Verification of LTL properties on Neural Networks for Chemical Process Monitoring

Julien Girard-Satabin[†], Simon Lutz[⋆,∗], and Daniel Neider[⋆,∗]

†: Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France,
julien.girard2@cea.fr
⋆: TU Dortmund University, Germany, first.last@tu-dortmund.de
∗: Center for Trustworthy Data Science and Security, UA Ruhr, Germany

Abstract. The specification and verification of temporal properties is crucial to assess the safety of monitoring systems. Said systems need to ensure that certain properties are validated during its full operation time. The rich pattern-finding capabilities of deep learning are a strong incitation to implement such monitors as neural networks. However, neural networks verification historically focused on properties related to a single point in time. In this paper, we present a way to encode Linear Temporal Logic (LTL) properties in a neural network specification language. We derive from this encoding an automated translation to existing off-the-shelf provers. We apply our system to the verification of an industrial use case: a monitoring system for batch distillation. We were able to successfully specify and verify most of the properties in a small runtime.

1 Introduction

Machine learning has achieved remarkable success in numerous fields, which has led to its adoption in safety-critical applications such as autonomous driving[19], fraud detection[5] and in healthcare[33]. However, even when neural networks demonstrate exceptional performance they are error-prone[16]. This is particularly problematic in safety-critical domains, where an error may result in imminent hazards to the environment, financial loss, or the harm of human life. The desire to utilize the capabilities of machine learning in safety-critical domains has therefore given rise to a growing demand for methods that ensure the safety and reliability of neural networks. Inspired by methods from traditional software verification, this led to the development of a variety of verification techniques for neural networks in recent years. Given a neural network, these methods aim to autonomously verify (i.e., to mathematically prove) that the network fulfills a set of desired correctness properties.
In the last years, most of the research in the field of neural network verification has focused on the development of efficient and scalable verification tools. This has led to an exponential improvement in the scale of networks that can be verified within a reasonable time frame (e.g., under an hour) — from networks with just a few hundred thousand parameters to models with up to 70 million parameters[7]. In the context of

neural network verification, the most prominent and widely studied correctness property is adversarial robustness[16], which requires that small perturbations to a network's input must not cause significant changes in its output. Although robustness is an essential property that any neural network should satisfy, it is insufficient to ensure safety in many complex, safety-critical domains where neural networks must also adhere to a well-defined set of safety and regulatory requirements. However, current specification languages often focus on a single point in time and thus are insufficient for capturing the temporal behavior inherent in many real-world applications and their safety requirements.
In this paper, we overcome this gap by presenting a framework to incorporate temporal properties in a neural network specification language. More precisely, we extent the CAISAR[1] specification language to allow the definition of properties in a temporal specification language combining *Linear Temporal Logic* (LTL)[26] and *Linear Real Arithmetic* (LRA). For these properties, we derive an automated translation to the VNN-Lib standard[12] which allows us to verify them using existing off-the-shelf verification tools. We demonstrate the effectiveness of our approach in the context of an industrial use case: neural anomaly detectors for the chemical process of batch distillation.

2 Preliminaries

2.1 Linear Temporal Logic

Temporal properties can be expressed with *Linear Temporal Logic* (LTL)[26]: a logic that extends propositional logic with a set of temporal operators. We start by introducing the syntax of LTL. Let Var be a finite, nonempty set of propositional variables. Then

- each variable $x \in \text{Var}$ is an LTL formula;
- if Φ and Ψ are formulas, so are $\neg\Phi$, $\Phi \vee \Psi$, $\mathbf{X}\Phi$ (next), and $\Phi\mathbf{U}\Psi$ (until)

We also add the formulas *True*, *False*, $\Phi \wedge \Psi$, $\Phi \rightarrow \Psi$, and $\Phi \leftrightarrow \Psi$ as syntactic sugar. Furthermore, we can also add the temporal operators $\mathbf{F}\Phi$ (finally) and $\mathbf{G}\Phi$ (globally).
Formulas in linear temporal logic are interpreted over (infinite) sequences $w \in \left(2^{\text{Var}}\right)^{\omega}$ of sets over the variables. Intuitively, each position of the sequence describes the set of variables which are True at this point in time.
The semantics of LTL is defined by the means of an *interpretation* $\mathcal{J} : (\Phi, w) \mapsto 0, 1$, a function mapping pairs of LTL formulas and infinite sequences to Boolean values. With Φ and Ψ being formulas in Linear Temporal Logic, this function is inductively defined by:

