ProbStar Temporal Logic for Verifying Complex Behaviors of Learning-enabled Systems

Hoang-Dung Tran
dtran30@unl.edu
University of Nebraska-Lincoln
Lincoln, Nebraska, USA

Sung Woo Choi
schoi9@huskers.unl.edu
University of Nebraska-Lincoln
Lincoln, Nebraska, USA

Yuntao Li
University of Nebraska-Lincoln
Lincoln, Nebraska, USA

Hideki Okamoto
Toyota NA R&D
Ann Arbor, Michigan, USA

Bardh Hoxha
Toyota NA R&D
Ann Arbor, Michigan, USA

Georgios Fainekos
Toyota NA R&D
Ann Arbor, Michigan, USA

ABSTRACT

This paper introduces a novel quantitative verification framework for analyzing the temporal behaviors of learning-enabled systems (LES). Our approach employs ProbStar Temporal Logic (ProbStarTL) to specify LES temporal behaviors alongside advanced reachability and verification algorithms. Unlike existing qualitative methods focusing primarily on reach-avoid properties, our framework enables quantitative analysis of temporal properties. ProbStarTL, distinct from Signal Temporal Logic, operates on sequences of timed probabilistic star reachable sets, known as ProbStar signals. It features a clear syntax and dual qualitative and quantitative semantics. Our framework includes depth-first search algorithms for generating ProbStar traces and novel verification algorithms that transform ProbStarTL specifications into a computable disjunctive normal form for analysis. Our verification algorithms allow for both exact and approximate analyses. The exact scheme guarantees sound and complete results with precise satisfaction probabilities, while the approximate scheme offers sound results with maximum and minimum satisfaction probabilities at a reduced computational cost. The new verification framework is implemented using StarV, and its effectiveness is demonstrated through case studies on a learning-based adaptive cruise control system and an advanced emergency braking system.

CCS CONCEPTS

• **Software and its engineering** → **Software verification**.

ACM Reference Format:
Hoang-Dung Tran, Sung Woo Choi, Yuntao Li, Hideki Okamoto, Bardh Hoxha, and Georgios Fainekos. 2025. ProbStar Temporal Logic for Verifying Complex Behaviors of Learning-enabled Systems. In *28th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '25), May 6–9, 2025, Irvine, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3716863.3718036

HSCC '25, May 6–9, 2025, Irvine, CA, USA

ACM ISBN 979-8-4007-1504-4/2025/05.
https://doi.org/10.1145/3716863.3718036

1 INTRODUCTION

The rapid advancement and integration of machine learning techniques into various domains present tremendous opportunities and at the same time significant challenges, especially in safety-critical applications. The increasing reliance on deep learning models, such as deep neural networks (DNNs), in these applications, underscores a pressing need to ensure these systems function as expected, even under environmental uncertainties or adversarial attacks [29]. Recent years have seen substantial efforts in the formal verification of learning-enabled systems (LES), focusing primarily on assessing the safety and robustness of both open-loop [24] and closed-loop systems [38], where neural networks control physical plant dynamics. Despite advancements in verification methods, a crucial gap remains in verifying spatio-temporal behaviors and properties of LES. This paper addresses this gap by proposing a novel verification framework centered on temporal properties.

We introduce Probstar Temporal Logic (ProbStarTL), a formalism enabling quantitative verification of temporal properties of LES. As we focus on verification involving the computation of reachable sets of LES, ProbStarTL is defined on a (bounded-time) ProbStar signal (or ProbStar trace), a sequence of discrete, timed probabilistic star reachable sets. The interpretation of ProbStarTL captures a symbolic representation of the set of LES traces that satisfy the specification. The logic supports two basic temporal operators: *always* ($\Box$) and *eventually* ($\Diamond$). Since ProbStarTL is defined only over discrete-time and bounded-time intervals, the *until* ($\mathcal{U}$) operator is evaluated using the equivalent formula composed of the *always* ($\Box$) and *eventually* ($\Diamond$) operators. To construct ProbStar traces, we develop ProbStar reachability algorithms, focusing on closed-loop LES reachability. This paper investigates exact and approximate ProbStar reachability algorithms for closed-loop LES with a ReLU feedforward neural network controlling a discrete linear plant model. Our approach could also be extended for verifying temporal behaviors of networks handling time-series data, such as recurrent neural networks [35]. Verifying an LES's temporal behaviors involves checking whether its ProbStar traces satisfy a user-defined ProbStar specification, which is done in two steps. First, the user-defined ProbStar specification is transformed into a *disjunctive normal form (DNF)* that exact or approximate verification algorithms can verify. The exact verification algorithm, while

computationally expensive, provides sound and complete verification results with exact satisfaction probability. In contrast, the approximate verification algorithm, less expensive than the exact one, estimates only the lower and upper bounds of satisfaction probability.

The ProbStar temporal logic verification framework is implemented using StarV, a new qualitative and quantitative verification tool for LES [34]. We evaluate the proposed framework using the well-known learning-based adaptive cruise control system (Le-ACC) [39] and the advanced emergency braking system (AEBS) [33]. The experimental results show that our approach has successfully verified multiple temporal properties of the Le-ACC system and AEBS (under a reasonable amount of time) that the state-of-the-art cannot verify. We have thoroughly analyzed the timing performance and conservativeness of the results via extensive experiments with two metrics, including conservativeness and constitution values. The conservativeness value shows how good the verification results are. The constitution value indicates when the exact probability of satisfaction can be achieved. We have also analyzed the verification complexity and discussed strategies for reducing it and increasing the scalability of our approach.

Contributions. We make the following contributions:

(1) Propose the ProbStarTL temporal logic for LES, with a procedure for computation of satisfaction probability.
(2) Depth-first Search Algorithm for exact (sound and complete) and approximate (sound) reachability analysis for constructing reachable set traces of closed-loop LES for verification.
(3) A new quantitative verification algorithm that can compute the exact and approximate satisfaction probability of temporal properties.
(4) A new verification tool was developed using Python and showcased in the Le-ACC and AEBS case studies.

2 PRELIMINARIES

In the following, we use the notation $\mathbb{R}$ for the reals and $\mathbb{N}$ for the natural numbers (including zero).

2.1 Probabilistic Star

Definition 2.1 (Probabilistic Star [34]). *A probabilistic star (or simply ProbStar) Θ is a tuple $\langle c, V, \mathcal{N}, P, l, u\rangle$ where $c \in \mathbb{R}^n$ is the center, $V = \{v_1, v_2, \cdots, v_m\}$ is a set of m vectors in $\mathbb{R}^n$ called basis vectors, $P : \mathbb{R}^m \rightarrow \{\top, \bot\}$ is a predicate, l and u are the lower-bound and upper-bound vectors of the predicate variables, which are random variables of a Gaussian distribution $\mathcal{N}$. The basis vectors are arranged to form the probstar's $n \times m$ basis matrix. The set of states represented by the probstar is given as:*

$$\begin{aligned}[\![\Theta]\!] = \{x \mid x = c + \Sigma_{i=1}^{m}(\alpha_i v_i),\ \alpha = [\alpha_1, \cdots, \alpha_m]^T \sim \mathcal{N},\\ P(\alpha) \triangleq C\alpha \leq d,\ l[i] \leq \alpha_i \leq u[i],\}.\end{aligned} \tag{1}$$

We will refer to both the tuple Θ and the set of states $[\![\Theta]\!]$ as Θ. Moving from the structural definition of a ProbStar, we now turn to defining its probabilistic characteristics.

Definition 2.2 (Probability). *Given a probstar Θ, the probability of the probstar is the probability of the predicate random variables $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_m]^T$ satisfying its constraints and bounds, i.e., $\mathbf{P}[\Theta] = \mathbf{P}[C\alpha \leq d \wedge\ l \leq \alpha \leq u,\ \alpha \sim \mathcal{N}(\mu, \Sigma)]$, where μ and Σ are the mean and the covariance of the predicate random variables, respectively. A probstar is an empty set if its probability is zero, i.e., $\mathbf{P}[\Theta] = 0$.*

2.2 Closed-loop LES

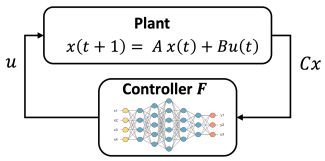


Figure 1: A closed-loop Learning-Enabled System (LES).

We aim to verify a closed-loop Learning Enabled System (LES) which comprises a plant with linear dynamics and a feedforward neural network controller as in Fig. 1. The plant $x(t+1) = Ax(t) + Bu(t)$ is a discrete time linear system with state $x \in \mathbb{R}^n$. The controller F is a ReLU feedforward neural network (FNN), processing inputs through layers with weights $W_{l,l-1}$, biases b_l, and ReLU activation functions to produce output $u = F(Cx)$. The output of each neuron is given by $v_i^{out} = ReLU(\Sigma_{j=1}^{n} w_{ij} v_j^{in} + b_i)$, where v^{in} is the input (output from the previous layer). In the following, we consider an LES to be the tuple $\mathcal{S} = \langle A, B, C, F\rangle$. Given a set of initial conditions X_0 and a maximum time T, then for every $x(0) \in X_0$ and for $t \in \mathbb{T} = [0, T-1] \subset \mathbb{N}$, a trajectory (signal) of LES is the solution of the usual iteration $x(t+1) = Ax(t) + BF(Cx(t))$.

Definition 2.3 (ProbStar Signal of LES). *Given an LES $\mathcal{S} = \langle A, B, C, F\rangle$, a time T and a ProbStar set of initial conditions X_0, we call the sequence $\mathcal{R} = (X_0, X_1, X_2, \ldots, X_T)$ a bounded-time ProbStar signal of the LES if*

$$\forall t \in \mathbb{T}\,.\, X_{t+1} = AX_t \oplus BF(CX_t) \tag{2}$$

Here, $\oplus$ is the Minkowski sum operation.

