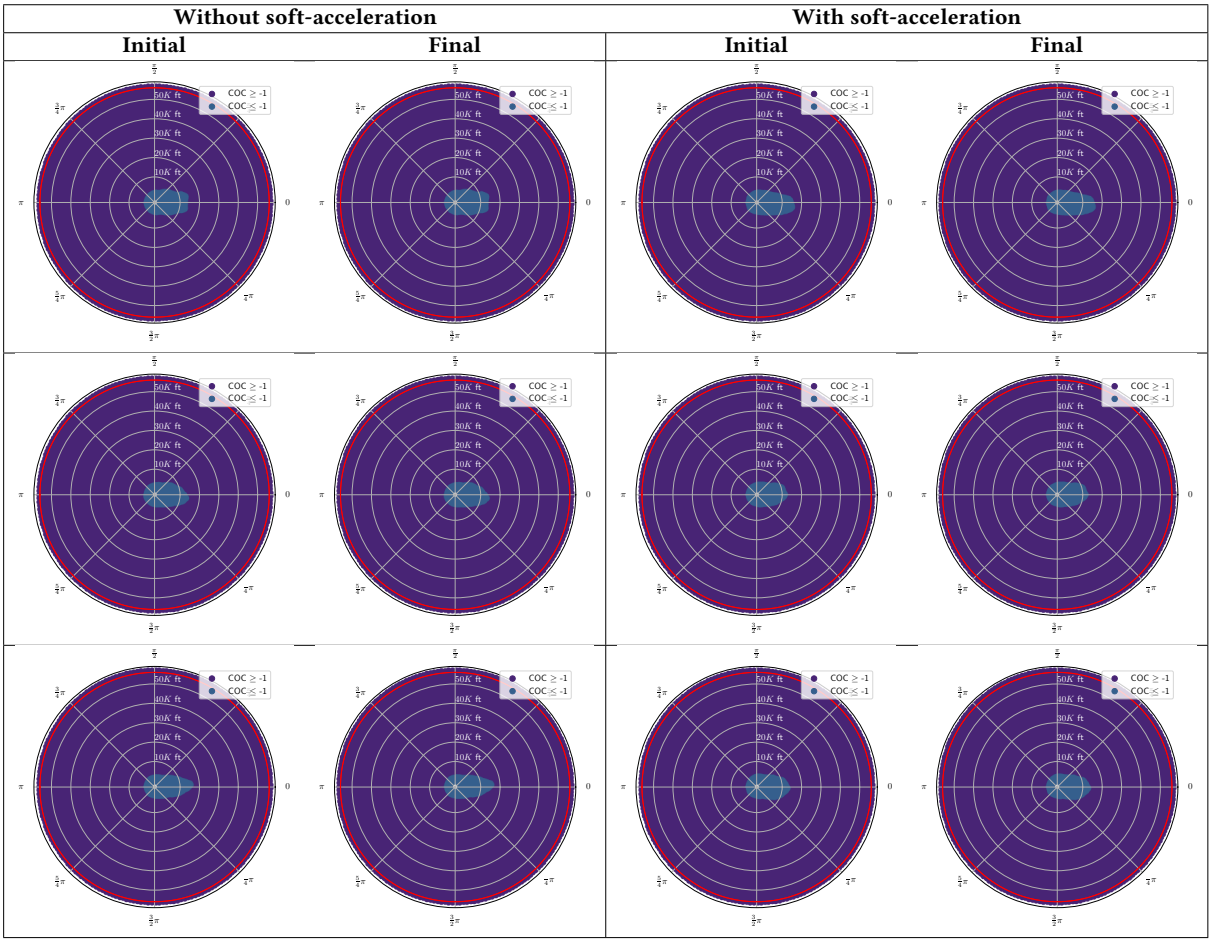


Table 12. HCAS certified policy synthesis for P_2. The networks are synthesized for $a_{prev} = \mathbf{COC}$, $\tau = 0$. In all inputs we fix $\psi = -\pi$.