$$\begin{aligned}
\mathcal{J}(x, w) &= 1 \Leftrightarrow x \in w_0 \text{for} x \in \text{Var}, \\
\mathcal{J}(\neg\Phi, w) &= 1 - \mathcal{J}(\Phi, w), \\
\mathcal{J}(\Phi \vee \Psi, w) &= \mathcal{J}(\Phi, w) \text{or} \mathcal{J}(\Psi, w), \\
\mathcal{J}(\mathbf{X}\Phi, w) &= \mathcal{J}(\Phi, w[1, \infty)), \wedge \\
\mathcal{J}(\Phi\mathbf{U}\Psi, w) &= \max_{\{i \geq 0\}} \min \mathcal{J}(\Psi, w[i, \infty)), \min_{\{0 \leq j < i\}} \mathcal{J}(\Phi, w[j, \infty)),
\end{aligned}$$

where w_0 describes the first position of a sequence w and $w[i, \infty)$ describes the sequence w but starting at position i.
Following the syntax this definition can be also extended to the formulas *True*, *False*, $\Phi \wedge \Psi$, $\Phi \rightarrow \Psi$, $\mathbf{F}\Phi := \text{True}\mathbf{U}\Phi$, and $\mathbf{G}\Phi := \neg\mathbf{F}(\neg\Phi)$.
Intuitively, the temporal formula $\mathbf{X}\Phi$ encodes that the formula Φ needs to hold at the next position in time. Furthermore, the formula $\Phi\mathbf{U}\Psi$ encodes that at some point in the future the formula Ψ needs to hold and on every point until then the formula Φ holds. Moreover, the formulas $\mathbf{F}\Phi$ and $\mathbf{G}\Phi$ indicate that Φ needs to hold at some or all points in time, respectively.
We say w *satisfies* Φ if $\mathcal{J}(\Phi, w) = 1$ and call $\mathcal{J}(\Phi, w)$ the *interpretation of* Φ *under* w.

2.2 Neural Network Verification

Neural networks (NNs) are programs $f : \mathcal{X} \mapsto \mathcal{Y}$ where $\mathcal{X} \in \mathbb{R}^i$ (resp. $\mathcal{Y} \in \mathbb{R}^o$) is an input space (resp. output space). We denote by X_i the $i-$th coordinate of a vector In this paper, we consider neural networks after training (all parameters are fixed and do not change). NNs can be represented as ONNX [9] operations, a community-driven standard. In ONNX, NNs are represented as Directed Acyclic Graphs (DAGs) $G = (E, V)$ that describe the NN data flow. Vertices $v \in V$ are computation nodes that takes an input and compute a parametrized operation that produces an output. Inputs and outputs are represented as multidimensional arrays with real values; they are often called *tensors*. There exist 196 operators in the ONNX standard; we informally introduce here some operators used in this paper:

- `MatMul(X,A)`, the matrix multiplication $Y = AX$ where $X \in \mathbb{R}^d$, $Y = \mathbb{R}^p$, $A = \mathbb{R}^{p \times d}$
- `Add(X,b)`, the element-wise addition on a vector X; $Y_i = X_i + b$
- `ReLU(X)`, the rectified linear unit $Y_i = \max(0, X_i)$

The inclusion of NNs in real-world industrial processes sparked interest on formally specifying and verifying NNs. After the seminal work of [28] and [20], the community produced various tools such as Marabou[32], $\alpha - \beta -$ CROWN[31], PyRAT[22] and nnenum[3]. The field structured around the International Verification of Neural Networks (VNN-COMP) [7], which proposes various benchmarks for the community to compete on.

2.3 The CAISAR platform

Historically, most tools focused on solving a class of problems known as local robustness[8]: given a small perturbation on the input, the output should not change. Hence, specification languages for provers were mostly designed to accommodate those properties. VNN-Lib, the *de facto* standard, is a fragment of the SMTLIB [4] Quantifier-Free Linear Real Arithmetic (QF_LRA). It has been shown that such specification language is lacking expressiveness [10] [1] to express more complex properties (for instance, properties related to multiple execution traces [6] or global robustness [2]).

CAISAR [1] is an attempt (among others, like Vehicle [11]) to circumvent this expressiveness bottleneck. CAISAR provides both a higher-level specification language, inspired by WhyML [13], and a compiler that rewrites a WhyML specification into a set of VNN-Lib queries. Such queries can then be sent to all VNN-Lib compliant provers. As to ease the verification process, CAISAR can automatically call provers, collect their results and check the satisfiability of the original WhyML specification. CAISAR supports 9 provers, among which are the top contenders of the VNN-COMP.
CAISAR language and compilation process are described in details in [1]. Following is a short summary of features relevant to this contribution:

- strongly typed language *a la* OCaml;
- built-in types and functions for vectors and neural networks computations;
- integration of a subset of WhyML specification directly within the neural network.