For notational convenience, we let $\mathcal{R}_t$ denote X_t. The following result is immediate.

Proposition 1. *For any $t \in \mathbb{T}$ and $x(0) \in [\![X_0]\!]$, we have $x(t) \in [\![X_t]\!]$. Conversely, for any $\tilde{x} \in [\![X_t]\!]$, there exists $x(0) \in [\![X_0]\!]$ such that $x(t) = \tilde{x}$.*

2.3 Discrete-time Temporal Logic for State Sequences

Many properties of interest for LES can be expressed through temporal logic over LES trajectories. Several variants of temporal logic have been developed which depend on whether (1) the trajectories are continuous time or discrete time, (2) the specifications are formulated over predicates or atomic propositions, (3) the properties are spatial and/or timed, and so on. In the following, for more information on temporal logic in the context of Cyber-Physical Systems (CPS), we recommend referring to the survey chapter [2].

In this work, the systems of interest, i.e., Fig. 1, produce discrete-time trajectories (signals) which can also be represented as state sequences up to a horizon T. We can ignore the state (sample) timestamps since we assume a constant sampling rate Δt. Given an LES state sequence $x : \mathbb{T} \rightarrow \mathbb{R}^n$, where $\mathbb{T} = \{0, 1, 2, \ldots, T\} \subset$

$\mathbb{N}$, we are looking to express properties over the signal values $x(t)$ through arithmetic expressions. A logic that can express such requirements is Signal Temporal Logic (STL) with discrete-time semantics. However, discrete-time semantics and bounded time intervals on the temporal operators reduce STL to Linear Temporal Logic (LTL) with syntactic sugar to capture the timing bounds (if we abstract the STL predicates by atomic propositions). Therefore, our specification language is actually a regular language and it could be described by regular expressions (RE) or finite automata (FA). Since we will not use FA in this work and conjunctions in RE are not standard, we will primarily use the STL notation and semantics for notational convenience.

Let $\mu_{p,q}$ be an *atomic predicate* defined as a linear inequality constraint on state variables x at time t with the canonical form:

$$(x,t) \models \mu_{p,q} \text{ iff } p^T x(t) + q \geq 0, \tag{3}$$

where $\models$ stands for *satisfies*, and p, q are some parameter vectors. In the later chapters, depending on the context, we will also denote the set of states x that satisfy $p^T x + q \geq 0$ by $[[\mu_{p,q}]]$ (or just $\mu_{p,q}$ depending on the context).

Definition 2.4 (DT-STL Syntax). *The discrete-time STL syntax is:*

$$\varphi := \mu_{p,q} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \Box_I \varphi,$$

where $\wedge$ is the and *Boolean operator, $\bigcirc$ is the next time operator, and $\Box_I$ is the always temporal operator over a bounded time interval $I \subset \mathbb{N}$.*

The standard Boolean (e.g., or ($\vee$), implies ($\rightarrow$)) and temporal operators (e.g., eventually ($\Diamond$)) can be derived using the usual equivalences. For example, *or* ($\vee$) is defined as $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$ and *eventually* ($\Diamond_I$) as $\Diamond_I \varphi \equiv \neg\Box_I \neg\varphi$. Notice that we do not consider the *until* operator ($\mathcal{U}$) in Def. 2.4. When the semantics are defined over discrete and bounded time, the until operator can be defined as a disjunction of a finite number of always and eventually operators, e.g.,

$$\phi_1 \mathcal{U}_{[m,n]} \phi_2 \equiv \vee_{i=m}^{n} (\Box_{[0,i-1]} \phi_1 \wedge \Diamond_{[i,i]} \phi_2).$$

or using a finite number of compositions of the next operator.

Definition 2.5 (DT-STL Semantics). *Given a discrete time signal $x : \mathbb{T} \rightarrow \mathbb{R}^n$ and a DT-STL specification φ, the semantics are defined by:*

$$\begin{aligned}
(x,t) &\models \mu_{p,q} && \text{iff} && p^T x(t) + q \geq 0,\\
(x,t) &\models \neg\varphi && \text{iff} && (x,t) \not\models \varphi\\
(x,t) &\models \varphi_1 \wedge \varphi_2 && \text{iff} && (x,t) \models \varphi_1 \wedge (x,t) \models \varphi_2\\
(x,t) &\models \bigcirc\varphi && \text{iff} && (x,t+1) \models \varphi\\
(x,t) &\models \Box_{[t_1,t_2]}\varphi && \text{iff} && \forall t' \in [t+t_1, t+t_2]\,.\,(x,t') \models \varphi
\end{aligned}$$

where $\not\models$ stands for "does not satisfy".

Without loss of generality, we assume that the signal length (i.e., T) is longer than the time horizon needed to evaluate an STL formula [26]. If this is not the case, then we can define alternative semantics where the formula horizon is permitted to be longer than the signal length [5].

3 PROBABILISTIC DT-STL SATISFIABILITY FOR LES

Problem Statement In this paper, our goal is to compute the probability that an LES $\mathcal{S}$ over a time domain $\mathbb{T}$ and with initial conditions in a ProbStar set X_0, satisfies a DT-STL formula φ. In other words, we would like to compute

$$\mathbf{P}[(x,0) \models \varphi \mid x(0) \in [[X_0]]]. \tag{4}$$

Toward that goal, we develop efficient probabilistic star reachability algorithms for LES's ProbStar signals under DT-STL specifications.

3.1 ProbStar Temporal Logic (ProbStarTL)

In order to compute probability (4), we need to "measure" how many trajectories starting from the initial ProbStar set X_0 satisfy the STL requirement.

As a foundation, such a computation needs to capture the number of trajectories satisfying an atomic predicate at a given time t. In other words, given an atomic predicate $\mu_{p,q}$ and some time t, we need to be able to assess $\mathbf{P}[(x,t) \models \mu_{p,q} \mid x(0) \in [[X_0]]]$. Due to Prop. 1, this is equivalent to $\mathbf{P}[[[X_t]] \cap [[\mu_{p,q}]]]$. For temporal operators, a formula like $\Box_{t_1,t_2}\mu_{p,q}$ would require satisfaction of the predicate $\mu_{p,q}$ for all times between t_1 and t_2, i.e., $\mathbf{P}[(x,t_1) \models \mu_{p,q} \wedge \ldots \wedge (x,t_2) \models \mu_{p,q} \mid x(0) \in [[X_0]]]$. Again, this is equivalent to $\mathbf{P}[[[X_{t_1}]] \cap [[\mu_{p,q}]] \wedge \ldots \wedge [[X_{t_2}]] \cap [[\mu_{p,q}]]]$. Note that since X_t is a ProbStar, its intersection with an (atomic) predicate, i.e., $[[X_t]] \cap [[\mu_{p,q}]]$, is also a ProbStar whose probability can be computed by Algorithm 1 in [34].

In order to generalize the above intuition to arbitrary DT-STL formulas, we will need to define a recursion that collects the constraints that must be satisfied over time. We refer to this recursive definition as ProbStarTL since it resembles DT-STL semantics over discrete-time signals. More concretely, ProbStarTL is a function C which returns – at the end of the recursion – a propositional formula over atomic propositions that represent sets. The propositional formula encodes which trajectories of the ProbStar signal have a non-zero probability of satisfying the DT-STL formula.

Definition 3.1 (ProbStarTL). *Given a ProbStar signal $\mathcal{R} = (X_0, X_1, \ldots, X_T)$ and a DT-STL formula φ in Negation Normal Form (NNF), the constraints over time that characterize the LES state trajectories x that satisfy φ can be derived recursively based on the structure of the formula φ:*

$$\begin{aligned}
C(\mathcal{R},t,\mu_{p,q}) &= [[X_t]] \cap [[\mu_{p,q}]] = [[X_t]] \cap \{x \mid p^T x + q \geq 0\},\\
C(\mathcal{R},t,\neg\mu_{p,q}) &= [[X_t]] \cap \overline{[[\mu_{p,q}]]} = [[X_t]] \cap \{x \mid p^T x + q < 0\},\\
C(\mathcal{R},t,\varphi_1 \wedge \varphi_2) &= C(\mathcal{R},t,\varphi_1) \wedge C(\mathcal{R},t,\varphi_2),\\
C(\mathcal{R},t,\varphi_1 \vee \varphi_2) &= C(\mathcal{R},t,\varphi_1) \vee C(\mathcal{R},t,\varphi_2),\\
C(\mathcal{R},t,\bigcirc\varphi) &= C(\mathcal{R},t+1,\varphi),\\
C(\mathcal{R},t,\Box_{[t_1,t_2]}\varphi) &= \wedge_{t'=t+t_1}^{t+t_2} C(\mathcal{R},t',\varphi),\\
C(\mathcal{R},t,\Diamond_{[t_1,t_2]}\varphi) &= \vee_{t'=t+t_1}^{t+t_2} C(\mathcal{R},t',\varphi),
\end{aligned}$$

where $\overline{S}$ indicates the complement of a set S.

We reiterate that the function C in Def. 3.1 does not return Boolean formulas, but rather symbolically captures the constraints that time-varying probability distributions must satisfy. The use of

Boolean operators simply provides notational convenience. In other words, the ProbStarTL formula $([[X_0]] \cap [[\mu_1]]) \vee ([[X_1]] \cap [[\mu_1]])$ captures the event of "*how many LES trajectories satisfy the predicate μ_1 at time zero, or satisfy the same predicate at time one*".