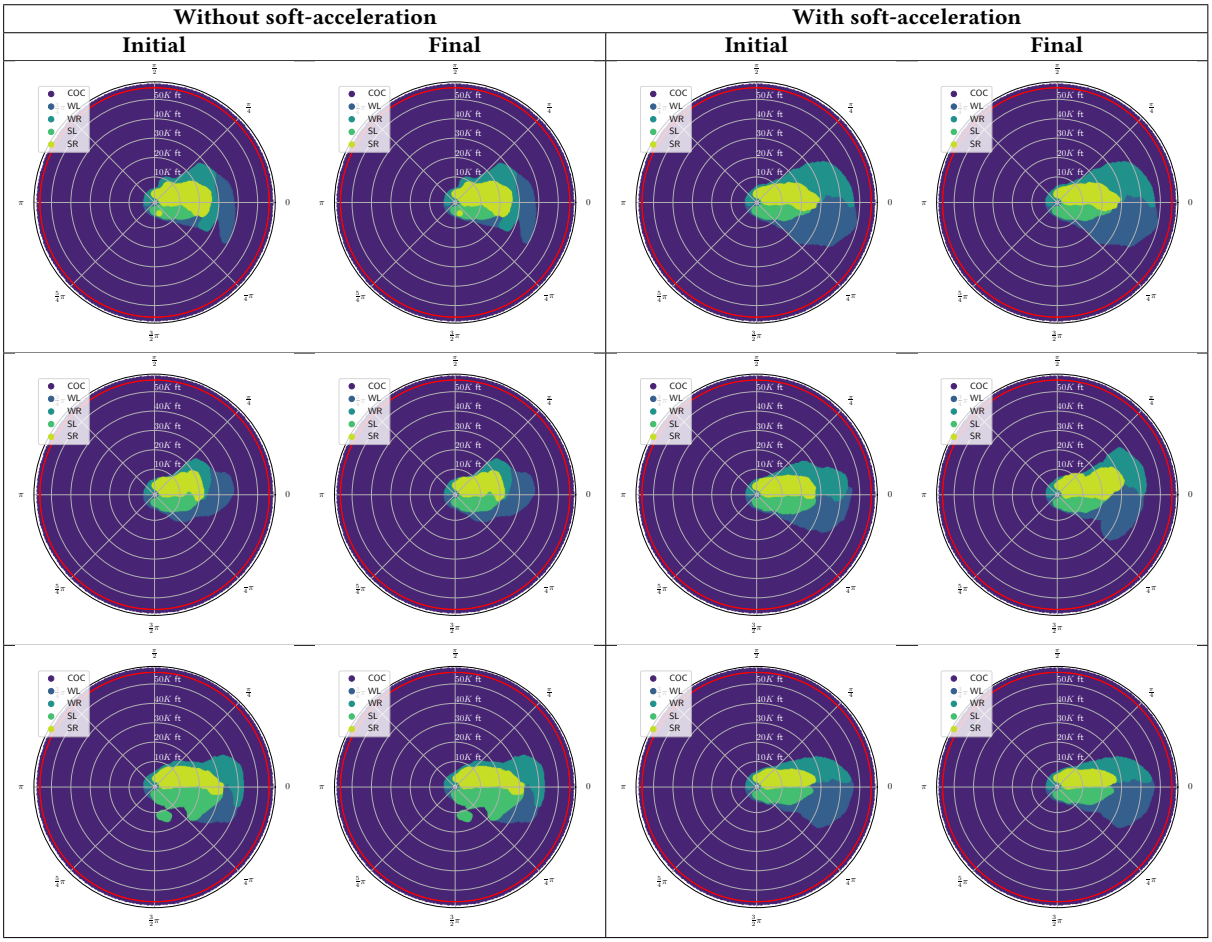


Table 13. HCAS certified policy synthesis for P_3. The networks are those for $a_{prev} = \mathbf{COC}$, $\tau = 0$. In all inputs we fix $\psi = 3.10$.