3 Formalization of the specification language extension

We rely on the existing specification language of CAISAR, defined comprehensively in [1], which provides a typed first-order logic with pattern-matching. In this section, we will briefly summarize our extension to incorporate the temporal operators of Linear Temporal logic. Similar to the work by Geatti et al. [14], the specification language of our extension is structured around the idea of a *term*. Let Var be a set of real-valued variables, $c \in \mathbb{R}$ be a real-valued constant, and t_1, t_2 be two terms. Then

$$t := x \in \text{Var} | c | c * t_1 | t_1 + t_2 | \ominus x$$

is a term in our specification language. Compared to Linear Real Arithmetic, the new addition is the *dash.ox* operator. Based on this definition, we define a formula in our specification language as follows: Let t_1, t_2 be terms and φ_1, φ_2 be formulas. Then

$$\varphi := t_1 < t_2 | \neg\varphi_1 | \varphi_1 \vee \varphi_2 | \mathbf{X}\varphi | \varphi_1 \mathbf{U} \varphi_2$$

is a formula in our specification language. We also add the term $t_1 - t_2$ and the formulas *True*, *False*, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$, $\mathbf{F}\varphi_1$, $\mathbf{G}\varphi_1$, and $t_1 \circ t_2$ with $\circ \in \{\leq, >, \geq, =\}$ as syntactic sugar.
After defining the syntax, we will define the semantics of our specification language next. Similar to Linear Temporal Logic, our specifications are interpreted over sequences, but over finite sequences of sets of real values. Intuitively, one of these sets represents the sensor values at a specific point in time. The length of the sequence is denoted by T. The semantics of our specification language is defined by means of an interpretation $\mathcal{J}_s$. It can be seen as a specialization of the previously defined interpretation function $\mathcal{J}$. In the context of our specification language, an interpretation is a function $\mathcal{J}_s : (\varphi, m, t) \mapsto 0, 1$ mapping sets of formulas φ, multivariate input sequences (represented as a matrix m), and time points t to

Boolean values. This function is inductively defined following the usual definitions for Boolean connectives, and arithmetic functions and relations. The function $dash.ox$ will refer to the value of a variable at the following position of a sequence, i.e., $\mathcal{J}_s\left(dash.ox, m, t\right) = m\left[t+1, i_x\right]$, where i_x refers to the column containing the values of the variable x.
We define now the semantics of the temporal operators $\mathbf{X}$ and $\mathbf{U}$, as well as $\mathbf{F}$ and $\mathbf{G}$.

3.1 Next

The Next operator (noted $\mathbf{X}\varphi$) checks that the formula φ is true at the next time-step. Evaluating the Next operator at the last position of a sequence will return *False*. Formally, we define

$$\mathcal{J}_s\left(\mathbf{X}\varphi, m, t\right) \equiv t+1 \in [1..T] \wedge \mathcal{J}_s\left(\varphi, m, t+1\right)$$

3.2 Until

Given two the formula φ_1 and φ_2, the Until operator (noted $\varphi_1 \mathbf{U} \varphi_2$) checks that there exist at least one time-step such that φ_2 holds and that until then, φ_1 holds. Formally, we define

$$\mathcal{J}_s\left(\varphi_1 \mathbf{U} \varphi_2, m, t\right) \equiv \exists j \in [t..T]\,.\mathcal{J}_s\left(\varphi_2, m, j\right) \wedge \forall k \in [t..j-1]\,.\mathcal{J}_s\left(\varphi_1, m, k\right)$$

3.3 Finally

The Finally operator (noted $\mathbf{F}\varphi$) checks that the formula φ is valid at a time-step t or for at least one time-step after t. Formally, we define

$$\mathcal{J}_s\left(\mathbf{F}\varphi, m, t\right) \equiv \exists j \in [t..T]\,.\mathcal{J}_s\left(\varphi, m, j\right)$$

3.4 Globally

The Globally operator (noted $\mathbf{G}\varphi$) checks that the formula φ is valid for all time-steps after t (including t). Formally, we define

$$\mathcal{J}_s\left(\mathbf{G}\varphi, m, t\right) \equiv \forall k \in [t..T]\,.\mathcal{J}_s\left(\varphi, m, k\right)$$