Notice that Def. 3.1 assumes that the DT-STL formula is in the Negation Normal Form (NNF). In NNF, negations ($\neg$) can only appear before atomic predicates. Any DT-STL formula can be converted into NNF by using the usual equivalences of Boolean and temporal operators and pushing the negation operators before the atomic predicates. We assume that a rewriting function $nnf(\varphi)$ converts a DT-STL formula φ in NNF. By definition, we have $\mathbf{P}[(x,0) \models \varphi \mid x(0) \in [[X_0]]] = \mathbf{P}[C(\mathcal{R}, t, nnf(\varphi))]$. Also notice that $\mathbf{P}[(x,0) \models \varphi \mid x(0) \in [[X_0]]] + \mathbf{P}[(x,0) \not\models \varphi \mid x(0) \in [[X_0]]] = 1$.

3.2 ProbStarTL Disjunctive Normal Form

Given a ProbStar signal $\mathcal{R}$ and a DT-STL specification φ, Definition 3.1 defines an algorithm that computes a symbolic representation of the trajectories that satisfy φ.

Example 1. *Let us consider the specification:*

$$\varphi_{e1} = \Box_{[0,1]} \left(\mu_1 \rightarrow \Diamond_{[1,2]} \mu_2 \right) \tag{5}$$

*for μ_1, μ_2 as in Example **??**. Given a ProbStar signal $\mathcal{R}$, we can compute symbolically the set of trajectories that satisfy φ_{e1}:*

$$\begin{aligned} C(\mathcal{R}, 0, \varphi_{e1}) &= \wedge_{t=0}^{1} C(\mathcal{R}, t, \mu_1 \rightarrow \Diamond_{[1,2]} \mu_2) \\ &= \wedge_{t=0}^{1} \left(C(\mathcal{R}, t, \mu_1) \rightarrow C(\mathcal{R}, t, \Diamond_{[1,2]} \mu_2) \right) \\ &= \wedge_{t=0}^{1} \left(C(\mathcal{R}, t, \mu_1) \rightarrow \vee_{t'=t+1}^{t+2} C(\mathcal{R}, t', \mu_2) \right) \\ &= \wedge_{t=0}^{1} \left(\neg([[X_t]] \cap [[\mu_1]]) \vee \vee_{t'=t+1}^{t+2} ([[X_{t'}]] \cap [[\mu_2]]) \right) \end{aligned}$$

However, computing the probability of satisfaction on the output, i.e., $\mathbf{P}[C(\mathcal{R}, 0, \varphi)]$, is a non-trivial computational task due to the arbitrary nesting of conjunctions and disjunctions. This issue can be addressed by placing the formula returned by $C(\mathcal{R}, 0, \varphi)$ into *Disjunctive Normal Form (DNF)*, i.e., a disjunction of conjunctive terms:

$$dnf(C(\mathcal{R}, 0, \varphi)) = \bigvee_i \left(\bigwedge_j \psi_{ij} \right)$$

where dnf denotes the function that translates any formula into its equivalent DNF, and each ψ_{ij} is some ProbStar $[[X_{t_{ij}}]] \cap [[\mu_{ij}]]$ or $[[X_t]] \cap \overline{[[\mu_{p,q}]]}$. Any propositional formula can be translated to DNF [4]. For instance, the formula $C(\mathcal{R}, 0, \varphi_{e1})$ in Example 1 is in Conjunctive Normal Form (CNF) with 2 clauses with 3 literals (ProbStars) each. On the other hand, using standard Boolean algebra transformations, the equivalent symbolic representation in DNF, i.e., $dnf(C(\mathcal{R}, 0, \varphi_{e1}))$, has 9 terms with 2 literals (ProbStars) each.

When a ProbStarTL formula is in DNF, then two predicates in a conjunctive term may impose different constraints on the same time slice of the ProbStar signal. That is, we may encounter terms of the form $([[X_t]] \cap [[\mu_1]]) \wedge ([[X_t]] \cap [[\mu_2]])$. Such terms can be simplified by collecting the constraints in a single set, i.e., $[[X_t]] \cap ([[\mu_1]] \cap [[\mu_2]])$. In the next section, we show how DNF can be used to provide a lower bound on the probability of satisfaction.

3.3 Probability of a ProbStar Computable Disjunctive Normal Form (CDNF)

To quantify the satisfaction of a specification φ on a ProbStar signal $\mathcal{R}$, we need to compute the probability of its associated ProbStarTL using the following lemma.

Lemma 2 (Probability of a ProbStarTL). *Given a DT-STL φ and a ProbStar signal $\mathcal{R}$, let*

$$\varphi_{dnf} = dnf(C(\mathcal{R}, 0, nnf(\varphi))) = \bigvee_{i=0}^{n} \psi_i = \bigvee_{i=0}^{n} \bigwedge_{j=0}^{m_n} \psi_{ij}$$

where ψ_{ij} correspond to ProbStars for some n, m_n. Then, the probability of satisfaction is

$$\begin{aligned} \mathbf{P}[\varphi_{dnf}] = \Sigma_{i=1}^{n} \mathbf{P}[\psi_i] &+ (-1)^1 \Sigma_{i \neq j} \mathbf{P}[\psi_i \wedge \psi_j] \\ &+ (-1)^2 \Sigma_{i \neq j \neq k} \mathbf{P}[\psi_i \wedge \psi_j \wedge \psi_k] \\ &+ \cdots + (-1)^{n-1} \mathbf{P}[\psi_1 \wedge \psi_2 \cdots \wedge \psi_n] \end{aligned}$$

and its estimated lower bound can be computed as:

$$\tilde{P}_l(\varphi_{dnf}) = \max(\mathbf{P}[\psi_1], \mathbf{P}[\psi_2], \ldots, \mathbf{P}[\psi_n]) \leq \mathbf{P}[\varphi_{dnf}].$$

Lemma 3 (Complexity in computing a ProbStarTL probability). *Let N_{exact} and $N_{estimate}$ be the number of ProbStar probability computations involved in computing the exact probability of $\varphi_{dnf} = dnf(C(\mathcal{R}, 0, nnf(\varphi))) = \vee_{i=0}^{n} \psi_i$ and estimating its lower bound. Then, we have: $N_{exact} = \Sigma_{k=1}^{n} C_n^k = 2^n - 1$, where C_n^k is the number of k-combinations of n elements, and $N_{estimate} = n$.*

4 QUANTITATIVE VERIFICATION OF LES

4.1 Probabilistic Star Reachability of LES

The k-step reachability analysis for LES (Figure 1) works as follows: At $t = 0$, the plant's state and output sets are $X_0 = \langle c_0, V_0, \mathcal{N}, P_0, l, u \rangle$ and $Y_0 = CX_0$, respectively. At $t = 1$, the neural network controller computes the control set from the feedback output set: $U_0 = F(Y_0)$. As F is a ReLU FNN, the exact control set is a union of m_1 ProbStars $U_0 = \{U_0^1, U_0^2, \ldots, U_0^{m_1}\}, U_0^i = \langle \tilde{c}_i, \tilde{V}_i, \mathcal{N}, \tilde{P}_i, l, u \rangle$ (see [34]). Note that the normal distribution $\mathcal{N}$ and the lower and upper bound vectors l and u do not change in the analysis. The reachable set of the plant is the Minkowski sum of the mapped initial state AX_0 and the mapped control input set BU_0, denoted as $X_1 = AX_0 \oplus BU_0$. We observe that the first step reachable set X_1 is also a union of multiple ProbStars, represented as $X_1 = X_1^1, X_1^2, \ldots, X_1^{m_1}$. Interestingly, similar to the star set approach [33], the state set X_0, and the control set U_0 are defined based on a unique set of random predicate variables. For any probstar U_0^i within the control set U_0, its predicate contains all linear constraints from the state set X_0. This results in a key characteristic: only a specific subset of X_0 can lead to an individual control set U_0^i, and the predicate of this control set precisely matches the predicate of that subset. Therefore, the i^{th} probstar in the first-step reachable set $X_1^i = \langle Ac_0 + B\tilde{c}_i, AV_0 + B\tilde{V}_i, \mathcal{N}, \tilde{P}_i, l, u \rangle$. To compute the next-step reachable set, i.e., X_2, we feedback the output $Y_1 = CX_1$ to the controller and repeat the same computation procedure.