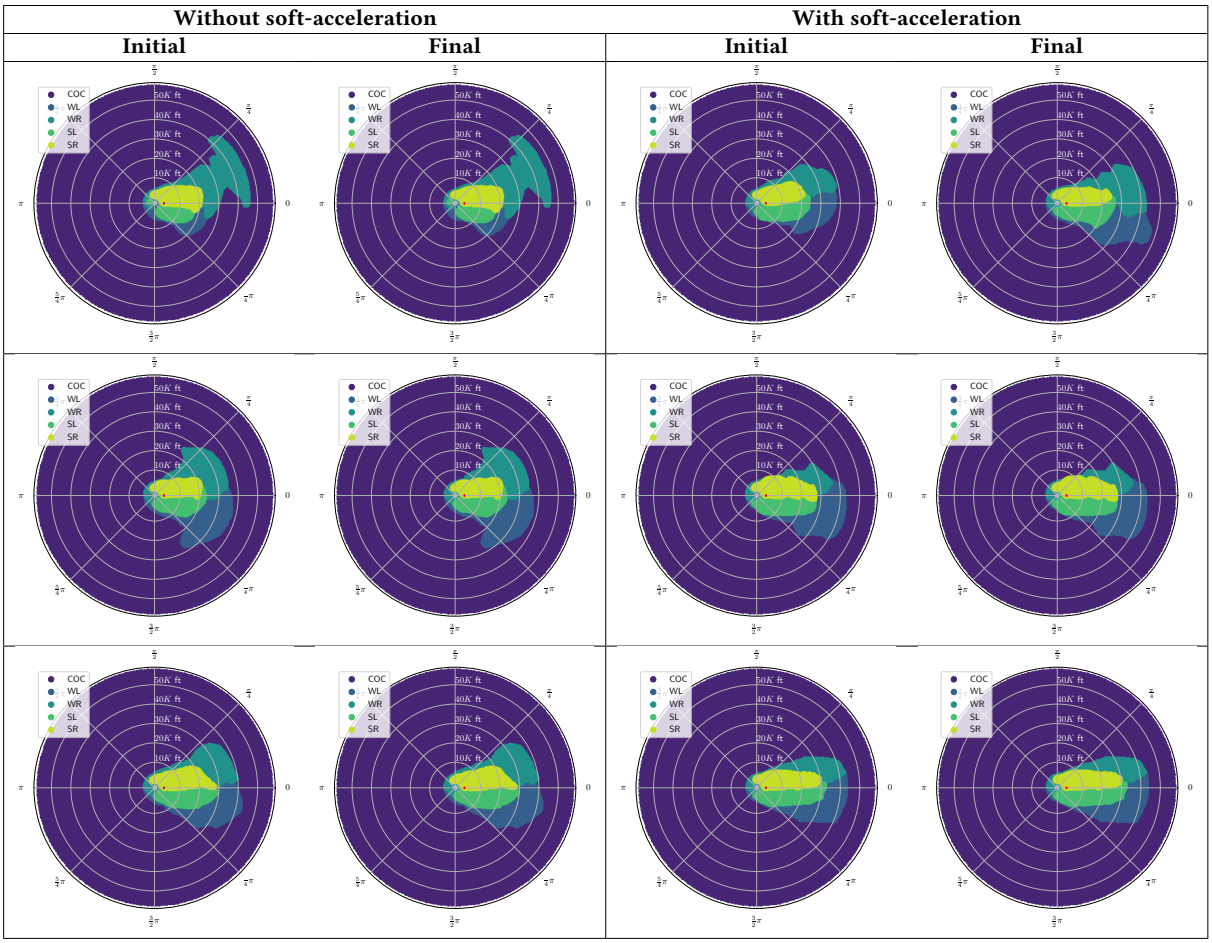


Table 14. HCAS certified policy synthesis for P_4. The networks are those for $a_{prev} = \mathbf{COC}$, $\tau = 0$. In all inputs we fix $\psi = 0$.

]

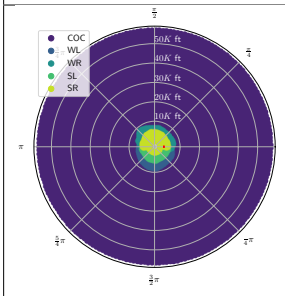
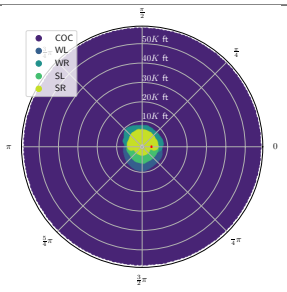
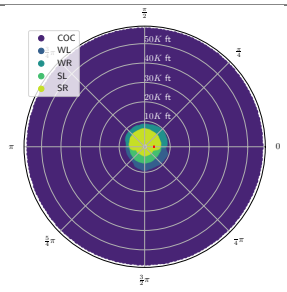
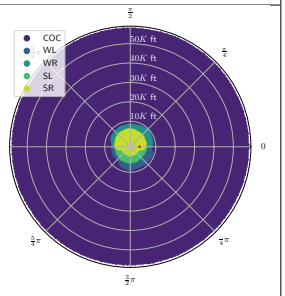
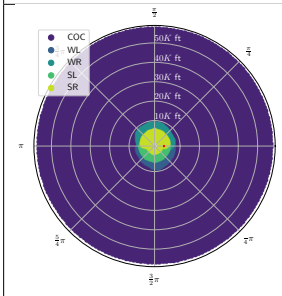
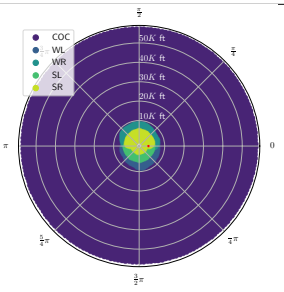
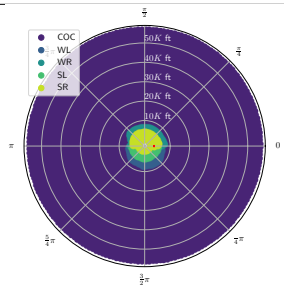
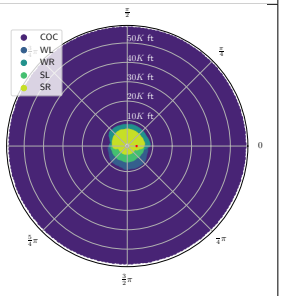
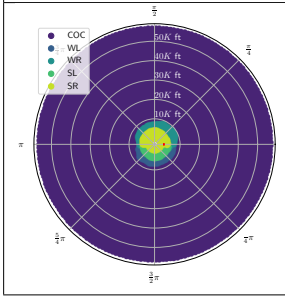
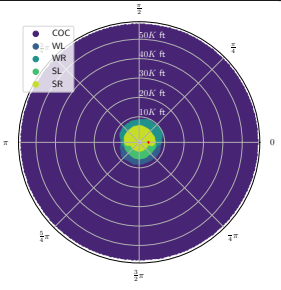
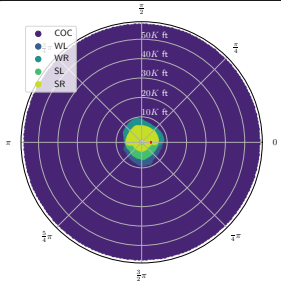
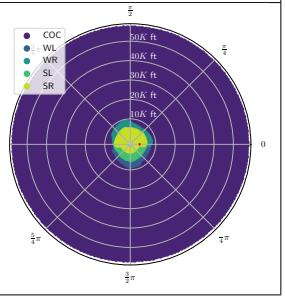
Without soft-acceleration		With soft-acceleration	
Initial	**Final**	**Initial**	**Final**
			
