LTL Verification of Memoryful Neural Agents

Mehran Hosseini
King's College London
London, United Kingdom
mehran.hosseini@kcl.ac.uk

Alessio Lomuscio
Imperial College London
London, United Kingdom
a.lomuscio@imperial.ac.uk

Nicola Paoletti
King's College London
London, United Kingdom
nicola.paoletti@kcl.ac.uk

ABSTRACT

We present a framework for verifying Memoryful Neural Multi-Agent Systems (MN-MAS) against full Linear Temporal Logic (LTL) specifications. In MN-MAS, agents interact with a non-deterministic, partially observable environment. Examples of MN-MAS include multi-agent systems based on feed-forward and recurrent neural networks or state-space models. Different from previous approaches, we support the verification of both bounded and unbounded LTL specifications. We leverage well-established bounded model checking techniques, including lasso search and invariant synthesis, to reduce the verification problem to that of constraint solving. To solve these constraints, we develop efficient methods based on bound propagation, mixed-integer linear programming, and adaptive splitting. We evaluate the effectiveness of our algorithms in single and multi-agent environments from the Gymnasium and PettingZoo libraries, verifying unbounded specifications for the first time and improving the verification time for bounded specifications by an order of magnitude compared to the SoA.

KEYWORDS

Formal Verification; Neural Net Verification; Recurrent Neural Nets; Verification of Multi-Agent Systems; Safe Reinforcement Learning

ACM Reference Format:
Mehran Hosseini, Alessio Lomuscio, and Nicola Paoletti. 2025. LTL Verification of Memoryful Neural Agents. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 10 pages.

1 INTRODUCTION

With the rise of machine learning, we are increasingly witnessing applications of neural networks (NNs) to autonomous systems. While *neural agents* may solve significantly more complex tasks than traditionally programmed counterparts [44, 51], NNs suffer from fragilities [45], which is a concern in safety-critical applications. Thus, they require rigorous guarantees, as those offered by formal verification.

In this paper, we develop the methodology for verifying systems comprising multiple memoryful neural agents that interact with an uncertain environment over time in a closed loop. We refer to such systems as *Memoryful Neural Multi-Agent System* (*MN-MAS*).

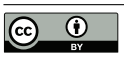
The verification of MN-MAS is more challenging than the verification of individual predictions made by NNs, due to the presence of multiple networks, multiple time steps, and temporal dependencies between predictions. Several methods for the verification of systems with NNs "in the loop" have been recently proposed [16, 28]. Nevertheless, these are limited by at least two of the following: they only consider single-agent systems; are limited to the analysis of (time-)bounded reachability; do not account for uncertainty in the environment; or the agents are implemented by feedforward models, thus are stateless/memoryless.

We propose a verification approach for MN-MAS that addresses all the above limitations. The approach presented here supports verification of MN-MAS against full *linear temporal logic* [40]. Thus, the method presented here goes beyond simple bounded reachability analysis and, unlike existing methods, can verify both bounded and *unbounded* specification, as well as nested properties. These include (unbounded) safety (e.g., *will an agent always remain in a safe state region?*) or stability (e.g., *will an agent reach its target and remain there long enough thereafter?*).

Although we show that the verification of unbounded specifications is undecidable in general, we develop sound or complete procedures based on Bounded Model Checking (BMC) [9], which allows us to verify unbounded properties of the MN-MAS using finite paths. In particular, we develop three BMC procedures, based on, respectively, simple path unrolling, lasso search, and inductive invariants. Thus, when the presented method establishes that an MN-MAS S satisfies an LTL specification ϕ, written $S \models \phi$, then ϕ must hold for all the paths of the system induced by the uncertain (non-deterministic) environment dynamics and initial states.

With BMC, we reduce the verification problem to solving a set of logical constraints over the MN-MAS states. In particular, we target linearly definable systems, i.e., systems that can be formulated by a finite set of piecewise-linear (PWL) constraints. This class of systems remains quite expressive, as it includes, among others, neural agents with ReLU-based activations.

We introduce two efficient algorithms to solve the resulting PWL constraints. The first is *Bound Propagation through Time (BPT)*, an extension of interval and linear bound propagation [19, 52] to support memoryful models, such as those based on *Recurrent Neural Networks* (*RNNs*). BPT is highly efficient but results in an over-approximation of the model's outputs, meaning that it cannot be used in all verification instances (see § 3.2). The second is a *Mixed-Integer Linear Programming* (*MILP*)-based algorithm, called *Recursive MILP* (*RMILP*), which allows encoding the constraints and variables during multiple time steps in MILP. Unlike BPT, the latter is precise (i.e., does not overapproximate) and is more efficient than past MILP approaches for recurrent models [4, 28] (see § 5.3). We also employ *adaptive splitting* [35], which exploits the information about the reachable sets computed by BPT to resolve *a priori* some of the discrete variables of the MILP instances, thereby considerably simplifying the MILP encoding.



Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

In summary, we make the following contributions: **(1)** we introduce the first framework for verifying memoryful, neural multi-agent systems against full LTL specifications under uncertainty; **(2)** we characterize the decidability of the verification problem; **(3)** we develop efficient solutions based on bounded model checking, bound propagation, and constraint solving; and **(4)** we evaluate the approach on deep Reinforcement Learning (RL) benchmarks with single and multiple agents, demonstrating its efficiency compared to the SoA (being strictly faster in all instances, up to one order of magnitude) and its versatility in verifying, for the first time, both time-bounded and unbounded specifications.

Related Work. Following several seminal works exposing the vulnerability of machine learning models to adversarial perturbations [10, 45], significant attention has been paid to the verification of neural networks against input-output relations. A special class of these consists of the verification of local robustness including output reachability. Most solution techniques are based on (a combination of) bound propagation [19, 52] and constraint solving [11, 15, 32, 36, 48].

While this line of research focuses on open-loop systems, we here consider closed-loop systems with NN components, i.e., sequential processes characterised by multiple interdependent NN predictions over time. In recent years, several methods have been proposed to verify this kind of systems [2, 3, 7, 16, 18, 28, 30, 43, 50, 54, 55].

Among the above, the work of Akintunde et al. [3] focuses on temporal logic verification of multi-agent systems but considers a bounded fragment of the logic and agents implemented via feedforward NNs (FFNN), i.e., memoryless models. Closer to this contribution is the method of Akintunde et al. [4], which supports RNN-based agents but is limited to single-agent systems and bounded LTL properties, and its recent extension by Hosseini and Lomuscio [28], which introduces an inductive invariant algorithm for unbounded safety. Our work considerably expands on these previous proposals in that it is the first to support multi-agent systems comprising neural memoryful agents and partially-observable uncertain environments, as well as arbitrary LTL specifications, including unbounded and nested formulas.

Verifying recurrent models on the other hand is significantly more challenging—even for recurrent linear models w.r.t. time-unbounded linear specifications [6, 12, 27, 29, 39, 47].

To verify models with recurrent layers, one approach is to explicitly unroll the RNN to derive an equivalent FFNN [4]. However, the size of the resulting network blows up with the unrolling depth (i.e., the time bound). This problem is mitigated in [28] by introducing a (more efficient) direct MILP encoding of the RNN constraints, but still results in often unfeasible queries. Other related RNN verification methods, but applied to open-loop systems, include [34], based on linear bound propagation, and [31], which derives a time-invariant overapproximation of the RNN. Our solution extends the MILP encoding of [28] to allow for multiple agents and composite $\text{LTL}_{\mathbb{R}}$ formulae, and further enhances it with bound propagation techniques and adaptive splitting [25, 35], as we discuss in §3.2.