Depth-first search (DFS) reachability algorithm. One can see that at any time step, due to the ReLU network controller, a single plant's state set can lead to multiple reachable sets in the

Algorithm 1 Depth-First Search Reachability Algorithm for LES

Inputs: LES $\mathcal{S}$, initial state set X_0, steps k, filter prob. p_f
Output: ProbStar signals $\mathcal{T}$, ignored trace probability p_{ig}

1: **procedure** REACH($\mathcal{S}, X_0, k, p_f$)
2: Initialize: $T \leftarrow [\ ]$, $R \leftarrow [X_0]$, $p_{ig} = 0$
3: **while** R not empty **do**
4: $i = |R|$, $R_i = R[i]$
5: **if** R_i empty **then**
6: $R.delete(i)$, $T.delete(i)$, **continue**
7: $X_i = R_i.pop()$, $T.append(X_i)$
8: $X_{i+1}, p^i_{ig} = stepReach(\mathcal{S}, X_i, p_f)$
9: $p_{ig} += p^i_{ig}$, $R.append(X_{i+1})$ if $i < k$ # update
10: **if** $i = k$ **then**
11: $\mathcal{T}.append(T + [x])$ for x in X_{i+1}, $T.delete(i)$ # store traces
12: **return** $\mathcal{T}, p_{ig}$
13: **procedure** STEPREACH($\mathcal{S}, X_i, p_f$)
14: $A, B, C \leftarrow \mathcal{S}.C$ # extract matrices
15: $Y_i = CX_i, U_i, p^i_{ig} = F(Y_i, p_f)$ # compute control set
16: $X_{i+1} = [AX_i \oplus BU^j_i$ for U^j_i in $U_i]$ # compute next step
17: **return** X_{i+1}, p^i_{ig}

next time step. This means that from a signal initial set of state X_0, a LES can produce multiple ProbStar signals (reachable set traces) $\mathcal{T} = \{T_1, T_2, \ldots, T_M\}$. A k-step ProbStar signal is defined as $T_i = X_0 \rightarrow X^i_1 \rightarrow X^i_2 \rightarrow \cdots \rightarrow X^i_k$ in which X^i_j is produced by $X^i_{j-1}, 1 \leq j \leq k$. To construct the ProbStar signals for LES, we need to perform the reachability analysis in a *depth-first search manner* in which we need to keep track of the *production chains* of reachable sets. By doing that, we know that X_0 produces X^1_1 ($X_0 \rightarrow X^1_1$), and X^1_1 produces X^1_2, and so on. Algorithm 1 presents our DFS reachability algorithm for LES. The inputs to the algorithm are the LES $\mathcal{S} = \langle F, \mathcal{M} \rangle$, the plant's initial state set X_0, the number of time steps k, and the filtering probability p_f. The filtering probability p_f is used to filter out all the traces in the analysis whose probabilities are smaller than p_f. If $p_f = 0$, the exact reachability analysis scheme is performed to construct all ProbStar signals created by the initial state set X_0. The algorithm returns the list of all ProbStar signals $\mathcal{T}$ and the total probabilities of ignored ProbStar signals in the analysis p_{ig}. Our algorithm works as follows. We initialized the trace T as an empty list, the remaining sets R as X_0, and the total probability of ignored traces p_{ig} as zeros (line 2). We use a while loop to perform forward reachability analysis with remaining sets R and only stop when R is empty. Note that the length of R, denoted as $i = |R|$, is also the current considering time step (line 4). At the current time step i, we get the intermediate sets of this step $R_i = R[i]$ (line 4). If R_i is empty, we delete the empty R_i, reset the trace T to the previous trace T_{i-1} (line 5-6), and start the forward reachability analysis with step $i-1$ with a new set X_{i-1} the remaining sets in this step R_{i-1}. Otherwise, we get the first set X_i in R_i, add it to the trace T (line 7), and compute the reachable set for the next step using *stepReach* subprocedure (line 8). The *stepReach* procedure produces the reachable set of the next time step X_{i+1} and the total probability of ignored reachable sets p_{ig}. If the next step $i+1$ is the final time step k, i.e., $i+1 == k$ (line 10), then all the sets in X_{i+1} are the leaves of the trace T. Therefore, we construct all traces with these leaves and add them to the list of traces $\mathcal{T}$ (line 11) and then delete the last set in the current trace (to move forward later). If the next step $i+1$ is not the final step k, we add the constructed set X_{i+1} into the intermediate set R (line 9) and keep moving forward.

Lemma 4 (Reachability Complexity). *Given an LES $\mathcal{S} = \langle F, \mathcal{M} \rangle$ and a probstar initial condition X_0, the number of reachable set traces produced by the DFS reachability algorithm (Algorithm 1) in the worst case is $O(2^{k*N})$, where k is the number of time steps and N is the total number of ReLU neurons in the network controller F.*

4.2 Quantitative verification algorithm

After constructing the ProbStar signals of LES, we can verify these signals against a temporal specification written using ProbStarTL. Algorithm 2 describes our quantitative verification for LES. The inputs to the algorithm are the considering LES $\mathcal{S}$, its initial set of states X_0, number of time steps k, filtering probability p_f, and the specification φ. The algorithm returns upper (ρ_{max}) and lower (ρ_{min}) bounds of satisfaction probability, the conservativeness value $v_{conserv}$, and the constitution value $v_{constit}$, defined in the following:

$$v_{conserv} = \begin{cases} 100 * \frac{\rho_{max} - \rho_{min}}{\rho_{max}}, & \text{if } \rho_{max} > 0 \\ 0, & \text{otherwise} \end{cases}, \quad v_{constit} = 100 * \frac{\rho_{ig}}{\rho_{max}}.$$

The conservativeness value $v_{conserv}$ shows how tight the estimation range of satisfaction probability is. The higher the conservativeness value, the more conservative the estimation is. The conservativeness value is zero if $\rho_{max} = \rho_{min}$ or $\rho_{max} = 0.0$. The constitution value $v_{constit}$ shows how much the total probability of the ignored traces and CDNFs contributes to the estimation. If $v_{constit} = 0$, there are no ignored traces or CDNFs in verification there for the ρ_{max} is the exact satisfaction property. If $v_{constit} = 100\%$, the estimation of $\rho_{max} = \rho_{ignored}$, i.e., the estimation is entirely based on the optimistic assumption that all ignored traces and CDNFs satisfy the property. The algorithm works as follows. We initialize all verification results as zeros in line 2. Using Algorithm 1, we construct all ProbStar signals $\mathcal{T}$ and compute the probability of ignored traces p_{ig} (line 3). We loop over all signals (traces) in the list $\mathcal{T}$ (line 4). For each signal T in $\mathcal{T}$, we get the CDNF φ_{CDNF} of the specification φ on T (line 5). We check the feasibility of the obtained CDNF φ_{CDNF}. If feasible, the lower and upper bounds of satisfaction probability for trace T (ρ_T) are computed using its CDNF (lines 7 and 8). If the CDNF is too large, i.e., its length is larger than 11; its corresponding trace T will be ignored too with the ignored probability p'_{ig}. The upper bound of satisfaction probability ρ_1 from computing the probability of this CDNF will be the ignored probability p''_{ig}. We advance the total probability bounds and the total probability of ignored traces in line 8. After looping through all traces, we advance the upper bound of the satisfaction probability ρ_{max} by p_{ig} (the total probability of ignored traces in reachability). We compute the conservativeness and constitution values in lines 10 and 11 and return all verification results in line 12.

Verification complexity. The verification complexity depends on the number of traces produced in reachability analysis and the complexity of the specification. The final verification result is obtained by computing the probabilities of ProbStar CDNF formulas derived from realizing the corresponding specification on multiple traces (Definition 3.1). The size of the CDNF formula increases

Algorithm 2 Quantitative Verification Algorithm for LES

Inputs: $\mathcal{S}, X_0, k, p_f, \varphi$ # LES, initial state set, number of steps, filtering probability, specification

Outputs: $\rho_{max}, \rho_{min}, v_{conserv}, v_{constit}$ # Quantitative verification results

procedure $\rho_{max}, \rho_{min}, v_{conserv}, v_{constit}$ = verify($\mathcal{S}, X_0, k, p_f, \varphi$)
  $\rho_{max} = 0, \rho_{min} = 0, v_{conserv} = 0, v_{constit} = 0, p'_{ig} = 0$
  $\mathcal{T}, p_{ig} = reach(\mathcal{S}, X_0, k, p_f)$ # Algorithm 1
  for T in $\mathcal{T}$ **do**
    $\varphi_{CDNF} = getDNF(\varphi, T)$
    if φ_{CDNF} is feasible **then**
      $\rho_1, \rho_2, \rho''_{ig} = getProbability(\varphi_{CDNF})$ # Lemma 2
      $\rho_{max} += \rho_1, \rho_{min} += \rho_2, \rho'_{ig} += \rho''_{ig}$
  $\rho_{max} += p_{ig}$
  if $\rho_{max} > 0$: **then**
    $v_{conserv} = 100 * (\rho_{max} - \rho_{min})/\rho_{max}, v_{constit} = 100(\rho_{ig} + \rho'_{ig})/\rho_{max}$
  return $\rho_{max}, \rho_{min}, v_{conserv}, v_{constit}$

(potentially exponentially) along with the size of the specification (if we have disjunctive predicates). Therefore, if the specification is too large, we may encounter a memory problem when constructing the CDNF formula, which can contain millions of disjunctive constraints. The following lemma describes the computational complexity of our quantitative verification algorithm.

Lemma 5 (Verification complexity). *Given an LES $\mathcal{S} = \langle F, \mathcal{M} \rangle$, a probstar initial condition X_0, and a specification φ, the computational complexity (the number of ProbStar probability computations) for computing the exact probability of $\mathcal{S} \models \varphi$ in the worst-case is $O(2^{kN+L})$, where k is the number of time steps, L is the maximum length of the CDNF formulas, and N is the number of neurons of the network controller F.*

5 EVALUATION

We implement the probabilistic star temporal logic verification approach using StarV, a new tool for qualitative and quantitative verification of DNNs and Le-CPS written in Python [34]. The code for reproducing all results in this paper will be available for artifact evaluation. The experiments were executed on an iMAC 3.8 GHz 8-Core Intell Core i7 with a 128GB memory with a virtual 64-bit Ubuntu 20.04.4 LTS system.

5.1 Case Study 1: Verification of Le-ACC System.

We evaluate our verification approach using the well-known learning-based adaptive cruise control (Le-ACC) system used in the ARCH competition [19, 39].