3.5 Formalization in CAISAR

Given an input vector $X \in \mathbb{R}^f$ and a finite number of time-steps T, it is possible to encode the full sequence as a matrix $m \in \mathbb{R}^{T \times f}$. Each row represents a time-step, each column being an input feature. The application of an operator for a formula consists then on applying the interpretation function $\mathcal{J}_s$ on the proper matrix coordinates. See 1 for an illustrative figure.
This formalization can be expressed in the CAISAR specification language [1] in two steps. First, as CAISAR lacks a way to represent matrices, we add this as a `Matrix` theory. In CAISAR terminology, a theory

x_0^0	x_1^0	$\dots$	x_f^0	$\leftarrow$ time value
x_0^1	x_1^1	$\dots$	x_f^1	
$\vdots$	$\vdots$	$\ddots$	$\vdots$	
x_0^t	x_1^t	$\dots$	x_f^t	
$\uparrow$ feature value				

Fig. 1: Representation of a time-series input as coordinates of matrix m. Rows describe a given time-step t_i, column represent a feature value x_i. Coordinate $m\,(i, j)$ reads as value of feature j at time-step i.

```
1  theory Matrix
2    use Vector
3    use int.Int
4
5    (* Flat representation. [r] is the number of rows, [c] the number of columns *)
6    type t α = {d: vector α; r: int; c: int}
7
8    (* True if the matrix has [r] rows and [c] columns *)
9    predicate has_shape (m: t α) (r: int) (c: int) =
10   m.r = r ∧ m.c = c
11
12   (* True if the provided index [t] is between 0 and the number of rows *)
13   predicate valid_row (m: t α) (t: int)  =
14   0 ≤ t ∧  t < m.r
15
16   (* True if the provided index [f] is between 0 and the number of columns *)
17   predicate valid_column (m: t α) (f: int)  =
18   0 ≤ f ∧  f < m.c
19
20   predicate valid_coordinate (m: t α) (t: int) (f: int) =
21   valid_row m t ∧ valid_column m f
22
23   (* [get m t f] returns the value located at row [t] and column [f] in matrix [m] *)
24   function get (m: t α) (t: int) (f: int) : α =
25   let coord = (t * m.c) + f in
26   m.d[coord]
27
28 end
```

Fig. 2: A minimal Matrix theory in CAISAR.

is akin to a library defining symbols, predicates, lemmas and functions to be reused in other contexts. The `Matrix` theory derives from the existing `Vector` theory. Likewise, it supports function polymorphism on real and floating point values. See 2 for an extract of the `Matrix` theory specification.
Second, we define the LTL operators introduced earlier using this Matrix theory. The definitions are available in 3.

```
function global_m
(p: Matrix.t α→ int → bool)
(m: Matrix.t α)
(t: int)
: bool =
  valid_row m t ∧ forall j: index. t ≤ j < m.r → p m t

function finally_m
(p: Matrix.t α → int → bool)
(m: Matrix.t α)
(t: int)
: bool =
  valid_row m t ∧ (exists j: index. t ≤ j < m.r ∧ p m j)

function next_m
(p: Matrix.t α → int → bool)
(m: Matrix.t α)
(t: int)
: bool =
  valid_row m t ∧ (valid_row m (t + 1) ∧ p m (t + 1))

function until_m
(p: Matrix.t α → int → bool)
(q: Matrix.t α → int → bool)
(m: Matrix.t α)
(t: int)
: bool =
  valid_row m i ∧ (
    exists j: index. t ≤ j < m.r ∧ q m j ∧
    forall k: index. (t ≤ k < j) → p m k
  )
```

Fig. 3: Formalization of LTL operators in CAISAR. Value `t` represents a time-step.

3.6 Automated rewrite via CAISAR

Since the shape of matrices is finite and know statically, it is possible to derive a formula directly in VNN-Lib to enable verification. All existential quantifications over matrix components are straightforwardly interpreted as a quantification over each individual element. Thus, a formula in our language can be translated to an equisatisfiable of a first-order logic formula with linear arithmetic operations (akin to `QF_LRA` in SMTLIB.)
Existing limitations from CAISAR apply; namely, formulas with alternating quantifiers are not supported. There is no support for broadcasting or shape inference for matrices. Finally, if it is possible to write a

specification with operations between matrices, they will not be translated directly as VNN-Lib. This would require additional modifications of CAISAR internals, which was out-of-scope for this work.