A related line of work concerns the verification of multi-agent reinforcement learning (MARL) systems with neural agents [38, 41, 56, 57]. The main difference is that in MARL the environment is stochastic while in MN-MASit is non-deterministic.

2 BACKGROUND & PROBLEM FORMULATION

We denote the n-dimensional vector space over $\mathbb{R}$ by $\mathbb{R}^n$. Vectors $\boldsymbol{x} \in \mathbb{R}^n$ are shown by bold italic typeface while scalars $x \in \mathbb{R}$ are shown by normal italic typeface. We denote the set of all finite sequences over a set $C \subseteq \mathbb{R}^n$ by C^* and the set of all countable infinite sequences over C by C^ω. We use $(\boldsymbol{x}^{\langle i \rangle})_{i=0}^{n}$ and $(\boldsymbol{x}^{\langle i \rangle})_{i \in \mathbb{N}}$ to indicate a sequence in $(\mathbb{R}^n)^*$ and $(\mathbb{R}^n)^\omega$, respectively. The t-th element of such sequence denoted by $\boldsymbol{x}^{\langle t \rangle}$. For a function f, its t consecutive applications is denoted by $f^{(t)}$. For the sake of brevity, we treat sets like vectors and scalars; e.g., for a set $\mathcal{P} \subseteq \mathbb{R}^n$, we use $\boldsymbol{c}^\intercal \cdot \mathcal{P}$ to refer to the set $\{\boldsymbol{c}^\intercal \cdot \boldsymbol{x} : \boldsymbol{x} \in \mathcal{P}\}$.

Next, we define PWL constraints, and then, in the rest of the section, we lay out the problem formulation.

Definition 1. A *Piecewise-Linear* (*PWL*) *constraint* is a constraint of the form

$$A_{\boldsymbol{x}}\boldsymbol{x} + A_{\boldsymbol{y}}\boldsymbol{y} + A_{\boldsymbol{\delta}}\boldsymbol{\delta} \leq \boldsymbol{d}, \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^{n_{\boldsymbol{x}}}, \boldsymbol{y} \in \mathbb{R}^{n_{\boldsymbol{y}}}$, and $\boldsymbol{\delta} \in \Delta \subseteq \mathbb{Z}^{n_{\boldsymbol{\delta}}}$ are vector variables (or simply variables), and $A_{\boldsymbol{x}} \in \mathbb{Q}^{n_c \times n_{\boldsymbol{x}}}, A_{\boldsymbol{y}} \in \mathbb{Q}^{n_c \times n_{\boldsymbol{y}}}, A_{\boldsymbol{\delta}} \in \mathbb{Q}^{n_c \times n_{\boldsymbol{\delta}}}$, and $\boldsymbol{d} \in \mathbb{Q}^{n_c}$ are real-valued matrices. The variable $\boldsymbol{\delta}$ is called the *discrete variable* and often $\Delta = \{0, 1\}^{n_{\boldsymbol{\delta}}}$, i.e., $\boldsymbol{\delta}$ is a Boolean variable. Variables $\boldsymbol{x}$ and $\boldsymbol{y}$ are the *free* and *pivot* variables, respectively. In other words, Eq. (1) defines $\boldsymbol{y}$ linearly in terms of $\boldsymbol{x}$, in a piecewise fashion because of the discrete variable $\boldsymbol{\delta}$. We say that $\boldsymbol{y}$ as a PWL functions of $\boldsymbol{x}$ if for each $\boldsymbol{y} \in \mathbb{R}^{n_{\boldsymbol{y}}}$, there exists at most one $\boldsymbol{x} \in \mathbb{R}^{n_{\boldsymbol{x}}}$ that satisfies Eq. (1).

Problem Formulation

We consider systems comprising a non-deterministic and partially observable environment with multiple agents, controlled by arbitrary PWL neural policies. Such policies can be memoryful, e.g., RNN-based, or without memory, e.g., FFNN-based. In this paper, we focus on the verification of specifications expressed in *Linear Temporal Logic* over the *reals*, $\text{LTL}_{\mathbb{R}}$, an extension of LTL [40] with atomic predicates involving real variables, as defined in Def. 4.

Definition 2 (Memoryful Neural Multi-Agent System)**.** A *Memoryful Neural Multi-Agent System* (*MN-MAS*) S comprises s *agents* interacting over time with a non-deterministic *environment*, as per following equations.

$$\boldsymbol{x}^{\langle t \rangle} \in \tau(\boldsymbol{x}^{\langle t-1 \rangle}, \boldsymbol{a}_1^{\langle t-1 \rangle}, \ldots, \boldsymbol{a}_s^{\langle t-1 \rangle}), \tag{2}$$

$$\boldsymbol{a}_i^{\langle t \rangle} = \alpha_i(\boldsymbol{o}_i^{\langle 1 \rangle}, \ldots, \boldsymbol{o}_i^{\langle t \rangle}), \tag{3}$$

$$\boldsymbol{o}_i^{\langle t \rangle} = o_i(\boldsymbol{x}^{\langle t \rangle}). \tag{4}$$

In Eqs. (2) to (4),

- $\boldsymbol{x}^{\langle t \rangle} \in \mathcal{X}$ is the *state* of the environment at time t and $\boldsymbol{x}^{\langle 0 \rangle} \in \mathcal{X}^{\langle 0 \rangle}$ is the *initial state* of the environment, with $\mathcal{X} \subseteq \mathbb{R}^m$ being the environment's *state space* and $\mathcal{X}^{\langle 0 \rangle} \subseteq \mathcal{X}$ the set of *initial states*;
- $(\boldsymbol{a}_1^{\langle t \rangle}, \ldots, \boldsymbol{a}_s^{\langle t \rangle}) \in \mathcal{A}$ are the *actions* performed at time t by agents $1, \ldots, s$ respectively, where $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_s$ and $\mathcal{A}_i$ is the *action space* of agent i; and
- $\tau : \mathcal{X} \times \mathcal{A} \to 2^{\mathcal{X}}$ is the environment's *transition relation* which, given the current state of the environment $\boldsymbol{x} \in \mathcal{X}$ and agents' actions $\boldsymbol{a} \in \mathcal{A}$, returns the set of possible next states $\tau(\boldsymbol{x}, \boldsymbol{a}) \subseteq \mathcal{X}$.

Moreover, for each agent $i = 1, \ldots, s$,

- $\boldsymbol{o}_i^{\langle t \rangle} \in O_i$ is agent i's *observation* of the environment at t, with $O_i \subseteq \mathbb{R}^{\ell}$ being the agent's *observation space*; observations are made by an *observation function* $o_i : \mathcal{X} \to O_i$;
- $\alpha_i : O_i^* \to \mathcal{A}_i$ is the agent i's *policy*, which given a finite sequence of environment observations returns an admissible action from $\mathcal{A}_i$. The policy α_i is realised via a *recurrent function* $r_i : O_i \times \mathcal{H}_i \to \mathcal{A}_i \times \mathcal{H}_i$, where $\mathcal{H}_i \subseteq \mathbb{R}^{n_i}$ is the agent's *set of hidden states*. The function r_i, defined

$$r_i(\boldsymbol{o}_i^{\langle t \rangle}, \boldsymbol{h}_i^{\langle t \rangle}) = (\boldsymbol{a}_i^{\langle t \rangle}, \boldsymbol{h}_i^{\langle t+1 \rangle}), \tag{5}$$

  maps observation $\boldsymbol{o}_i^{\langle t \rangle}$ and agent's hidden state $\boldsymbol{h}_i^{\langle t \rangle}$ to action $\boldsymbol{a}_i^{\langle t \rangle}$ and updated state $\boldsymbol{h}_i^{\langle t+1 \rangle}$. When $t = 0$, $\boldsymbol{h}_i^{\langle 0 \rangle} \in \mathcal{H}_i^{\langle 0 \rangle}$, the set of *initial hidden states* of agent i. We refer to $\boldsymbol{h}_i$ as the *memory* or *hidden state* of the agent.