Scenarios and complex temporal properties of interest. When the ego vehicle is in the speed control mode and at a safe distance, the lead vehicle driver suddenly decelerates with $a_{lead} = -5(m^2/s)$ to reduce the speed. The neural network controller is expected to decelerate the ego vehicle to maintain a safe distance between the two cars. The properties of interest are given in Table 1. For property φ_1, we want to compute the probability of the system being unsafe at some steps between 0 and T, i.e., $D_r \leq D_{safe}$. Conversely, for property φ'_1, we want to compute the probability of the system being always safe between 0 and T. We note that $\rho_{\varphi_1} + \rho_{\varphi'_1} = \mathbb{P}_{X_0}$. For property φ_2, we want to compute the probability that the lead or ego cars reduce their speed to avoid collision. The property φ'_2 is opposite to the property φ_2. For property φ_3, we want to verify that eventually, in T steps, when the lead car reduces its speed, the ego car also reduces its speed within five steps. Property φ_4 specifies the expected reactive behavior requiring that always the case that between 0 and T, if two cars are in an unsafe distance, eventually, two cars will be in a safe distance again within five steps. Opposite to property φ_4, property φ'_4 states that eventually, between 0 and T, there is the case that when two cars are under an unsafe distance, they are always in the same unsafe condition for the next five steps.

Specification
$\varphi_1 = \diamond_{[0,T]}(x_{lead}(t) - x_{ego}(t) \leq D_{safe} = 10 + 1.4v_{ego}(t))$
$\varphi'_1 = \square_{[0,T]}(x_{lead}(t) - x_{ego}(t) \geq D_{safe} = 10 + 1.4v_{ego}(t))$
$\varphi_2 = \diamond_{[0,T]}((v_{lead}(t) \leq min(v_{lead}(0)) - 0.1) \vee (v_{ego}(t) \leq min(v_{ego}(0)) - 0.1))$
$\varphi'_2 = \square_{[0,T]}((v_{lead}(t) \geq min(v_{lead}(0)) - 0.1) \wedge (v_{ego}(t) \geq min(v_{ego}(0)) - 0.1))$
$\varphi_3 = \diamond_{[0,T]}((v_{lead}(t) \leq min(v_{lead}(0)) - 0.1) \wedge \diamond_{[0,5]}(v_{ego}(t) \leq min(v_{ego}(0)) - 0.1))$
$\varphi_4 = \square_{[0,T]}((x_{lead}(t) - x_{ego}(t) \leq D_{safe}) \rightarrow \diamond_{[0,5]}((x_{lead}(t) - x_{ego}(t) \geq D_{safe})$
$\varphi'_4 = \diamond_{[0,T]}((x_{lead}(t) - x_{ego}(t) \leq D_{safe}) \wedge \square_{[0,5]}((x_{lead}(t) - x_{ego}(t) \leq D_{safe})$

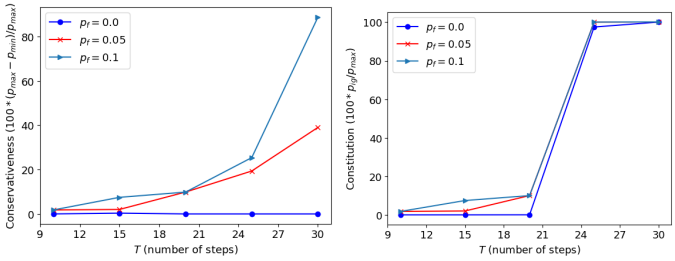
Table 1: Properties of interest of the Le-ACC system.

Can our approach verify quantitatively the temporal complex behaviors of Le-ACC? Our approach successfully verifies multiple temporal properties of the Le-ACC system. Table 2 describes the verification results of Le-ACC with the network controller $N_{5\times20}$ (i.e., 5 layers, 20 neurons per layer) for different properties with and without filtering under different time steps T. We can verify all properties within a reasonable verification time except for φ_4, where we have a memory problem due to its large abstract disjunctive formula. However, as mentioned above, property φ'_4 is an opposite property of φ_4. Therefore, φ_4 can be verified via verifying φ'_4. Property φ_1 is satisfied for all T. For properties φ_2 and φ'_2, we can see that there is a zero probability that both cars do not reduce their speed. For property φ_3, there is a high probability (0.95124) that when the lead car reduces its speed, the ego car also reduces its speed in five time steps. For φ'_4, there will be a case with a high probability (e.g., 0.95124 for $T = 30$) that the expected reactive behavior of the system is not satisfied, i.e., the system cannot recover to a safe condition in 5 steps after it goes into the unsafe condition. Figure 5b depicts a reachable set trace showing this system behavior.

How conservative the verification results are, and what effects the conservativeness of verification? Can we compute the exact probability of property satisfaction? The conservativeness of the verification results comes from two sources: 1) some traces with very small probabilities are ignored in verification, and 2) some very large CDNFs are ignored in the exact probability computation (Lemma 2). As we ignore these traces and CDNFs, we can optimistically estimate the upper bound of the satisfaction probability by assuming that all ignored traces and CDNFs will satisfy the considering property. If no traces and CDNFs are ignored in verification, then the upper bound ρ_{max} is the exact probability of satisfaction. The conservativeness value $v_{conserv}$ shows how tight the estimation range is. The higher it is, the more conservative the result is. The conservativeness value is zero when whether $\rho_{max} = \rho_{min}$ or $\rho_{max} = 0.0$. In both cases, we achieve the exact satisfaction probability. The constitution value $v_{constit}$ shows how much the total probability of the ignored traces and CDNFs contributes to the estimation. If $v_{constit} = 0$, there are no ignored traces

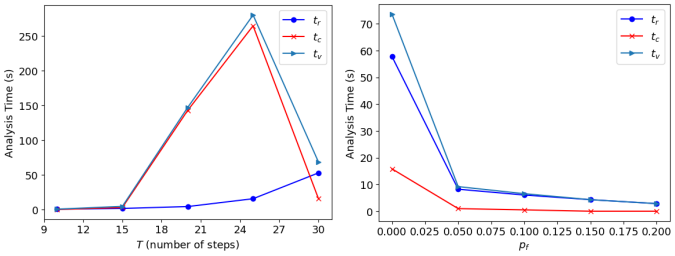
Spec.	T	$p_f = 0.0$ (No Filtering)					$p_f = 0.1$ (Filtering)				
		ρ_{max}	ρ_{min}	t_r	t_c	t_v	ρ_{max}	ρ_{min}	t_r	t_c	t_v
φ_1	10	0.00316878	0.00316878	0.384465	0.0641825	0.448647	0.00316851	0.00311178	0.390045	0.0602827	0.450327
φ_1'		0.948399	0.948399	0.384465	0.0525982	0.437063	0.948058	0.947352	0.390045	0.0535336	0.443578
φ_2		0.95124	0.95124	0.384465	0.152381	0.536846	0.95124	0.95124	0.390045	0.156047	0.546092
φ_2'		0	0	0.384465	0.0101957	0.394661	5.67218e-05	0	0.390045	0.0140345	0.404079
φ_3		0.95124	0.95124	0.384465	0.474227	0.858692	0.95124	0.95124	0.390045	0.463002	0.853047
φ_4'		0.00316001	0.00315984	0.384465	0.0671768	0.451642	0.00316544	0.00310871	0.390045	0.0603747	0.450419
φ_1	20	0.95124	0.95124	3.30168	64.3575	67.6591	0.95124	0.857019	1.54465	16.227	17.7716
φ_1'		0	0	3.30168	0.0672245	3.36891	0.0952739	0	1.54465	0.0190148	1.56366
φ_2		0.95124	0.95124	3.30168	2.29778	5.59946	0.95124	0.85819	1.54465	0.630488	2.17514
φ_2'		0	0	3.30168	0.0711493	3.37283	0.0952739	0	1.54465	0.0198286	1.56448
φ_3		0.95124	0.95124	3.30168	6.07836	9.38004	0.95124	0.859574	1.54465	1.72259	3.26723
φ_4'		0.95124	0.95124	3.30168	142.133	145.435	0.95124	0.858195	1.54465	35.1492	36.6938
φ_1	30	0.95124	0.95124	40.683	13.3291	54.0122	0.95124	0.106357	4.77714	0.406348	5.18349
φ_1'		0	0	40.683	0.731146	41.4142	0.846995	0	4.77714	0.0279951	4.80514
φ_2		0.95124	0.95124	40.683	36.9469	77.63	0.95124	0.106571	4.77714	1.08114	5.85828
φ_2'		0	0	40.683	0.769494	41.4525	0.846995	0	4.77714	0.0272472	4.80439
φ_3		0.95124	0.95124	40.683	102.669	143.352	0.95124	0.106539	4.77714	3.01574	7.79289
φ_4'		0.95124	0.95124	40.683	15.5658	56.2488	0.95124	0.106327	4.77714	0.486822	5.26397

Table 2: Verification results for Le-ACC with the network controller $N_{5\times20}$ (i.e., 5 layers, 20 neurons per layer) in which ρ_{max} and ρ_{min} are the upper and lower bound of prob. of satisfaction; p_f is the filtering prob., t_r, t_c and t_v are the reachability time, checking time and total verification time in seconds respectively.



(a) **(b)**

Figure 2: Conservativeness analysis of φ_1. Conservativeness varies with the number of time steps, and more filtering leads to more conservative results. The ρ_{max} will be the exact satisfaction probability if its constitution is zero, i.e., no traces or CDNFs are ignored in verification.