4 Use-case presentation

4.1 Batch distillation

In this paper, we demonstrate the effectiveness of our approach in the context of anomaly detection in a chemical process called batch distillation. Distillation is the most widely used separation process in the chemical industry with applications, for instance, in petroleum refinement or the pharmaceutical industry. Simplified, it describes the processes of separating a liquid mixture into multiple, possibly unknown chemical components. In a so-called distillation column the mixture is heated up until it starts to vaporize. The resulting vapor rises up through the rectifying column, until it is condensated in the condenser at the top of the column. Batch distillation processes often run only for a certain time frame in a lab-scale distillation plant and are in general used for small-scale production units with high product-quality requirements.
Detecting anomalies in chemical precesses is a crucial task to ensure the safety of chemical plants and their operators. When an anomaly is missed, this can cause severe environmental damage or endanger human lives as unexpected behavior can lead to fatal incidents such as the explosion of a plant. When developing machine learning solutions to assist human operators by automatically detecting anomalies in a process, it needs to be ensured that these neural anomaly detectors are safe and reliable.

4.2 Implementation as a Neural Network

In this section, we introduce the neural network we trained to detect anomalies in a batch distillation process. The network was trained on a subset of the publically available NoBOOM dataset[29] that contains both fault-free and anomalous data from real operating distillation plants[1].
Our work does not aim at state-of-the-art performance and rather aim to demonstrate the feasibility of verification. As such, to keep training time low, we focus on only a single operating point of the ternary 1-butanol, 2-propanol, and water system.
In order to ensure that the network can be verified by many available verification tools, we rely on a simple, fully connected network with ReLU activation function instead of widely used anomaly detectors such as long short term memory networks (LSTMs)[24] or generative adversarial networks[15]. The network expects a multivariate time-series of fixed length as input and predicts whether it is anomalous or not. For our

[1] The full dataset is available https://www.kaggle.com/datasets/faebs94/noboom-anomaly-detection-in-chemical-processes.

experiments, we restrict the network to four input features and time-series of length five. These features are the level of liquid in the so-called sump at the bottom of the distillation column, two temperature sensors at the bottom and top of the distillation column, and the pressure in the column. The input layer is followed by two hidden layers of size 128 and an output layer of size 2. The first output y_0 describes the score for the input to be a normal operation while the second output y_1 describes the score for anomaly. Hence, if $y_0 < y_1$, then our network detects an anomaly.

4.3 Properties

Lutz et al.[23] proposed an extensive set of correctness properties for anomaly detection in distillation processes. These properties describe necessary conditions for the formal correctness of neural anomaly detectors in the form of chemical and physical laws that must hold during normal operation of a chemical plant. Whenever such a law is violated, an anomaly must be detected. Formally, these correctness properties are given by formulas

$$\neg \mathbf{G}\psi \rightarrow f(x) = 1 \tag{1}$$

$$\equiv \mathbf{F}(\neg\psi) \rightarrow f(x) = 1, \tag{2}$$

where ψ represents a formula describing a chemical or physical law and $f(x) = 1$ indicates that the neural network f predicts an anomaly.
In the following, we describe the set of properties we consider in our evaluation. They present a subset of properties introduced by Lutz et al. restricted to our use case of batch distillation and the features used in the training of our neural network. All properties assume time series containing the sensor readings of 4 features for 5 time-steps as input and a neural network $f : \mathbb{R}^{20} \mapsto \mathbb{R}^2$ described as above. Note that in the implementation of the properties within CAISAR the thresholds below are normalized following the same normalization process used during the training of the network.

Property φ_1 The first property describes the physical law that the pressure in the plant can never be zero or below. Therefore, whenever the input to the neural network contains a value below or equal to zero, an anomaly should be predicted, as indicated by the output y_1 being larger than y_0. Formally, this can be expressed as

$$\varphi_1 \equiv \mathbf{F}(x_{\text{pressure}} \leq 0) \rightarrow y_0 < y_1, \tag{3}$$

where x_{pressure} refers to the sensor readings of the pressure sensor. 4 displays how this property is implemented in CAISAR.