When the transition relation τ is a function $\tau : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$, the environment is *deterministic*. We say that agent i is *memoryless* or *feed-forward* whenever $\mathcal{H}_i = \emptyset$. As done for $\mathcal{A}$, we define $O = O_1 \times \cdots \times O_s$ and $\mathcal{H} = \mathcal{H}_1 \times \cdots \times \mathcal{H}_s$ to be the *sets of systems's joint observations* and *hidden states*, respectively. We denote by $\boldsymbol{h} \in \mathcal{H}$ and $\boldsymbol{o} \in O$ a *joint observation* and a *joint hidden state*, respectively. Consequently, we denote the *initial set of joint agent states* by $\mathcal{H}^{\langle 0 \rangle}$. This allows us to define the *joint observation function* $o(\boldsymbol{x}) = (o_1(\boldsymbol{x}), \ldots, o_s(\boldsymbol{x}))$ and *joint policy* $r(\boldsymbol{o}, \boldsymbol{h}) = (\boldsymbol{a}, \boldsymbol{h}') = (r_1(\boldsymbol{o}_1, \boldsymbol{h}_1), \ldots, r_s(\boldsymbol{o}_s, \boldsymbol{h}_s))$. We can now define the *system evolution* $e(\boldsymbol{x}, \boldsymbol{h}) = (\tau(\boldsymbol{x}, \boldsymbol{a}), \boldsymbol{h}')$, which given the current joint system state $\boldsymbol{x}$ and joint memory $\boldsymbol{h}$, returns the next joint system state $\tau(\boldsymbol{x}, \boldsymbol{a})$ and joint memory $\boldsymbol{h}'$.

Assumption 1. *We will focus on MN-MAS that are linearly definable or can be linearly approximated, that is, τ, r, o, and the sets of initial states $\mathcal{X}^{\langle 0 \rangle}, \mathcal{H}^{\langle 0 \rangle}$, can be expressed by finite conjunction and disjunction of linear inequalities. This assumption is insignificant as it captures NN policies with ReLU-like activations, among others. Moreover, non-linear systems can be linearly approximated to an arbitrary level of precision as described in [1, 14].*

Remark 1. In the experiments, we implement memoryful policies in Eq. (5) using RNNs and memoryless policies using FFNNs. Our formulation however is more general and can accommodate other kinds of memoryful models, such as LSTM [26], memory networks [53], and state space models [17], such as Mamba [21].

Definition 3 (MN-MAS path)**.** A sequence $\pi = (\boldsymbol{x}^{\langle 0 \rangle}, \boldsymbol{h}^{\langle 0 \rangle}, \boldsymbol{x}^{\langle 1 \rangle}, \boldsymbol{h}^{\langle 1 \rangle}, \ldots) \in (\mathcal{X} \times \mathcal{H})^{\omega}$ is a path of an MN-MAS S if, for any $t \geq 0$, $\boldsymbol{x}^{\langle t+1 \rangle}$ and $\boldsymbol{h}^{\langle t+1 \rangle}$ are recursively derived from $\boldsymbol{x}^{\langle t \rangle}$ and $\boldsymbol{h}^{\langle t \rangle}$ following Eqs. (2) to (5). We denote the set of all infinite paths of S by $\Pi_S \subseteq (\mathcal{X} \times \mathcal{H})^{\omega}$. For a path $\pi \in \Pi_S$ and index t, we denote with $\pi_{\mathcal{X}}^{\langle t \rangle} \in \mathcal{X}$ and $\pi_{\mathcal{H}}^{\langle t \rangle} \in \mathcal{H}$ the environment state and agents' hidden states, respectively, at time t in π. For a subset of systems paths $\mathcal{P} \subseteq \Pi_S$, we denote its projection onto $\mathcal{X}$ by $\mathcal{P}_{\mathcal{X}}$, defined as $\mathcal{P}_{\mathcal{X}} = \{\boldsymbol{x} \in \mathcal{X} : (\boldsymbol{x}, \boldsymbol{h}) \in \mathcal{P} \text{ for some } \boldsymbol{h} \in \mathcal{H}\}$. The projection of $\mathcal{P}$ onto $\mathcal{H}$ is denote by $\mathcal{P}_{\mathcal{H}}$ and is defined similarly.

We can now proceed to introduce the $\text{LTL}_{\mathbb{R}}$ specifications.

Definition 4 (Specifications)**.** The *Linear Temporal Logic* over the *reals*, $\text{LTL}_{\mathbb{R}}$, is defined by following BNF.

$$\phi ::= \boldsymbol{c}^{\intercal} \cdot \boldsymbol{x} \leq d \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \bigcirc \phi \mid \phi \,\mathcal{U}\, \phi \mid \phi \mathcal{R} \phi \mid \Box\phi \mid \Diamond\phi,$$

where $\boldsymbol{c} \in \mathbb{R}^m$ and $d \in \mathbb{R}$ are constants. $\text{LTL}_{\mathbb{R}}$ atoms are linear inequalities of the form $\boldsymbol{c}^{\intercal} \cdot \boldsymbol{x} \leq d$ involving the environment state $\boldsymbol{x}$. If a formula contains any of the temporal operators $\mathcal{U}, \mathcal{R}, \Box, \Diamond$, it is an *unbounded formula*; otherwise, a *bounded formula*.

Remark 2. $\text{LTL}_{\mathbb{R}}$ atoms are defined over the environment's states as we are not interested in reasoning about the hidden states of agents, although these can be trivially included in the logic if required.

The formula $\bigcirc\phi$ is read as "ϕ holds at the next time step"; $\psi \,\mathcal{U}\, \phi$ is read as "ψ holds at least until ϕ becomes true, which must hold at the current or a future position"; $\psi \mathcal{R} \phi$ is read as "ϕ has to be true until and including the point where ψ first becomes true; if ψ never becomes true, ϕ must remain true forever"; $\Box\phi$ is read as "ϕ always holds"; and $\Diamond C$ is read as "C holds at some point in the future".

For the sake of brevity, we use $\bigcirc^k \phi$, to indicate k consecutive applications of $\bigcirc$, i.e., to state "ϕ holds after k steps". For an unbounded operator, such as $\mathcal{U}$, we denote with $\mathcal{U}^{\leq k}$ its bounded version; e.g., $\psi \,\mathcal{U}^{\leq k} \phi$ reads "ϕ holds within k time steps and ψ holds up to then." Note that $\psi \,\mathcal{U}^{\leq k} \phi$ is, in fact, a bounded formula, which can be expressed using Boolean operators and finitely many $\bigcirc$; specifically, $\psi \,\mathcal{U}^{\leq k} \phi = \bigvee_{i=0}^{k} ((\bigwedge_{j=0}^{i-1} \bigcirc^j \phi) \wedge \bigcirc^i \psi)$.

Next, the $\text{LTL}_{\mathbb{R}}$ satisfaction relation is defined as follows.