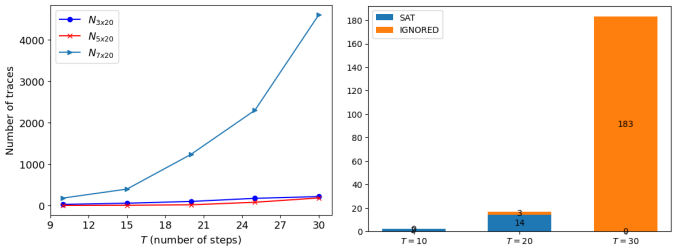
(a) **(b)**

Figure 3: Verification timing performance of φ_4'. The reachability time t_r increases along with the selected number of steps T. However, the total verification time t_v fluctuates due to the fluctuation in the checking time t_c. The analysis times t_r, t_c, and t_v can be reduced by applying filtering in verification, i.e., set $p_f > 0$.

or CDNFs in verification there for the ρ_{max} is the exact satisfaction property. If $v_{constit} = 100\%$, the estimation of $\rho_{max} = \rho_{ignored}$, i.e., the estimation is entirely based on the optimistic assumption. Figure 2a shows the conservativeness of verification results for property φ_1 in different time steps. We can see that without filtering out traces in analysis, i.e., $p_f = 0$ in Algorithm 2, the verification results for φ_1' are very good (they are exact satisfaction probabilities). The figure shows that increasing the filtering probability p_f generally increases the conservativeness of verification results. For example, if we choose $p_f = 0.1$, the conservativeness of verification results increases from $\approx 1.2\%$ at $T = 10$ to $\approx 87.9\%$ at $T = 30$. If $p_f = 0.05$, the conservativeness of verification results increases from $\approx 1.2\%$ at $T = 10$ to $\approx 39\%$ at $T = 30$. The constitution $V_{constit}$ shows the percentage of ignored traces or CDNFs contributing to satisfaction probability estimation. Figure 2b depicts the constitution values of φ_1' verification. With $p_f = 0$, the constitution values for all $T \leq 20$ are zeros, which means the verification results are the exact satisfaction probabilities. When $T = 25$ and $T = 30$, the ignored CDNFs contribute $\approx 100\%$ in estimating satisfaction probability. For $p_f = 0.1$, the ignored traces and CDNFs contribute $\approx 9\%$ for $T = 20$, and $\approx 100\%$ for $T = 25$ and $T = 30$ in estimating satisfaction probability.

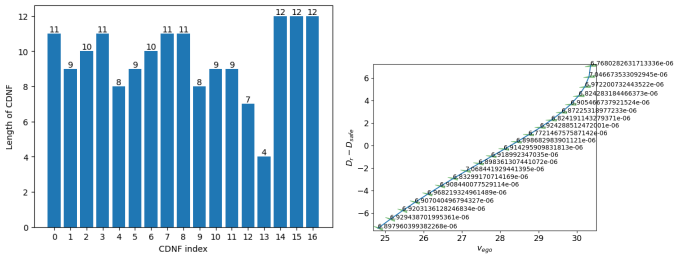
How do the complexities of the specifications affect verification time? We use the number of time steps T to adjust the complexities of the specifications and monitor its effect. As shown in Table 2, the total verification time t_v varies for different properties and T. Both reachability time t_r and checking time t_r can dominate the total verification time. For example, for $T = 30$, in φ_2' verification, reachability time dominates the verification (40.683s vs. 0.769494s), while in φ_3 verification, the checking time is the most significant one (102.669s vs. 40.683s). Another example is, in φ_4' verification, the checking time is the significant one (142.133s vs. 3.30168s) for $T = 20$, but the reachability time is the dominant one (0.384s vs. 0.067s) for $T = 10$. One can see from the Table or Figure 3a that the reachability time increases when we increase the number of time steps T in verification. This is because the number of traces increases along with the growth of the time steps involved in the analysis (this fact is shown in Figure 4a). However, the total verification time fluctuates significantly due to the fluctuation in the checking time. One can see from Figure 3a that the checking time for $T = 25$ is larger than the ones for $T = 15$ and $T = 30$. Figure 3b shows that we can reduce the verification time significantly by applying filtering. However, the cost is the conservativeness of verification results (as analyzed above).

How are the verification complexity and scalability? what affects verification complexity and scalability? In our approach, the verification complexity depends on 1) the number of traces involved in verification, 2) the number of CDNFs after realizing the specification on these traces, and 3) the lengths of obtained CDNFs. The number of traces of a LES varies significantly for different numbers of time steps, networks, and initial conditions. Different neural network controllers may behave differently on the same initial conditions, even if trained with the same data set. Figure 4a shows the number of traces of three trained network controllers $N_{3\times20}$, $N_{5\times20}$, and $N_{7\times20}$ for Le-ACC. The biggest controller $N_{7\times20}$ behaves very complicatedly as it creates more than 4500 traces at $T = 30$, which makes the verification tasks even more challenging and time-consuming. Therefore, filtering and parallel computation are crucial techniques to reduce verification complexity and



(a) (b)

Figure 4: Verification complexity depends on 1) the number of traces (which varies for different networks and different initial conditions), 2) the number of CDNFs, and 3) the lengths of CDNFs. Large CDNFs (one with a length larger than 11) are ignored in verification, which leads to a conservative estimation of the maximum probability of satisfaction.



(a) Length of CDNF for φ_4'. (b) A satisfaction trace.

Figure 5: Length of CDNFs for φ_4' verification with $T = 20$ and the visualization of a trace satisfying the specification. CDNFs with lengths larger than 11 are ignored in exact verification, and an upper bound of the probability of satisfaction (ρ_{max}) is given in this case with an optimistic view that the ignored CDNFs will satisfy the specification.

improve scalability. The second factor affecting the verification complexity is the number of CDNFs. Not every trace in the reachable traces will create a CDNF after the realization. Interestingly, the number of CDNFs for a property may vary significantly for different numbers of time steps. A smaller number of time steps does not guarantee a smaller number of CDNFs, which makes the total verification fluctuate for different time ranges. The last factor affecting the verification complexity is the lengths of CDNFs. From Lemma 3, computing the exact probability of a CDNF with the length of n, requires $2^n - 1$ ProbStar probability estimation. Therefore, to reduce the verification complexity and increase the scalability, we ignore large CDNFs in verification. The total probability of ignored CDNFs is used to estimate the upper bound of satisfaction, with an optimistic view that all ignored CDNFs will satisfy the property. In our implementation, $n_{max} = 11$ is the maximum length allowed for computing the exact satisfaction probability. Figure 5a shows the lengths of CDNFs produced in φ_4' verification with $T = 20$. Figure 4b describes the number of CDNFs produced and ignored in verifying φ_4' for different numbers of time steps T and $p_f = 0.0$ (i.e., no filtering). We note that ignoring large CDNFs with small probabilities is also a critical technique for improving the scalability of our verification approach. For example, at $T = 20$, we ignore 3 CDNFs in verification but still can achieve a good estimation for satisfaction probability for φ_4', i.e., $[0.95124, 0.858195]$ (Table 2)

Can we intuitively verify if a temporal property is being satisfied? One significant advantage of our verification approach is the ability to visualize satisfied traces, which helps users intuitively verify and interpret the verification results. Figure 5b illustrates one satisfactory trace with probability for φ_4' property with $T = 20$ (note that we have multiple satisfactory traces for this property). For this visualization, one can clearly see that there is the case that when two cars go into an unsafe condition, i.e., $D_r \leq D_{safe}$, they keep staying in this condition for the next 5 steps. Note that we can also visualize all reachable set traces produced in the reachability analysis process. It would be interesting to explore in the future whether we can use satisfactory traces to repair or retrain the neural network controller to increase its probability of being satisfactory for some specific properties.

Comparison with the state-of-the-art. The neurosymbolic verification [11] is possibly the first approach to verifying a neural network control system against temporal properties specified using the well-known STL formalism. The authors introduce a custom FNN architecture with ReLU activation functions for general STL specifications. Together with the neural network control system model, the complex verification problem is reduced to the reachability analysis of a large FNN whose output range represents the robustness values of satisfaction. There are two key differences between our ProbStarTL approach and the neurosymbolic one. Firstly, the neurosymbolic approach follows the quantitative semantics from the traditional STL to compute the robustness value. While we borrow the STL syntax, our quantitative semantics is new as it focuses on computing the probability of satisfaction when the initial conditions are formulated as a multivariate constrained Gaussian distribution. Therefore, it is more suitable for applications involving probabilistic uncertainties, e.g., robotic applications. Secondly, we separate the reachability process and the property verification. Therefore, we do not encounter the reachability analysis of large networks. This allows us to verify multiple specifications simultaneously when the reachability process is done, saving the verification time. Table 3 illustrates the verification results from two approaches for Le-ACC with different network controllers for property $\varphi^* = \square_{[0,T]}((x_{lead}(t) - x_{ego}(t) \leq D_{safe}) \rightarrow \diamond_{[0,3]}((x_{lead}(t) - x_{ego}(t) \geq D^*_{safe})$, where $D^*_{safe} = D_{safe} + 2$ (the φ_3 property in [11]). Our approach provides the probability of satisfaction, which is consistent with the neurosymbolic approach, showing that the system is robustly safe for entire initial conditions (note that the initial condition in this comparison is from [11], which is different from the initial conditions we use above). As shown in the table, our approach is significantly faster than the neurosymbolic in all cases. The verification times of the two approaches increase along with the size of the property φ^* (i.e., increasing T).