			
			

Table 15. HCAS certified policy synthesis for P_5. The networks are those for $a_{prev} = \mathbf{COC}$, $\tau = 0$. In all inputs we fix $\psi = -\pi$.

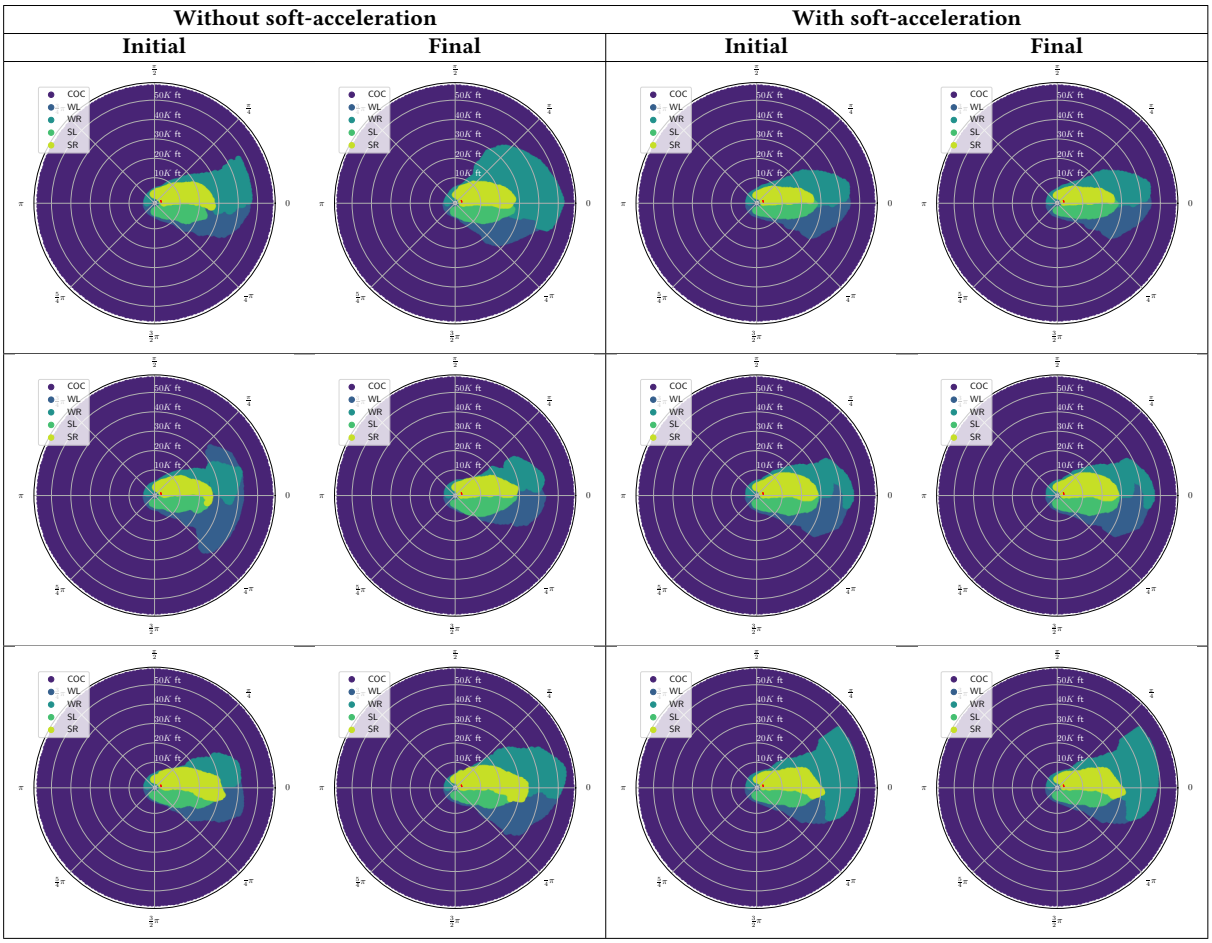


Table 16. HCAS certified policy synthesis for P_6. The networks are those for $a_{prev} = \mathbf{COC}$, $\tau = 0$. In all inputs we fix $\psi = -\pi$.