Definition 5 (Satisfaction)**.** The satisfaction relation $\models$ for a path $\pi \in \Pi_S$ of an MN-MAS S, time step t, and $\text{LTL}_{\mathbb{R}}$ formulae ϕ and ψ is defined as follows.

$$\begin{aligned}
&(\pi, t) \models \boldsymbol{c}^{\intercal} \cdot \boldsymbol{x} \leq d \text{ iff } \boldsymbol{c}^{\intercal} \cdot \pi_{\mathcal{X}}^{\langle t \rangle} \leq d;\\
&(\pi, t) \models \psi \vee \phi \quad \text{iff } (\pi, t) \models \psi \text{ or } (\pi, t) \models \phi;\\
&(\pi, t) \models \psi \wedge \phi \quad \text{iff } (\pi, t) \models \psi \text{ and } (\pi, t) \models \phi;\\
&(\pi, t) \models \neg\phi \quad \text{iff } (\pi, t) \not\models \phi;\\
&(\pi, t) \models \bigcirc^k \phi \quad \text{iff } (\pi, t+k) \models \phi;\\
&(\pi, t) \models \Box\phi \quad \text{iff } \forall i \geq 0,\ (\pi, t+i) \models \phi;\\
&(\pi, t) \models \Diamond\phi \quad \text{iff } \exists i \geq 0,\ (\pi, t+i) \models \phi;\\
&(\pi, t) \models \psi \,\mathcal{U}\, \phi \quad \text{iff } \exists i \geq 0,\ (\pi, t+i) \models \phi \wedge \\
&\qquad\qquad \textstyle\bigwedge_{j=0}^{i-1} (\pi, t+j) \models \psi;\\
&(\pi, t) \models \psi \mathcal{R} \phi \quad \text{iff } \forall i \geq 0,\ (\pi, t+i) \models \phi; \vee \\
&\quad \exists i \geq 0,\ (\pi, t+i) \models \psi \wedge \textstyle\bigwedge_{j=0}^{i} (\pi, t+j) \models \phi.
\end{aligned}$$

We say that a set of environment states $\mathcal{X}' \subseteq \mathcal{X}$ and a set of hidden states $\mathcal{H}' \subseteq \mathcal{H}$ satisfy ϕ, written $\mathcal{X}' \times \mathcal{H}' \models \phi$, if $(\pi, 0) \models \phi$ for all paths $\pi \in \Pi_S$ where $\pi_{\mathcal{X}}^{\langle 0 \rangle} \in \mathcal{X}'$ and $\pi_{\mathcal{H}}^{\langle 0 \rangle} \in \mathcal{H}'$ (i.e., for all paths whose initial states are in $\mathcal{X}'$ and $\mathcal{H}'$). We say that an MN-MAS S satisfies ϕ, written $S \models \phi$, if $\mathcal{X}^{\langle 0 \rangle} \times \mathcal{H}^{\langle 0 \rangle} \models \phi$, i.e., if every MN-MAS path that starts from an initial state of S satisfies ϕ.

We can now formulate the $\text{LTL}_{\mathbb{R}}$ *verification problem* for MN-MAS, *the problem we address in this paper.*

Definition 6 (Verification Problem)**.** Given an MN-MAS S and formula $\phi \in \text{LTL}_{\mathbb{R}}$, determine if $S \models \phi$.

When ϕ is bounded, we call the problem above the *Bounded Verification Problem* (*BVP*), and when ϕ is unbounded, we call it the *Unbounded Verification Problem* (UVP). If an instance $S \models \phi$ can be solved, we say that it is *solvable*.

As we discussed in the related work, the problem considered in Def. 6 is more general than previous works as it allows multiple

agents, arbitrary $\text{LTL}_{\mathbb{R}}$ formulae, and non-deterministic partially-observable environments.

3 MN-MAS' BOUNDED $\text{LTL}_{\mathbb{R}}$ VERIFICATION

Here, we introduce our approach for solving the bounded verification problem for MN-MAS. Particularly, In §3.1, we show that BVP is decidable. In §3.2, we provide two algorithms for solving BVP.

3.1 Decidability of Bounded $\text{LTL}_{\mathbb{R}}$ Verification

We start by proving the decidability of verifying atomic $\text{LTL}_{\mathbb{R}}$ propositions and $\bigcirc$ formulas in Lems. 1 and 2.

Lemma 1. *Verifying $S \models \phi = (\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d)$ is decidable.*

Proof. From Defs. 5 and 6 we know that $S \models \phi$ iff $\boldsymbol{c}^\intercal \cdot \mathcal{X}^{\langle 0 \rangle} \leq d$, where $\mathcal{X}^{\langle 0 \rangle}$ is the set of initial states of the environment. In other words, we want to check whether $\forall \boldsymbol{x} \in \mathcal{X}^{\langle 0 \rangle}\ (\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d)$, which can be solved using MILP [13] because it entails checking the unsatisfiability of $\exists \boldsymbol{x} \in \mathcal{X}^{\langle 0 \rangle}\ \neg(\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d)$. □

Lemma 2. *Verifying $S \models \phi = \bigcirc^t (\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d)$ is decidable.*

Proof. As we showed in the proof of Lem. 2, solving $S \models \phi$ for $t = 0$ amounts to verifying $\forall \boldsymbol{x} \in \mathcal{X}^{\langle 0 \rangle}\ (\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d)$. For $t \geq 1$, it amounts to verifying

$$\forall \boldsymbol{x} \in \mathcal{X}^{\langle t \rangle}\ (\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d), \tag{6}$$

where $\mathcal{X}^{\langle t \rangle}$ is recursively defined via

$$\mathcal{X}^{\langle t \rangle} = \mathcal{P}_{\mathcal{X}}^{\langle t \rangle}, \quad \mathcal{P}^{\langle t \rangle} = \{e(\boldsymbol{x}, \boldsymbol{h}) : (\boldsymbol{x}, \boldsymbol{h}) \in \mathcal{P}^{\langle t-1 \rangle}\}. \tag{7}$$

In Eq. (7), the number of free, pivot, and discrete variables defining $\mathcal{X}^{\langle t-1 \rangle}$ grows linearly with t. Specifically, if the defining constraints of the MN-MAS have $m_{\boldsymbol{x}}, m_{\boldsymbol{y}}$, and $m_{\boldsymbol{\delta}}$ free, pivot, and discrete variables, respectively, then the defining constraints of $\mathcal{X}^{\langle t \rangle}$ have at most $(t+1)m_{\boldsymbol{x}}$, $(t+1)m_{\boldsymbol{y}}$, and $(t+1)m_{\boldsymbol{\delta}}$ free, pivot, and discrete variables, respectively. Therefore verifying Eq. (6) is decidable, which implies verifying $S \models \bigcirc^t\ (\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d)$ is decidable. □

This readily gives us the following result on the decidability of verifying Boolean combinations of $\text{LTL}_{\mathbb{R}}$ formulae.

Lemma 3. *Given $\text{LTL}_{\mathbb{R}}$ formulae ϕ and ψ, such that $S \models \phi$ and $S \models \psi$ are decidable, $S \models \phi \wedge \psi, S \models \phi \vee \psi$ and $S \models \neg\phi$ are also all decidable.*

We can now prove that BVP is decidable.

Theorem 1. *Bounded $\text{LTL}_{\mathbb{R}}$ verification of MN-MAS is decidable.*

Proof. Bounded $\text{LTL}_{\mathbb{R}}$ formulae consist of atomic propositions alongside $\neg, \wedge, \vee$, and $\bigcirc$ operators. Using the duality of $\bigcirc$ and $\neg$ and the distributivity of $\bigcirc$ on $\wedge$ and $\vee$,[1] we can rewrite every bounded $\phi \in \text{LTL}_{\mathbb{R}}$ formula as a finite (linear in the length of ϕ) combination (using $\wedge, \vee$, and $\neg$) of formulae of the form $\bigcirc^i(\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d)$. It follows from Lems. 1 to 3 that verifying $S \models \phi$ is decidable. □

[1] Duality means $\neg\bigcirc\phi \equiv \bigcirc\neg\phi$, and distributivity means $\bigcirc(\phi \wedge \psi) \equiv \bigcirc\phi \wedge \bigcirc\psi$ and $\bigcirc(\phi \vee \psi) \equiv \bigcirc\phi \vee \bigcirc\psi$ for all $\phi, \psi \in \text{LTL}_{\mathbb{R}}$.