Property φ_2 Similar to the first property, the values of the pressure sensor are also upper bounded during normal operation. In particular,

```
goal phi_1:
let rows = 5 in
let columns = 4 in
let pos_pres = 3 in
(* Normalized value, unnormalized value is 0.0 *)
let pressure_threshold = (-1.648166501486620028060769982403144240379333496093 75:Float64.t) in

forall v: vector Float64.t. has_length v (rows * columns) →
let m = {d = v; r = rows; c = columns} in
matrix_valid_input_various_bounds m lbounds ubounds →
let out = (nn @@ m.d) in
  final_m (matrix_column_bound_upper pos_pres pressure_threshold) m 0
  → out[0] .≤ out[1]
```

Fig. 4: Encoding of property φ_1 (defined in 4.3) in CAISAR

any sensor readings above 1013.25 mbar are considered abnormal. Formally, this can be expressed as

$$\varphi_2 \equiv \mathbf{F}\left(x_{\text{pressure}} > 1013.25\right) \rightarrow y_0 < y_1.$$

Properties $\boldsymbol{\varphi_3}$ and $\boldsymbol{\varphi_4}$ Our network contains two input features that correspond to temperature sensors in the distillation plant. During operation of the plant the temperature in the plant should be kept in between the normal operation conditions. Any deviation, either to high or to low, indicates an anomaly. For the two temperature sensors $x_{\text{temperature}_1}$ and $x_{\text{temperature}_2}$ this can be formalized as

$$\varphi_3 \equiv \mathbf{F}\left(x_{\text{temperature}_1} < 0 \vee x_{\text{temperature}_1} > 600\right) \rightarrow y_0 < y_1, \text{and}$$

$$\varphi_4 \equiv \mathbf{F}\left(x_{\text{temperature}_2} < 0 \vee x_{\text{temperature}_2} > 600\right) \rightarrow y_0 < y_1.$$

Property $\boldsymbol{\varphi_5}$ The two temperature sensors are placed along the distillation column such that $x_{\text{temperature}_1}$ is placed at the bottom of the column and $x_{\text{temperature}_2}$ is placed at the top near the condenser. With a heating element near the bottom of the distillation column and the uprising vapor being cooled down at the condenser, a simple physical law states that the temperature must decrease from bottom to the top of the distillation column. Formally, this is expressed by the formula

$$\varphi_5 \equiv \mathbf{F}\left(x_{\text{temperature}_1} < x_{\text{temperature}_2}\right) \rightarrow y_0 < y_1.$$

Property $\boldsymbol{\varphi_6}$ At the bottom of the distillation column is the so-called sump, a larger vessel that is connected to a heating element and in which the liquid mixture is filled in the beginning of the distillation

process. The level of the mixture is measured by a binary sensor x_{sump}, where 1 indicates that there is still liquid in the vessel and 0 indicates an empty sump. To ensure that the temperature in the plant will not rapidly increase causing it to melt or explode, the sump must always contain some amount of liquid mixture. This is formalized as

$$\varphi_6 \equiv \mathbf{F}\,(x_{\text{sump}} = 0) \rightarrow y_0 < y_1.$$

5 Related works

5.1 Temporal properties for neural network

Our contribution can be related to the verification of reactive systems with neural networks components. There already exist a heavy corpus of work regarding verification of reactive systems [27], with various specification languages such as Lustre [17]. Our extension of CAISAR is much more modest, as it does not aim to reason on infinite sequences. Developing our extension instead of relying on Lustre was motivated by CAISAR ease of installation and the presence of primitives for machine learning specifications which, to the best of our knowledge, are not directly available in Lustre. The overall small-scale of both specifications and inputs allowed us to focus on expressiveness. Furthermore, we are not interested into generating counter-examples, which is one of the interests of synchronous model-checking. There exist approaches where LTL specifications are mined using graph neural networks [25], or LTL policies are synthetised through reinforcement learning [30] [21]. Here, the specification is known beforehand, which only requires its efficient encoding as input and output constraints - which CAISAR provides.
Our approach can be related to [18]. In this work, authors provide elaborate bound propagation algorithms and Mixed-Integer Linear Programming (MILP) for the verification of synchronous systems with reinforcement learning agents, accounting for uncertainties in the system. Our narrower scope (bounded-time LTL, no uncertainty) does not require us to use dedicated algorithms. Furthermore, we do not wish to develop new proving techniques as they do, but to reuse state-of-the-art provers.

6 Experimental results

Experiments were conducted with CAISAR 5.0 available through the Opam package manager. They were conducted on a laptop Precision 3591, with a Intel Core Ultra 9 185H CPU, 64GB of RAM and running NixOS 26.05. Solver Marabou 2.0 [32] was used for all experiments; other provers versions were those provided by CAISAR. We launched the verification for all properties at once in a theory containing the `Matrix` and operators definitions, as well as the properties defined in 4.3. The specifications were using `IEEE-Float` values (as CAISAR 5.0 only supports specifications written with this type). See 1 for the detailed results.