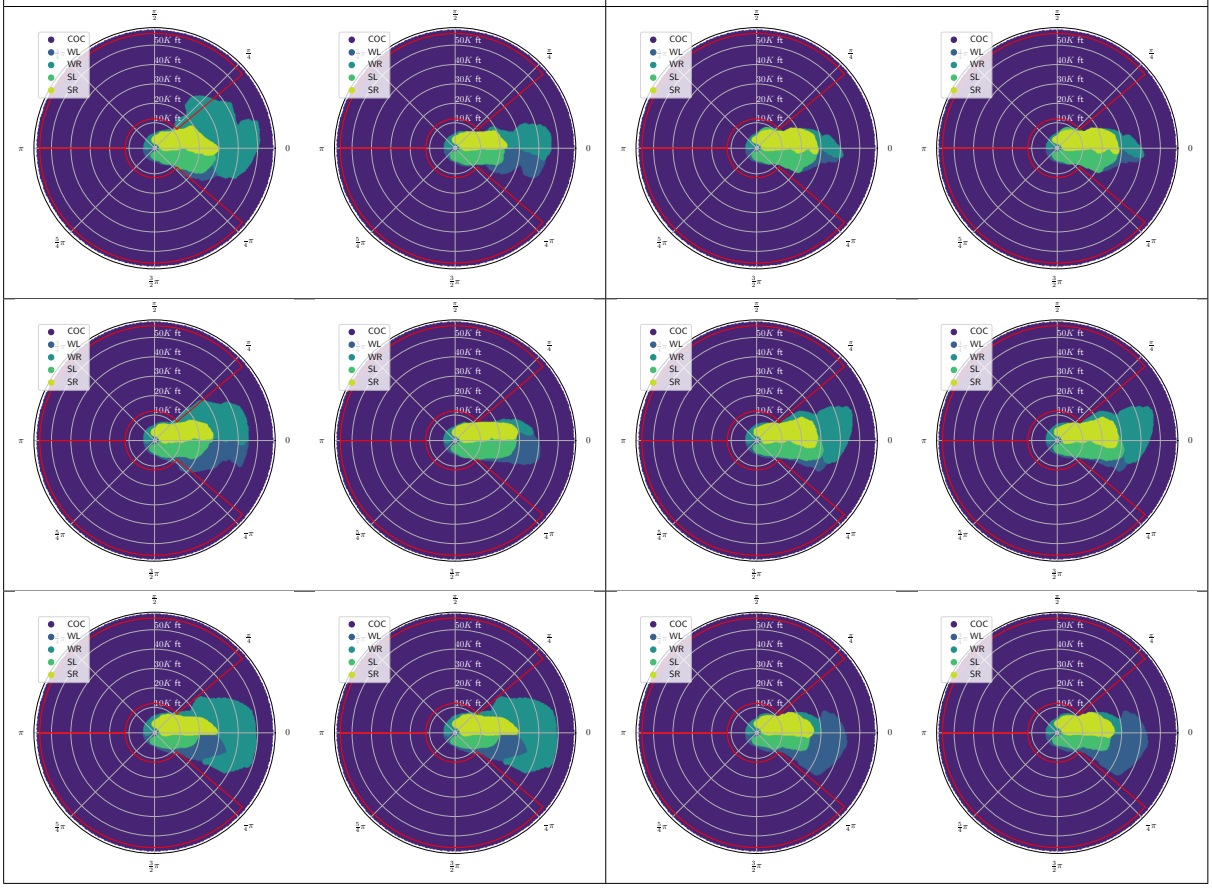
Without soft-acceleration		**With soft-acceleration**	
Initial	**Final**	**Initial**	**Final**
			

Table 17. HCAS certified policy synthesis for P_7. The networks are those for $a_{prev} = \mathbf{COC}$, $\tau = 60$. In all inputs we fix $\psi = -\pi$.