Algorithm 1: Bound Propagation through Time

Input : MN-MAS S, $\phi = \bigcirc^i(\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d) \in \text{LTL}_{\mathbb{R}}$
Output: `True/Unknown`

1 $\boldsymbol{\ell}^{\langle 0 \rangle}, \boldsymbol{u}^{\langle 0 \rangle} = \min_{\mathcal{X}^{\langle 0 \rangle} \times \mathcal{H}^{\langle 0 \rangle}}(\boldsymbol{x}), \max_{\mathcal{X}^{\langle 0 \rangle} \times \mathcal{H}^{\langle 0 \rangle}}(\boldsymbol{x})$
2 **for** $t \leftarrow 1, \ldots, i$ **do**
3 $\quad \boldsymbol{\ell}^{\langle t \rangle}, \boldsymbol{u}^{\langle t \rangle} = \mathsf{BP}(e(\cdot), (\boldsymbol{\ell}^{\langle t-1 \rangle}, \boldsymbol{u}^{\langle t-1 \rangle}))$
4 **if** $(\boldsymbol{c}^\intercal \cdot \boldsymbol{\ell}_{\mathcal{X}}^{\langle i \rangle} \leq d) \wedge (\boldsymbol{c}^\intercal \cdot \boldsymbol{u}_{\mathcal{X}}^{\langle i \rangle} \leq d)$ **then**
5 $\quad$ **return** `True`
6 **return** `Unknown`

3.2 Solving Bounded $\text{LTL}_{\mathbb{R}}$ Verification

Above, we outlined the procedure for solving BVP in §3.1 at a high level. We now provide two methods for solving the BVP algorithmically. The first is *Bound Propagation through Time* (*BPT*), which is a sound but incomplete (i.e., overapproximated) technique for solving BVP. Next, we introduce *Recursive MILP* (*RMILP*), which allows encoding the BVP as an MILP problem, thereby providing a sound and complete (i.e., exact) solution to BVP.

3.2.1 Bound Propagation through Time (BPT). Bound propagation [19, 52] is commonly used for verifying the robustness of neural networks. To use it for memoryful models, such as RNNs, as well as systems with temporal dependencies, such as MN-MAS, a more refined approach is required. We introduce *BPT*, an extension of bound propagation which allows us to propagate the network, memory, and system bounds through time.

The high-level procedure behind BPT is outlined in Alg. 1. First, we note that every bounded formulae in $\text{LTL}_{\mathbb{R}}$ is a finite composition (using $\wedge, \vee, \neg$) of formulae of the form $\bigcirc^i(\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d)$, and so, w.l.o.g. we illustrate BPT for checking the latter formulae. Alg. 1 starts with the bounds $\boldsymbol{\ell}^{\langle 0 \rangle}, \boldsymbol{u}^{\langle 0 \rangle}$ of the joint initial system and hidden state. It then uses a bound propagation method `BP` (such as interval or linear bound propagation) to propagate $\boldsymbol{\ell}^{\langle 1 \rangle}, \boldsymbol{u}^{\langle 1 \rangle}$ through the combined system evolution function $e(\cdot)$ to compute bounds $\boldsymbol{\ell}^{\langle 1 \rangle}, \boldsymbol{u}^{\langle 1 \rangle}$ on the joint system and hidden state at the next time step (line 3). This is repeated iteratively to obtain bounds for any finite time step. Finally, the algorithm checks whether the final bounds $\boldsymbol{\ell}^{\langle i \rangle}, \boldsymbol{u}^{\langle i \rangle}$ satisfy the given constraints (line 4).

Using bound propagation we have that, at each step t, $[\boldsymbol{\ell}^{\langle t \rangle}, \boldsymbol{u}^{\langle t \rangle}] \supseteq e([\boldsymbol{\ell}^{\langle t-1 \rangle}, \boldsymbol{u}^{\langle t-1 \rangle}])$, where $[\boldsymbol{\ell}^{\langle t \rangle}, \boldsymbol{u}^{\langle t \rangle}]$ is the hyper-box defined by the bounds $\boldsymbol{\ell}^{\langle t \rangle}$ and $\boldsymbol{u}^{\langle t \rangle}$. In other words, BPT computes an over-approximation of the reachable states; thus, the algorithm returns `Unknown` if the BPT bounds do not satisfy the constraints.

Let us demonstrate BPT using an example. We use an RNN for the sake of simplicity rather than the full system evolution function.

Example 1. *Consider the formula $\phi_i = \bigcirc^i(z \leq 2) \in \text{LTL}_{\mathbb{R}}$, where z is the output of the RNN in Fig. 1, with its input being $\boldsymbol{x} \in [0, 1]^2$. We want to verify ϕ_2 and ϕ_3 are satisfied.*

Let us check whether ϕ_2 and ϕ_3 are satisfied using Alg. 1. At $t = 0$, we have that $[\boldsymbol{\ell}^{\langle 0 \rangle}, \boldsymbol{u}^{\langle 0 \rangle}] = [0, 1]^2 \times [0, 0]$, where $[0, 1]^2$ indicates the bounds on $\boldsymbol{x}^{\langle 0 \rangle}$ and $[0, 0]$ indicates the bounds on $z^{\langle 0 \rangle}$ (line 1). To obtain bounds on $z^{\langle 1 \rangle}$, we need to propagate the bounds through the RNN, which gives us $y_1^{\langle 1 \rangle} = \text{ReLU}(-x_1^{\langle 0 \rangle} +$

Algorithm 2: Recursive MILP

Input : MN-MAS S, $\phi = \bigcirc^i(\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d) \in \text{LTL}_\mathbb{R}$
Output: True/False

1 $(\mathcal{V}, C)$ = buildConstraints(i)
2 **return** $\text{SAT}_{\text{MILP}}(\forall \boldsymbol{x}^{\langle i \rangle}.\ \boldsymbol{c}^\intercal \cdot \boldsymbol{x}^{\langle i \rangle} \leq d, \mathcal{V}, C)$
3
4 **Procedure** buildConstraints(t)
5 $\mathcal{V}' = \{\boldsymbol{x}^{\langle t \rangle}, \boldsymbol{h}^{\langle t \rangle}\}$
6 **if** $t = 0$ **then**
7 **return** $(\mathcal{V}', \{\boldsymbol{x}^{\langle 0 \rangle} \in \mathcal{X}^{\langle 0 \rangle}, \boldsymbol{h}^{\langle 0 \rangle} \in \mathcal{H}^{\langle 0 \rangle}\})$
8 **else**
9 $(\mathcal{V}, C)$ = buildConstraints($t-1$)
10 $C' = \{(\boldsymbol{x}^{\langle t \rangle}, \boldsymbol{h}^{\langle t \rangle}) = e(\boldsymbol{x}^{\langle t-1 \rangle}, \boldsymbol{h}^{\langle t-1 \rangle})\}$
11 **return** $(\mathcal{V} \cup \mathcal{V}', C \cup C')$

$x_2^{\langle 0 \rangle}) \in [0,1]$. Similarly, we have $y_2^{\langle 1 \rangle} \in [0,2]$ and $z^{\langle 1 \rangle} \in [0,1]$; hence, $[\boldsymbol{t}^{\langle 1 \rangle}, \boldsymbol{u}^{\langle 1 \rangle}] = [0,1]^3$. Repeating this, we have $[\boldsymbol{t}^{\langle 2 \rangle}, \boldsymbol{u}^{\langle 2 \rangle}] = [0,1]^2 \times [0,2]$ and $[\boldsymbol{t}^{\langle 3 \rangle}, \boldsymbol{u}^{\langle 3 \rangle}] = [0,1]^2 \times [0,3]$; thus, using BPT, we could verify that ϕ_2 is satisfied while ϕ_3 is not (since $3 \not\leq 2$).