Marabou[32]	PyRAT[22]	$\alpha - \beta -$ CROWN[31]	nnenum[3]
6.989s	🕐	65.34s	54.25s

Table 1: Results of the experimental study. Timeout was set at 120s for all provers. 🕐 denotes a timeout. All provers returned `Invalid` when they did not timed out.

7 Conclusion and perspectives

We presented a formalization of a bounded-LTL specification language applied to real values. This specification language was implemented as an extension within the CAISAR platform for the verification of AI systems. Its automated compilation to state-of-the-art solvers allowed us to formally verify properties on a batch-distillation industrial system.
We note that scalability is an open question. Indeed, the best-performing solver is Marabou, which is not best-ranking at the VNN-COMP. We note that $\alpha - \beta -$ CROWN, PyRAT and nnenum, while performing well on the competition (respectively first, second and fourth in the 2025 edition), either timeout or take much longer to verify the properties. It may either show the limitation of our approach and require dedicated tooling for reactive systems specification and verification.

References

1. Alberti, M., Bobot, F., Chihani, Z., Grastien, A., Girard-Satabin, J.: The CAISAR platform: extending the reach of Machine Learning specification and verification. In: International Conference on Integrated Formal Methods (IFM). pp. 290–309 (2025)
2. Athavale, A., Bartocci, E., Christakis, M., Maffei, M., Nickovic, D., Weissenbacher, G.: Verifying global two-safety properties in neural networks with confidence. In: International Conference on Computer Aided Verification (CAV). pp. 329–351 (2024)
3. Bak, S.: nnenum: verification of ReLU neural networks with optimized abstraction refinement. In: NASA Formal Methods Symposium (NFM). pp. 19–36
4. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability Modulo Theories. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, chap. 33, pp. 825–885. IOS Press (Feb 2021), http://theory.stanford.edu/~barrett/pubs/BSST21.pdf
5. Bin Sulaiman, R., Schetinin, V., Sant, P.: Review of machine learning approach on credit card fraud detection. Human-Centric Intelligent Systems **2**(1), 55–68 (2022)
6. Boetius, D., Leue, S.: Verifying Global Neural Network Specifications using Hyperproperties (2023)
7. Brix, C., Bak, S., Johnson, T.T., Wu, H.: The fifth international verification of neural networks competition (VNN-COMP 2024): summary and results (2024)

8. Casadio, M., Komendantskaya, E., Daggitt, M.L., Kokke, W., Katz, G., Amir, G., Refaeli, I.: Neural network robustness as a verification property: A principled case study. In: International Conference on Computer Aided Verification. pp. 219–231. Springer (2022)
9. Community: ONNX: Open Neural Network eXchange Format standard for machine learning interoperability (2022)
10. Cordeiro, L.C., Daggitt, M.L., Girard-Satabin, J., Isac, O., Johnson, T.T., Katz, G., Komendantskaya, E., Lemesle, A., Manino, E., Šinkarovs, A., Wu, H.: Neural Network Verification is a Programming Language Challenge. In: Vafeiadis, V. (ed.) Programming Languages and Systems. pp. 206–235. Springer Nature Switzerland (2025). https://doi.org/10.1007/978-3-031-91118-7_9
11. Daggitt, M.L., Kokke, W., Atkey, R., Slusarz, N., Arnaboldi, L., Komendantskaya, E.: Vehicle: Bridging the Embedding Gap in the Verification of Neuro-Symbolic Programs (2024). https://doi.org/10.48550/ARXIV.2401.06379
12. Demarchi, S., Guidotti, D., Pulina, L., Tacchella, A.: Supporting standardization of neural networks verification with VNN-LIB and CoCoNet. In: Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS). pp. 47–58 (2023)
13. Filliâtre, J.C., Paskevich, A.: Why3 - Where Programs Meet Provers. In: Felleisen, M., Gardner, P. (eds.) Programming Languages and Systems. pp. 125–128. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37036-6_8
14. Geatti, L., Gianola, A., Gigante, N., et al.: Linear temporal logic modulo theories over finite traces. In: IJCAI. vol. 22, pp. 2641–2647 (2022)
15. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks **63**(11), 139–144 (oct 2020). https://doi.org/10.1145/3422622, https://doi.org/10.1145/3422622
16. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and Harnessing Adversarial Examples. International Conference on Learning Representations (2015)
17. Halbwachs, N., Caspi, P., Pilaud, D.: The synchronous dataflow programming language lustre. another look at real time programming, proceedings of the ieee (1991)
18. Hosseini, M., Lomuscio, A., Paoletti, N.: LTL Verification of Memoryful Neural Agents. https://doi.org/10.48550/arXiv.2503.02512, http://arxiv.org/abs/2503.02512
19. Huang, Y., Chen, Y.: Autonomous driving with deep learning: A survey of state-of-art technologies (2020), https://arxiv.org/abs/2006.06091
20. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In: Computer Aided Verification, pp. 97–117. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-63387-9_5
21. Kuo, Y.L., Katz, B., Barbu, A.: Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas. In: 2020 IEEE/RSJ International Conference on Intelligent