5.2 Case Study 2: Verification of AEBS

The AEBS [33] is more complex than the considered ACC system due to the two neural networks involved in the feedback loop. To

		$T = 10$		$T = 20$		$T = 30$		$T = 50$	
CtrlNet	Method	RS	VT (sec)	RS	VT (sec)	RS	VT (sec)	RS	VT (sec)
$N_{3\times20}$	ProbStar	[0.9512, 0.9512]	0.1248	[0.9512, 0.9512]	2.9854	[0.9512, 0.9512]	10.1	[0.9512, 0.9512]	33.3988
	NeuroSymbolic	[26.9014, 48.2194]	4.9362	[26.9014, 48.2194]	12.3515	[26.2468, 48.0778]	34.8895	[16.6622, 38.4598]	160.843
$N_{5\times20}$	ProbStar	[0.9512, 0.9512]	1.7683	[0.9512, 0.9512]	5.892	[0.9512, 0.9512]	9.141	[0.9512, 0.9512]	29.0227
	NeuroSymbolic	[26.9067, 48.2244]	13.4654	[26.9067, 48.2244]	76.0396	[26.9067, 48.2244]	137.227	[24.0621, 44.3187]	356.74

Table 3: Verification results (robustness intervals) of NeuroSymbolic [11] are consistent with the proposed ProbStarTL verification results (probabilities of satisfaction). The proposed ProbStarTL verification approach is significantly faster than the NeuroSymbolic. *RS* is the verification result, VT is the verification time (in seconds), and T is the number of time steps.

$X_0 = d_0 \times v_0$	T	p_f	ρ_{max}	ρ_{min}	t_r	t_c	t_v	N_{traces}
$X_0^1 = [97, 97.5] \times [25.2, 25.5]$	10	0	0	0	2.87992	0.0844164	2.96433	14
	10	0.01	0.0119663	0	2.50623	0.0484138	2.55464	7
	20	0	0	0	20.7128	0.307941	21.0207	32
	20	0.01	0.0239297	0	10.8651	0.119331	10.9844	12
	40	0	0	0	71.6828	0.64255	72.3254	38
	40	0.01	0.0240082	0	33.2209	0.302022	33.5229	15
	50	0	0	0	106.702	1.19582	107.897	56
	50	0.01	0.0549497	0	47.889	0.427383	48.3164	19
$X_0^2 = [90, 90.5] \times [27, 27.2]$	10	0	0	0	4.57537	0.062093	4.63747	9
	10	0.01	0.0114052	0	3.45863	0.0471869	3.50582	6
	20	0	0	0	17.339	0.227031	17.566	23
	20	0.01	0.0222215	0	10.4313	0.103984	10.5353	9
	40	0	0	0	79.692	1.50938	81.2014	87
	40	0.01	0.0818814	0	30.472	0.409358	30.8814	19
	50	0	0	0	171.335	2.7099	174.045	129
	50	0.01	0.156198	0	50.8068	0.47649	51.2833	19
$X_0^3 = [48, 48.5] \times [30.2, 30.4]$	10	0	0	0	3.70928	0.0676892	3.77697	11
	10	0.01	0.00711187	0	1.91794	0.0293272	1.94727	3
	20	0	0	0	15.7227	0.285852	16.0086	30
	20	0.01	0.0355909	0	7.72026	0.129606	7.84987	12
	40	0	0	0	189.011	4.00749	193.019	231
	40	0.01	0.256797	0	49.1817	0.57451	49.7562	29
	50	0	0.975316	0.975316	415.776	70.6082	486.384	305
	50	0.01	0.975316	0.664081	78.6132	7.52558	86.1388	31
$X_0^4 = [5, 5.2] \times [1, 1.2]$	10	0	0	0	22.9072	0.432894	23.3401	62
	10	0.01	0.0264034	0	6.45371	0.0863433	6.54006	9
	20	0	0	0	95.07	1.23194	96.302	111
	20	0.01	0.0540896	0	17.464	0.150403	17.6144	12
	40	0	0	0	302.169	3.63554	305.805	209
	40	0.01	0.0635139	0	40.5005	0.23968	40.7401	12
	50	0	0	0	532.584	7.94045	540.525	372
	50	0.01	0.19101	0	54.787	0.24902	55.036	10

Table 4: Quantitative verification results of AEBS system against property $\varphi = \diamond_{[0,T]}(d_k \leq L \wedge v_k \geq 0.2)$ under different initial conditions in which ρ_{max} and ρ_{min} are the upper and lower bound of probability of satisfaction; p_f is the filtering probability; t_r, t_c and t_v are the reachability time, checking time and total verification time in seconds respectively; N_{traces} is the number of traces involved in verification.

test the controller reaction, we verify the system's safety under different initial conditions $d_0 \times v_0$ for T time steps and against the following specification: $\varphi = \diamond_{[0,T]}(d_k \leq L \wedge v_k \geq 0.2)$ which states that eventually in T time steps, the system will reach the unsafe region in which the distance between the car to the obstacle is smaller than $L = 2.5$ meters and the car's velocity is still larger than 0.2 m/s.

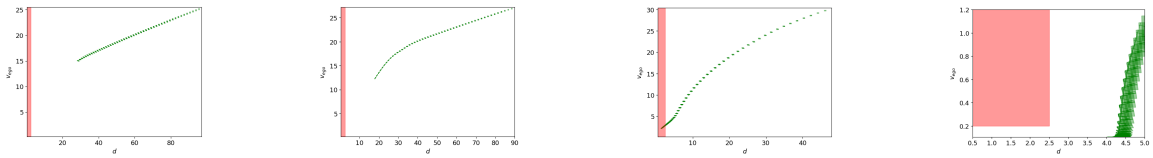
Verification results. Our approach successfully verifies the temporal property of the AEBS system under different initial conditions. The verification results are presented in Table 4. One can see that without filtering, the property is not satisfied (i.e., with zero satisfaction probability) for all initial conditions for all time ranges T, which means the system does not reach the unsafe condition for any time step k between 0 and T, except for the third scenario X_0^3 at $T = 50$. In the third scenario, when $48 \leq d_0 \leq 48.5$ and $27 \leq v_0 \leq 27.2$, the RL controller can not reduce the car's speed to lower than 0.2 m/s when the car is close to the obstacle with a distance $d \leq 2.5$ at step $T = 50$. The probability of satisfaction in this case is 0.975316, which is equal to the probability of the initial condition. This indicates that all initial states are unsafe.

Conservativeness and exact probability of satisfaction. Table 4 shows the exact satisfaction probabilities when $p_f = 0$ (no filtering, all traces are used in verification). When applying filtering $p_f = 0.01$, the estimated bounds for satisfaction probabilities are quite good for all scenarios. We can see that when the number of considered time steps T increases, the verification result (with filtering) is more conservative as the number of traces ignored in verification increases. For example, for the first scenario X_0^1, the estimated range of satisfaction probability is widened from $[0.0119663, 0]$ at $T = 10$ to $[0.0549497, 0]$ at $T = 50$. This is because when applying filtering, only 7 traces ($= 14 - 7$) are ignored in verification for $T = 10$, while when $T = 50$, the number of traces ignored in verification increases to 37 ($= 56 - 19$).

Timing performance analysis. In this case study, the reachability time dominates the verification time in all scenarios. The checking time is significantly small compared to the reachability time. For example, for X_0^4 with $T = 50$, the reachability time is $t_r = 54.787s$ which is 220× larger than the checking time $t_c = 0.249$. We note that the timing performance analyzed here is only valid for the specific considered property φ. Various specifications may lead to different checking times. We can see that the verification time grows rapidly with the increase in the number of considered time steps T due to the increase in the number of traces and the complexity of the corresponding CDNFs. For example, without filtering, the verification time for the fourth scenario X_0^4 grows from 23.34 seconds for $T = 10$ to 540 seconds for $T = 50$. By applying filtering in verification, the verification time is reduced to 55.036 seconds (the trade-off cost is the conservative estimation of satisfaction probability).

Verification complexity and scalability. The verification complexity varies for different initial conditions because the neural network components in the AEBS system behave quite differently for various initial conditions. For example, for the first scenario X_0^1, the number of traces produced at step $T = 50$ (without filtering) is 56, while in the second scenario X_0^2, the number of traces is 129. Not only more traces make verification complexity increase. The CDNF formulas obtained from realizing the specification on a trace are also different in length for different traces. More traces increase the possibility of more complex CDNF in verification, leading to more verification time and memory consumption. The table shows that our approach scales quite well with the sizes of the property (when increasing T).

Intuitive verification and reachable set trace visualization. The verification results can be intuitively justified using reachable set trace visualization. Figure 6 demonstrates the reachable set traces for the AEBS system for all scenarios. We can see that in scenarios X_0^1, X_0^2 and X_0^4, the system's states do not reach the unsafe region, i.e., $d_k \leq 2.5 \wedge v_k \geq 0.2$ after T steps (T = 10, 20, 40, 50).



(a) $[97, 97.5] \times [25.2, 25.5]$. **(b)** $[90, 90.5] \times [27, 27.2]$. **(c)** $[48, 48.5] \times [30.2, 30.4]$. **(d)** $[5, 5.2] \times [1, 1.2]$.

Figure 6: 50-step reachable sets (d_k vs. v_k) of AEBS system (in green) and the unsafe region (in red) for different initial conditions (scenarios) $d_0 \times v_0$. d_k and v_k are the ego car's distance (to the obstacle) and velocity at step k. The AEBS system is safe under all initial conditions except scenario (c). Although safe in scenario (d), the controller does not do well to avoid the early stopping scenario (the car should stop at $0.5 \leq d_k \leq 2.5$).

Therefore, the exact satisfaction probability for these cases is zero (Table 4). For the third scenario, the whole system's states reach the unsafe region at step 50. Therefore, the property φ is satisfied with a high probability of 0.975316 (which equals the initial condition probability). Interestingly, Figure 6-d shows that the RL controller reduces the car's speed to avoid collision. However, it cannot avoid the early stopping scenario as desired (i.e., stop the car with the expected distance $0.5 \leq d \leq 2.5$).