3.2.2 Recursive Mixed Integer Linear Programming (RMILP). At each step, BPT overapproximates the set of attainable values for each variable. Hence, it does not provide a complete method despite its efficiency. In RMILP, instead of propagating the bounds for each variable, we construct an MILP equation describing the set of attainable values *exactly* in terms of the constraints and variables of previous time steps, in a *recursive* manner. Similarly to BPT, it suffices to verify formulae of the form $\bigcirc^i(\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d)$.

As outlined in Alg. 2, we want to verify that for all $\boldsymbol{x} \in \mathcal{P}_{\mathcal{X}}^{\langle i \rangle}$, $\boldsymbol{c}^\intercal \cdot \boldsymbol{x} \leq d$, where $\mathcal{P}_{\mathcal{X}}^{\langle i \rangle}$ is the joint system and hidden state at time i. This is verified using MILP by first finding the $\boldsymbol{x}^* \in \mathcal{P}_{\mathcal{X}}^{\langle i \rangle}$ that maximises $\boldsymbol{c}^\intercal \cdot \boldsymbol{x}$ and then checking whether $\boldsymbol{c}^\intercal \cdot \boldsymbol{x}^* \leq d$. However, we first need to build the constraints that define $\mathcal{P}_{\mathcal{X}}^{\langle i \rangle}$. We build them recursively, as described in the subprocedure buildConstraints of Alg. 2 (lines 4-11). For any given step t, it returns the set of MILP variables $\mathcal{V}$ and constraints C that encode the reachable set $\mathcal{P}_{\mathcal{X}}^{\langle t \rangle}$. For simplicity, we denote the MILP variables using $\boldsymbol{x}^{\langle t \rangle}$ and $\boldsymbol{h}^{\langle t \rangle}$.

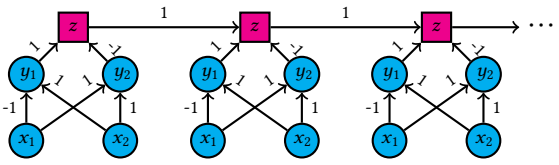


Figure 1: A simple RNN, consisting of a feed-forward first layer $\boldsymbol{y} = \text{ReLU}(\omega_1 \boldsymbol{x})$, which maps the input $\boldsymbol{x} = \binom{x_1}{x_2}$ to $\boldsymbol{y} = \binom{y_1}{y_2}$ via left multiplication by $\omega_1 = \left(\begin{smallmatrix} -1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$. The last layer is recurrent and is defined as $z^{\langle t \rangle} = \text{ReLU}(\omega_2 \boldsymbol{y} + z^{\langle t-1 \rangle})$, where $\omega_2 = (1\ -1)$ and z is the output variable with $z^{\langle 0 \rangle} = 0$. The superscript $^{\langle t \rangle}$ denotes indicates the time step and is omitted when there is no ambiguity (e.g., here, we have omitted it for x_1, x_2, y_1 and y_2, as they are not used in other time steps).

Returning to Ex. 1, recall that we could not check whether ϕ_3 is satisfied using BPT. Let us instead use RMILP to check ϕ_3. We have that ϕ_3 holds iff $z^{\langle 3 \rangle} \leq 3$. Since $z^{\langle 3 \rangle} = \text{ReLU}(\omega_2\, \text{ReLU}(\omega_1 \boldsymbol{x}^{\langle 2 \rangle}) + z^{\langle 2 \rangle})$, $z^{\langle 3 \rangle}$ is defined in terms of $z^{\langle 2 \rangle}$ and the input $\boldsymbol{x}^{\langle 2 \rangle}$. In a similar manner, we can write $z^{\langle 2 \rangle}$ in terms of $z^{\langle 1 \rangle}$ and $\boldsymbol{x}^{\langle 1 \rangle}$, and $z^{\langle 2 \rangle}$ in terms of $z^{\langle 0 \rangle}$ and $\boldsymbol{x}^{\langle 0 \rangle}$. This results in PWL constraints defining $z^{\langle 3 \rangle}$ in terms of $\boldsymbol{x}^{\langle 0 \rangle}, \boldsymbol{x}^{\langle 1 \rangle}, \boldsymbol{x}^{\langle 2 \rangle}$ and $z^{\langle 0 \rangle}$,[2] allowing us to check whether $z^{\langle 3 \rangle} \leq 3$, using MILP. As we show next, RMILP allows us to prove that ϕ_3, in fact, holds.

3.2.3 Adaptive Splitting. One limitation of RMILP is its high computational complexity, as we prove in Thm. 1, leading to an exponentially large search space for MILP solvers. We use adaptive splitting [25, 35] to mitigate this problem. In adaptive splitting, we use BPT to provide the MILP solver with *a priori* stricter bounds for the continuous variables, thereby reducing the number of branchings required to solve the MILP problem; i.e., the BPT bounds allow us to restrict the values that the discrete variables in Eq. (1) attain.

In Ex. 1, we showed that $x_1 + x_2 \in [0,2]$ using BPT (for all $t \in \mathbb{N}$, hence, dropping the suffix $^{\langle t \rangle}$). So, $y_2 = \text{ReLU}(x_1 + x_2)$ simplifies to $y_2 = x_1 + x_2$, allowing us to reduce the number of PWL constraints. Thus, adaptive splitting simplifies the equation for z to $z^{\langle t \rangle} = \text{ReLU}(\text{ReLU}(-x_1 + x_2) - (x_1 + x_2) + z^{\langle t-1 \rangle})$. We can solve the resulting constraints as follows: for $t = 1$, if $x_2 \geq x_1$, then $z^{\langle 1 \rangle} = \text{ReLU}(-2x_1) = 0$, and if $x_2 < x_1$, then $z^{\langle 1 \rangle} = \text{ReLU}(-x_1 - x_2) = 0$ for all $x_1, x_2 \in [0,1]$. Repeating this for the following steps, we obtain that $z^{\langle t \rangle} = 0$ for all t. Thus, not only ϕ_2, but also ϕ_3 holds.

In summary, the approach uses RMILP for verification, but first applies adaptive splitting to reduce any $\text{ReLU}(y)$ expression into either y or 0 whenever BPT can show that $y \geq 0$ or $y \leq 0$, respectively. Similar reduction rules can be defined for other PWL activation functions, such as LeakyReLU, HardShrink and SoftShrink.

4 MN-MAS' UNBOUNDED $\text{LTL}_\mathbb{R}$ VERIFICATION

In this section, we address the UVP for MN-MAS and show how to solve it for a wide range of unbounded $\text{LTL}_\mathbb{R}$ specifications using **(1)** *Bounded-Path Verification* (BPMC), **(2)** *lasso* search, and **(3)** *inductive invariant* synthesis. Before presenting our decidability results, we note that the UVP is undecidable in general (the proof is provided in the full version on arXiv).