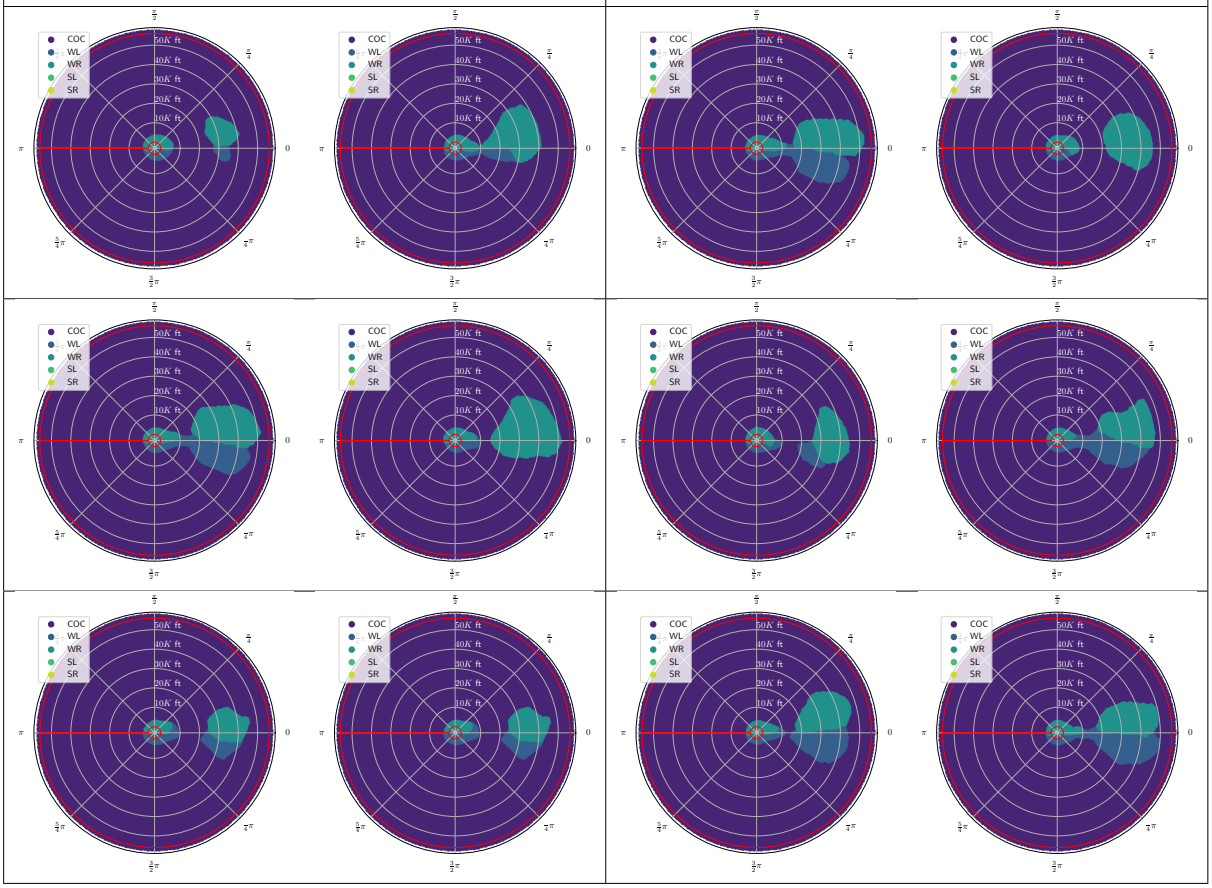
Without soft-acceleration		**With soft-acceleration**	
Initial	**Final**	**Initial**	**Final**
			

Table 18. HCAS certified policy synthesis for P_8. The networks are those for $a_{prev} = \mathbf{WL}$, $\tau = 60$. In all inputs we fix $\psi = -0.1$.