6 RELATED WORKS

Open-loop LES verification. Rigorous research has been done to verify *open-loop* LES [14, 24], focusing on the safety, robustness, and fairness of deep neural networks. Notable research on open-loop LES verification includes: Satisfiability Modulo Theory (SMT) [20, 21], optimization using mixed integer linear programming encoding [23, 25], reachability [1, 27, 34–39], facet-vertex incidence matrix [44], symbolic interval [42], semidefinite programming [6], abstract interpretation [7, 8, 30, 31, 40–43, 45, 46], input quantization [18], and relaxed convex programming [22], to name a few. This paper does not focus on the open-loop LES verification problem. However, our proposed probstar temporal logic can also be used for verifying temporal behaviors of neural networks involved in time-series data, such as recurrent neural networks [17, 35].

Closed-loop LES verification. Closed-loop LES verification focuses on the safety of closed-loop neural network control systems under bounded input conditions, involving complex interactions between the neural network controller and the physical plant model [38]. Notable closed-loop LES verification frameworks include VeriSig [16], ReachNN [13], Sherlock [3], and NNV [33]. Recently, significant effort has been made to verify perception-based control systems, which have proven to be significantly challenging and time-consuming due to large input space [10, 12, 15, 28, 32].

Most verification techniques for (open-loop/closed-loop) LES focus on *qualitative verification* to answer whether or not a safety, robustness, or fairness property is violated. ProbStar reachability is the first approach to quantitatively verify ReLU neural networks with a precise estimation of satisfaction probability under constrained Gaussian input uncertainties [34]. Recently, a new probabilistic abstraction named Zonotopic Dempster Shafer was introduced to construct tight overapproximation of the probabilistic outputs of a ReLU network using Interval Dempster Shafer arithmetic and probabilistic affine arithmetic [9]. Unlike the ProbStar approach, this approach can handle more general input uncertainties and provide guaranteed results. Nevertheless, there is a lack of quantitative verification methods to verify complex, *temporal properties* of LES, i.e., properties involve timing information. Neurosymbolic approach [11] is the first approach to verify closed-loop LES with STL properties. By introducing a novel transformation technique, verifying the temporal properties of a closed-loop LES is equivalent to performing reachability analysis of large feedforward neural networks. Our proposed approach is similar to the neurosymbolic approach in the sense that we target temporal properties verification. However, our approach is distinguished from the neurosymbolic one as we define our new set-based logic ProbStarTL whose semantics are satisfaction of constraints on random variables, which allows us to compute exactly the probability of satisfaction. Furthermore, our approach is direct as it does not involve transformation from STL to neural networks, which allows us to avoid performing reachability analysis of large networks.

7 CONCLUSION

In this paper, we have developed a novel quantitative verification framework for analyzing the temporal behaviors of learning-enabled systems (LES). Our approach has been implemented in StarV and successfully verified many useful, practical temporal properties of the well-known learning-enabled ACC and AEBS systems. Our verification approach's timing performance, conservativeness, complexity, and scalability have been analyzed via extensive experiments. In the future, we want to extend this verification frame to LES with nonlinear dynamics and optimize its timing performance so that it can be used at runtime. Importantly, we want to develop a generic quantitative verification approach for verifying temporal reactive behaviors of internal neural network components in complex LES (i.e., the LES containing multiple neural network components interacting with each other and the physical world, e.g., the AEBS system). We also want to extend this framework to verify neural networks with time-series data.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation (NSF) under grants NSF-CAREER-2441334, NSF-SLES-2331937, and NSF-FMitF-2220418. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. 2020. Improved Geometric Path Enumeration for Verifying ReLU Neural Networks. In *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer.

[2] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Nickovic, and Sriram Sankaranarayanan. 2018. Specification-based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications. In *Lectures on Runtime Verification - Introductory and Advanced Topics*. LNCS, Vol. 10457. Springer, 128–168.

[3] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 157–168.

[4] Herbert B. Enderton. 2001. *A Mathematical Introduction to Logic*. Elsevier.

[5] Georgios E. Fainekos and George J. Pappas. 2006. Robustness of Temporal Logic Specifications. In *Formal Approaches to Testing and Runtime Verification (LNCS)*, Vol. 4262. Springer, 178–192.

[6] Mahyar Fazlyab, Manfred Morari, and George J Pappas. 2020. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Trans. Automat. Control* (2020).

[7] Marc Fischer, Christian Sprecher, Dimitar Iliev Dimitrov, Gagandeep Singh, and Martin Vechev. 2022. Shared certificates for neural network verification. In *International Conference on Computer Aided Verification*. Springer, 127–148.

[8] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

[9] Eric Goubault and Sylvie Putot. 2024. A Zonotopic Dempster-Shafer Approach to the Quantitative Verification of Neural Networks. In *International Symposium on Formal Methods*. Springer, 324–342.

[10] P Habeeb, Deepak D'Souza, Kamal Lodaya, and Pavithra Prabhakar. 2024. Interval Image Abstraction for Verification of Camera-Based Autonomous Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 11 (2024), 4310–4321.

[11] Navid Hashemi, Bardh Hoxha, Tomoya Yamaguchi, Danil Prokhorov, Georgios Fainekos, and Jyotirmoy Deshmukh. 2023. A neurosymbolic approach to the verification of temporal logic properties of learning-enabled control systems. In *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)*. 98–109.

[12] Chiao Hsieh, Yangge Li, Dawei Sun, Keyur Joshi, Sasa Misailovic, and Sayan Mitra. 2022. Verifying controllers with vision-based perception using safe approximate abstractions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4205–4216.

[13] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. 2019. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–22.

[14] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review* 37 (2020), 100270.

[15] Radoslav Ivanov, Taylor J Carpenter, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. 2020. Case study: verifying the safety of an autonomous racing car with a neural network controller. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. 1–7.

[16] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. 2019. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 169–178.

[17] Yuval Jacoby, Clark Barrett, and Guy Katz. 2020. Verifying recurrent neural networks using invariant inference. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 57–74.

[18] Kai Jia and Martin Rinard. 2021. Verifying low-dimensional input neural networks via input quantization. In *International Static Analysis Symposium*. Springer, 206–214.

[19] Taylor T Johnson, Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Elena Botoeva, Francesco Leofante, Amir Maleki, Chelsea Sidrane, Jiameng Fan, and Chao Huang. 2020. ARCH-COMP20 category report: artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. *EPiC Series in Computing* 74 (2020).

[20] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

[21] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. 2019. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 443–452.

[22] Haitham Khedr, James Ferlez, and Yasser Shoukry. 2021. Peregrinn: Penalized-relaxation greedy neural network verifier. In *International Conference on Computer Aided Verification*. Springer, 287–300.

[23] Haitham Khedr and Yasser Shoukry. 2023. Certifair: A framework for certified global fairness of neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 8237–8245.

[24] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. 2021. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization* 4, 3-4 (2021), 244–404.

[25] Alessio Lomuscio and Lalit Maganti. 2017. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351* (2017).

[26] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *Proceedings of FORMATS-FTRTFT (LNCS)*, Vol. 3253. 152–166.

[27] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. 2018. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242* (2018).

[28] Ulices Santa Cruz and Yasser Shoukry. 2022. Nnlander-verif: A neural network formal verification framework for vision-based autonomous aircraft landing. In *NASA Formal Methods Symposium*. Springer, 213–230.

[29] Sanjit A Seshia, Dorsa Sadigh, and S Shankar Sastry. 2022. Toward verified artificial intelligence. *Commun. ACM* 65, 7 (2022), 46–55.

[30] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*. 10825–10836.

[31] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 41.

[32] Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. 2019. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 147–156.

[33] Hoang-Dung Tran, Feiyang Cai, Manzanas Lopez Diego, Patrick Musau, Taylor T Johnson, and Xenofon Koutsoukos. 2019. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–22.

[34] Hoang-Dung Tran, Sungwoo Choi, Hideki Okamoto, Bardh Hoxha, Georgios Fainekos, and Danil Prokhorov. 2023. Quantitative Verification for Neural Networks using ProbStars. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*. 1–12.

[35] Hoang Dung Tran, SungWoo Choi, Tomoya Yamaguchi, Bardh Hoxha, and Danil Prokhorov. 2023. Verification of Recurrent Neural Networks using Star Reachability. In *The 26th ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*.

[36] Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. 2019. Star-Based Reachability Analsysis for Deep Neural Networks. In *23rd International Symposisum on Formal Methods (FM'19)*. Springer International Publishing.

[37] Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Stanley Bak, and Taylor T Johnson. 2021. Robustness verification of semantic segmentation neural networks using relaxed reachability. In *International Conference on Computer Aided Verification*. Springer, 263–286.

[38] Hoang-Dung Tran, Weiming Xiang, and Taylor T Johnson. 2020. Verification approaches for learning-enabled autonomous cyber-physical systems. *IEEE Design & Test* (2020).

[39] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. 2020. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*. Springer, 3–17.

[40] Shubham Ugare, Debangshu Banerjee, Sasa Misailovic, and Gagandeep Singh. 2023. Incremental Verification of Neural Networks. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 1920–1945.

[41] Caterina Urban, Maria Christakis, Valentin Wüstholz, and Fuyuan Zhang. 2020. Perfectly parallel fairness certification of neural networks. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–30.

[42] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1599–1614.

[43] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems* 33 (2020), 1129–1141.

[44] Xiaodong Yang, Tomoya Yamaguchi, Hoang-Dung Tran, Bardh Hoxha, Taylor T Johnson, and Danil Prokhorov. 2021. Reachability Analysis of Convolutional Neural Networks. *arXiv preprint arXiv:2106.12074* (2021).

[45] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2022. General cutting planes for bound-propagation-based neural network verification. *Advances in neural information processing systems* 35 (2022), 1656–1670.

[46] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*. 4944–4953.