Robots and Systems (IROS). pp. 5604–5610 (Oct 2020). https://doi.org/10.1109/IROS45743.2020.9341325, https://ieeexplore.ieee.org/abstract/document/9341325
22. Lemesle, A., Lehmann, J., Gall, T.L., Chihani, Z.: Neural Network Verification with PyRAT. In: International Static Analysis Symposium (SAS). pp. 11–33 (2025)
23. Lutz, S., Arweiler, J., Muraleedharan, A., Kahlhoff, N., Hartung, F., Jungjohann, I., Nagda, M., Reinhardt, D., Wagner, D., Werner, J., Will, J., Burger, J., Bortz, M., Hasse, H., Fellenz, S., Jirasek, F., Kloft, M., Leitte, H., Mandt, S., Reithermann, S., Schmid, J., Neider, D.: A benchmark suite for verifying neural anomaly detectors in distillation processes. In: Cerrato, M., Kalinauskaitė, D., Lukoševičius, M., Pechenizkiy, M., Šutienė, K. (eds.) Machine Learning and Principles and Practice of Knowledge Discovery in Databases. pp. 290–306. Springer Nature Switzerland, Cham (2026)
24. Malhotra, P., Vig, L., Shroff, G., Agarwal, P.: Long short term memory networks for anomaly detection in time series (04 2015)
25. Neider, D., Roy, R.: What is Formal Verification without Specifications? A Survey on mining LTL Specifications (Jan 2025). https://doi.org/10.48550/arXiv.2501.16274, http://arxiv.org/abs/2501.16274
26. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (Sfcs 1977). pp. 46–57. IEEE (1977). https://doi.org/10.1109/sfcs.1977.32
27. Pnueli, A.: Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. In: Current Trends in Concurrency: Overviews and Tutorials, pp. 510–584. Springer (2005)
28. Pulina, L., Tacchella, A.: NeVer: A tool for artificial neural networks verification. Annals of Mathematics and Artificial Intelligence **62**(3–4), 403–425 (Jul 2011). https://doi.org/10.1007/s10472-011-9243-0, https://doi.org/10.1007/s10472-011-9243-0
29. Wagner, D., Hartung, F., Arweiler, J., Muraleedharan, A., Jungjohann, I., Nair, A., Reithermann, S., Schulz, R., Bortz, M., Neider, D., Leitte, H., Pfeffinger, J., Mandt, S., Fellenz, S., Katz, T., Jirasek, F., Burger, J., Hasse, H., Kloft, M.: Noboom: Chemical process datasets for industrial anomaly detection. In: NeurIPS (2025), https://neurips.cc/virtual/2025/loc/san-diego/poster/121442
30. Wang, C., Li, Y., Smith, S.L., Liu, J.: Continuous Motion Planning with Temporal Logic Specifications using Deep Neural Networks (Sep 2020). https://doi.org/10.48550/arXiv.2004.02610, http://arxiv.org/abs/2004.02610
31. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-CROWN: efficient bound propagation with per-neuron split constraints for complete and incomplete neural network robustness verification. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 29909–29921 (2021)
32. Wu, H., Isac, O., Zeljić, A., Tagomori, T., Daggitt, M., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., Huang, P., Lahav, O., Wu, M., Zhang, M., Komendantskaya, E., Katz, G., Barrett, C.:

Marabou 2.0: a versatile formal analyzer of neural networks. In: International Conference on Computer Aided Verification (CAV). pp. 249–264 (2024)
33. Yuan, H., Yu, K., Xie, F., Liu, M., Sun, S.: Automated machine learning with interpretation: A systematic review of methodologies and applications in healthcare. Medicine Advances **2**(3), 205–237 (2024). https://doi.org/https://doi.org/10.1002/med4.75, https://onlinelibrary.wiley.com/doi/abs/10.1002/med4.75