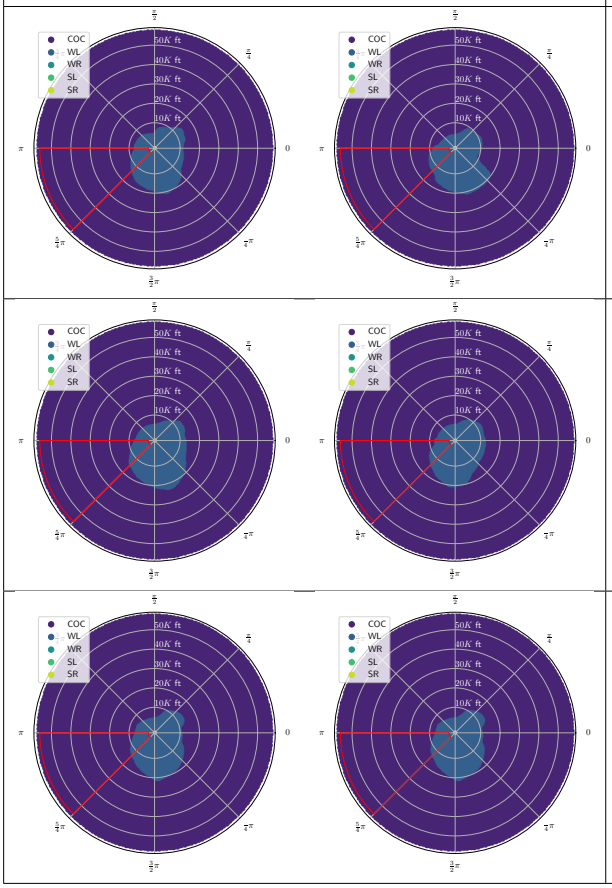
Without soft-acceleration		With soft-acceleration	
Initial	**Final**	**Initial**	**Final**
			

Table 19. HCAS certified policy synthesis for P_9. The networks are those for $a_{prev} = \mathbf{WR}$, $\tau = 0$. In all inputs we fix $\psi = -\pi$.