Theorem 2. *$\text{LTL}_\mathbb{R}$ verification of MN-MAS is undecidable.*

4.1 Bounded-Path Model Checking

Bounded-Path Model Checking (*BPMC*) allows us to partially solve the UVP by restricting unbounded formulae to finite paths. It is a straightforward, yet effective, technique to solve the UVP as we formally state in Thm. 3.

Theorem 3. *Given formulae $\phi, \psi \in \text{LTL}_\mathbb{R}$, such that $S \models \phi$ and $S \models \psi$ are decidable, we can solve the instances of the UVP checkmarked in Tbl. 1, using BPMC.*

Proof. To verify the specifications in Tbl. 1 using BPMC, it suffices to write their bounded versions using $\bigcirc$ and Boolean operators. To this end, we observe that for a given $k \in \mathbb{N}$, we have that $\Diamond^{\leq k}\phi \equiv$

[2] We note that $\text{ReLU}(x) = \max\{0, x\}$ can be written as a PWL constraint using the "big-M" method [20] as follows. $y = \text{ReLU}(x) \iff (y \geq x) \wedge (y \geq 0) \wedge (y = \delta x)$.

Algorithm 3: Lasso-assisted BPMC for $\Diamond\phi$

Input : MN-MAS S, $\phi \in \text{LTL}_{\mathbb{R}}$, and search depth k
Output: `False/Unknown`

1 $\mathcal{P}^{\langle 0\rangle} = \{(\boldsymbol{x}, \boldsymbol{h}) \in \mathcal{X}^{\langle 0\rangle} \times \mathcal{H}^{\langle 0\rangle} : \boldsymbol{x} \not\models \phi\}$
2 **for** $t \leftarrow 1, \ldots, k$ **do**
3 $\mathcal{P}^{\langle t\rangle} = \{(\boldsymbol{x}, \boldsymbol{h}) \in e(\mathcal{P}^{\langle t-1\rangle}) : \boldsymbol{x} \not\models \phi\}$
4 **for** $t' \leftarrow 0, \ldots, t-1$ **do**
5 **if** $\mathcal{P}^{\langle t\rangle} \cap \mathcal{P}^{\langle t'\rangle} \neq \emptyset$ **then**
6 **return** `False` ▷ *Lasso is found*
7 **return** `Unknown`

$\bigvee_{i=1}^{k} \bigcirc^i \phi$, $\square^{\leq k}\phi \equiv \bigwedge_{i=1}^{k} \bigcirc^i \phi$, $\psi \,\mathcal{U}^{\leq k} \phi \equiv \bigvee_{i=0}^{k}((\bigwedge_{j=0}^{i-1} \bigcirc^j \psi) \wedge \bigcirc^i \phi)$, and $\psi \mathcal{R}^{\leq k}\phi \equiv \bigvee_{i=0}^{k-1}((\bigwedge_{j=0}^{i} \bigcirc^j \psi) \wedge \bigcirc^i \phi)$. □

4.2 Lasso Search

Bounded model checking for solving the UVP can be further enhanced via lasso search [9]. For an MN-MAS S and formula $\phi \in \text{LTL}_{\mathbb{R}}$, the idea behind the lasso search is to find a path $\rho \in \Pi_S$, such that for some $0 \leq t' < t$, we have that $\rho^{\langle 1\rangle}, \ldots, \rho^{\langle t\rangle} \not\models \phi$ and $\rho^{\langle t'\rangle} = \rho^{\langle t\rangle}$. Such a path is called a *lasso*, and $\rho^{\langle 1,\ldots,t\rangle}$ is witness for $S \not\models \Diamond\phi$. This is summarised in Alg. 3.

We observe that Alg. 3 is complete. To prove this, we note that it returns `False` only if $\mathcal{P}^{\langle t\rangle} \cap \mathcal{P}^{\langle t'\rangle} \neq \emptyset$ for some $0 \leq t' < t \leq k$. This means that there exist $(\boldsymbol{x}, \boldsymbol{h}) \in \mathcal{P}^{\langle t\rangle} \cap \mathcal{P}^{\langle t'\rangle}, \boldsymbol{x}^{\langle 0\rangle}, \ldots, \boldsymbol{x}^{\langle t\rangle} \in \mathcal{X}$, and $\boldsymbol{h}^{\langle 0\rangle}, \ldots, \boldsymbol{h}^{\langle t\rangle} \in \mathcal{H}$, such that $\boldsymbol{x}^{\langle t'\rangle} = \boldsymbol{x}^{\langle t\rangle} = \boldsymbol{x}$, $\boldsymbol{h}^{\langle t'\rangle} = \boldsymbol{h}^{\langle t\rangle} = \boldsymbol{h}$, and $(\boldsymbol{x}^{\langle i\rangle}, \boldsymbol{h}^{\langle i\rangle}) = \tau(\boldsymbol{x}^{\langle i-1\rangle}, r(o(\boldsymbol{x}^{\langle i-1\rangle}), \boldsymbol{h}^{\langle i-1\rangle}))$ for $i = 1, \ldots, t$. This implies that $\boldsymbol{x}^{\langle 0\rangle}, \ldots, \boldsymbol{x}^{\langle t\rangle}$ is a lasso that entirely does not satisfy ϕ; thus, $S \not\models \Diamond\phi$, and Alg. 3 is complete. Thm. 4 generalises this reasoning.

Theorem 4. *Given $\text{LTL}_{\mathbb{R}}$ formulae ϕ and ψ, such that $S \models \phi$ and $S \models \psi$ are solvable (e.g., using BPMC, lasso, or invariants), then, using lasso search, we can solve the instances of UVP checkmarked in Tbl. 1.*

Proof. We have already shown in §4.2 that the lasso search allows verifying $S \not\models \Diamond\phi$. To check $S \not\models \psi \,\mathcal{U} \phi$, we first search for a lasso for $S \not\models \Diamond\phi$. If a lasso is not found, we cannot prove the specification. Otherwise, we have found a lasso of length k proving that $S \not\models \Diamond\phi$. This means that S will never satisfy ϕ; hence $S \not\models \psi \,\mathcal{U} \phi$. To check $S \not\models \psi\mathcal{R}\phi$, we first search for a lasso for $S \not\models \Diamond\psi$. If a lasso is not found, we cannot prove the specification. Otherwise, we have found a lasso of length k proving that $S \not\models \Diamond\psi$. This means that S will never satisfy ψ. Now, for a fixed bound $i \in \mathbb{N}$, we check whether $S \not\models \bigcirc^{\leq i}\phi$. If $S \not\models \bigcirc^{\leq i}\phi$, we have that $S \not\models \psi\mathcal{R}\phi$. □

Remark 3. Alg. 3 can only be used alongside MILP because BPT overapproximates $\mathcal{P}^{\langle t\rangle}$, and therefore the algorithm may wrongly return `False` in line 5.

4.3 Inductive Invariant Synthesis

Despite the versatility of the lasso approach, it cannot be used to verify specifications of the form $S \models \square\phi$ in non-deterministic environments. This is because to verify $S \models \square\phi$, we need to ensure that $\pi \models \square\phi$ for *all* valid paths $\pi \in \Pi_S$, whilst lassos only allow us to search for the *existence* of a path that satisfies a given specification. This leads us to use inductive invariants [9]. For an MN-MAS S, an *inductive invariant*, or *invariant* for short, is a set $\mathcal{I} \subseteq \mathcal{X} \times \mathcal{H}$, such that $\mathcal{X}^{\langle 0\rangle} \times \mathcal{H}^{\langle 0\rangle} \subseteq \mathcal{I}$ and $e(\mathcal{I}) \subseteq \mathcal{I}$. If there exists an inductive invariant $\mathcal{I}$ for S such that $\mathcal{I} \models \phi$ for a formula $\phi \in \text{LTL}_{\mathbb{R}}$, it follows that $S \models \square\phi$ since all initial and future states of S satisfy ϕ. This allows us to verify a wider range of unbounded formulae, as shown in Thm. 5 and outlined in Tbl. 1

Table 1: Specifications that some (not all) instances of that form *may* be verified using each method. A cross mark x means that an approach (e.g., Lasso) *cannot* be used for instances of that form (e.g., $\psi\mathcal{R}\phi$), and a checkmark ✓ means it *can*, however, it may return the correct answer or `Unknown`.