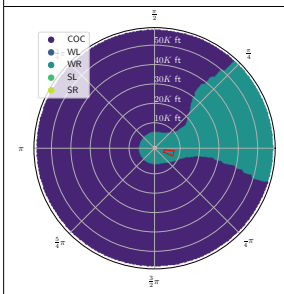
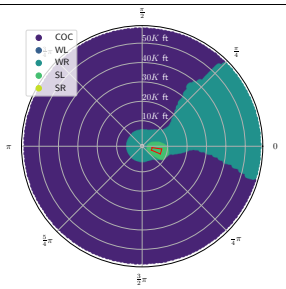
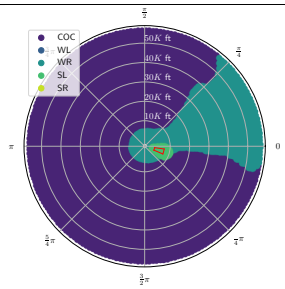
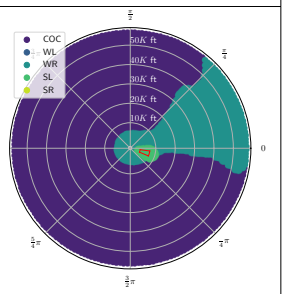
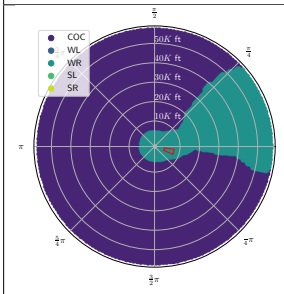
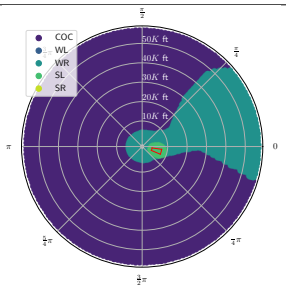
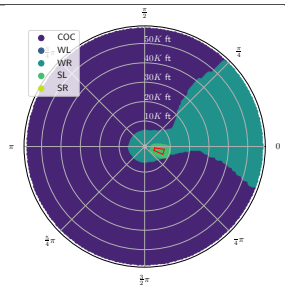
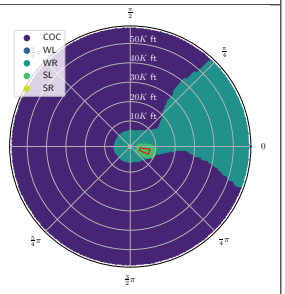
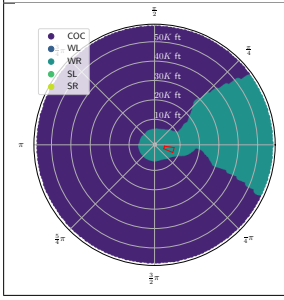
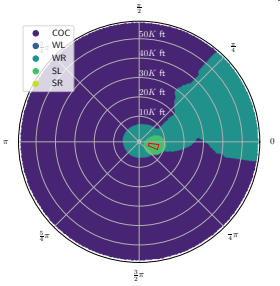
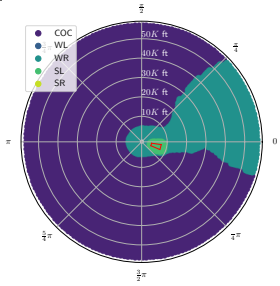
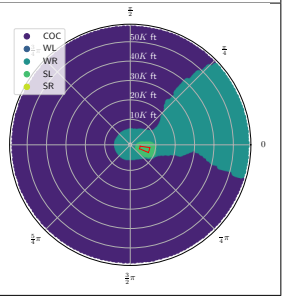
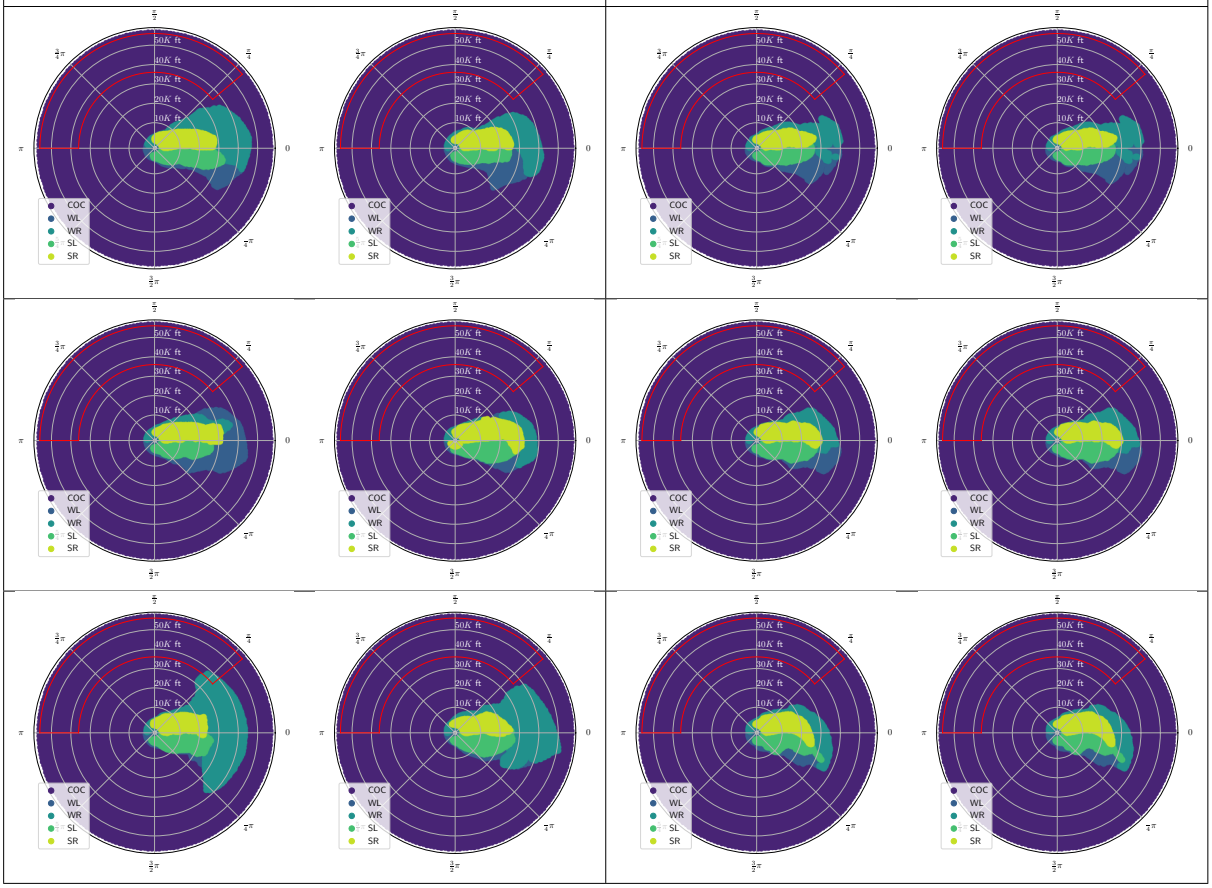
Without soft-acceleration		**With soft-acceleration**	
Initial	**Final**	**Initial**	**Final**
			
			
			

Table 20. HCAS certified policy synthesis for P_{10}. The networks are those for $a_{prev} = \mathbf{COC}$, $\tau = 0$. In all inputs we fix $\psi = -\pi$.

Without soft-acceleration		**With soft-acceleration**	
Initial	**Final**	**Initial**	**Final**
			

Received 07 May 2026; accepted 08 August 2026.