Method	Satisfaction	$\psi\mathcal{R}\phi$	$\square\,\phi$	$\psi\,\mathcal{U}\phi$	$\Diamond\,\phi$
BPMC	$S \models$	✓	x	✓	✓
	$S \not\models$	✓	✓	✓	x
Lasso	$S \models$	x	x	x	x
	$S \not\models$	✓	x	✓	✓
Invariant	$S \models$	✓	✓	x	x
	$S \not\models$	x	x	✓	✓

Algorithm 4: Inductive invariants for $\square\phi$

Input : MN-MAS S, $\phi \in \text{LTL}_{\mathbb{R}}$, and search depth k
Output: `True/False/Unknown`

1 $\mathcal{I}^{\langle 0\rangle} = \mathcal{X}^{\langle 0\rangle} \times \mathcal{H}^{\langle 0\rangle}$
2 $\mathcal{M}^{\langle 0\rangle} = \mathcal{X} \times \mathcal{H}$
3 **for** $t \leftarrow 1, \ldots, k$ **do**
4 $\mathcal{I}^{\langle t\rangle} = e(\mathcal{I}^{\langle t-1\rangle})$
5 $\mathcal{M}^{\langle t\rangle} = e(\mathcal{M}^{\langle t-1\rangle})$
6 **if** $\mathcal{I}_{\mathcal{X}}^{\langle t\rangle} \not\models \phi$ **then**
7 **return** `False`
8 **else if** $\mathcal{M}_{\mathcal{X}}^{\langle t-1\rangle} \models \phi$ **then**
9 **return** `True` ▷ *Maximal invariant*
10 **else if** $\mathcal{I}^{\langle t\rangle} \subseteq \bigcup_{i=0}^{t-1} \mathcal{I}^{\langle i\rangle}$ **then**
11 **return** `True` ▷ *Minimal invariant*
12 **return** `Unknown`

Theorem 5. *Given an MN-MAS S and $\text{LTL}_{\mathbb{R}}$ formulae ϕ and ψ, such that $S \models \phi$ and $S \models \psi$ are solvable, then, using inductive invariants, we can solve the instances of UVP checkmarked in Tbl. 1.*

Proof. We show that inductive invariants can be used to prove $S \models \square\phi$ specifications. Assume that $\mathcal{I}$ is an inductive invariant for S. If $\mathcal{I}_{\mathcal{X}} \models \phi$, then by definition, we have that the initial states and all future states resulting from finite evolution of S satisfy ϕ. Hence, an invariant $\mathcal{I}$ for S, such that $\mathcal{I}_{\mathcal{X}} \models \phi$, implies $S \models \square\phi$. To prove that inductive invariants can be used to verify $S \not\models \Diamond\phi$ and $S \not\models \psi \,\mathcal{U} \phi$, note that we can check for $S \models \square\neg\phi$ using inductive invariants. If we can verify that $S \models \square\neg\phi$, then $S \not\models \Diamond\phi$ and $S \not\models \psi \,\mathcal{U} \phi$. A similar reasoning applies to $S \models \psi\mathcal{R}\phi$. We can check $S \models \square\phi$ using invariants. If we can verify that $S \models \square\phi$, then $S \models \psi\mathcal{R}\phi$. □

Even though inductive invariants theoretically allow us to verify a wider range of unbounded formulae, finding such invariants in practice is not straightforward. We provide two techniques for this purpose in Alg. 4 by searching for *maximal* and *minimal* inductive invariants. For the minimal invariant, we start from $\mathcal{I} = \mathcal{I}^{\langle 0\rangle} = \mathcal{X}^{\langle 0\rangle} \times \mathcal{H}^{\langle 0\rangle}$ (line 1), which must be contained by any invariant by definition. We then keep expanding $\mathcal{I}$ by $\mathcal{I}^{\langle t\rangle} = e^{(t)}(\mathcal{I}^{\langle 0\rangle})$, until we find an invariant (line 10), in which case $\mathcal{I} = \bigcup_{i=0}^{t-1} \mathcal{I}^{\langle i\rangle}$ is the minimal invariant, or we exceed a given maximum number of iterations k. For the maximal invariant, we start with the trivial largest invariant, i.e., $\mathcal{M}^{\langle 0\rangle} = \mathcal{X} \times \mathcal{H}$. If $\mathcal{M}^{\langle 0\rangle}_{\mathcal{X}} = \mathcal{X} \models \phi$, it follows that $S \models \Box\phi$. Otherwise, the algorithm iteratively computes $\mathcal{M}^{\langle t\rangle} = e^{(t)}(\mathcal{M}^{\langle 0\rangle})$ until $\mathcal{M}^{\langle t\rangle}_{\mathcal{X}} \models \phi$ (line 8), in which case $\mathcal{M} = \bigcup_{i=0}^{t-1} \mathcal{I}^{\langle i\rangle} \cup \mathcal{M}^{\langle t-1\rangle}$ is the invariant, or it exceeds a given maximum time step k.

To prove that $\mathcal{I}$ and $\mathcal{M}$ (if found) in Alg. 4 are invariant, we observe that by construction, $\mathcal{X}^{\langle 0\rangle} \times \mathcal{H}^{\langle 0\rangle} \subseteq \mathcal{I}, \mathcal{M}$. For $\mathcal{I}$, we have that $e(\mathcal{I}) = e(\bigcup_{i=0}^{t-1} \mathcal{I}^{\langle i\rangle}) = \bigcup_{i=0}^{t-1} e(\mathcal{I}^{\langle i\rangle}) = \bigcup_{i=0}^{t-1} \mathcal{I}^{\langle i+1\rangle} \subseteq \bigcup_{i=0}^{t-1} \mathcal{I}^{\langle i\rangle} = \mathcal{I}$ by line 10; therefore, $\mathcal{I}$ is an invariant. For $\mathcal{M}$, we have that $\mathcal{I}^{\langle t\rangle}, \mathcal{M}^{\langle t\rangle} \subseteq \mathcal{M}^{\langle t-1\rangle} = e^{t-1}(\mathcal{M}^{\langle 0\rangle})$ because $\mathcal{I}^{\langle t\rangle} = e^{t-1}(\mathcal{I}^{\langle 1\rangle})$ and $\mathcal{M}^{\langle t\rangle} = e^{t-1}(\mathcal{M}^{\langle 1\rangle})$, and by construction, it follows that $\mathcal{I}^{\langle 1\rangle}, \mathcal{M}^{\langle 1\rangle} \subseteq \mathcal{M}^{\langle 0\rangle}$. Hence, $e(\mathcal{M}) = e(\bigcup_{i=0}^{t-1} \mathcal{I}^{\langle i\rangle} \cup \mathcal{M}^{\langle t-1\rangle}) = \bigcup_{i=0}^{t-1} e(\mathcal{I}^{\langle i\rangle}) \cup e(\mathcal{M}^{\langle t-1\rangle}) = \bigcup_{i=0}^{t-1} \mathcal{I}^{\langle i+1\rangle} \cup \mathcal{M}^{\langle t\rangle} \subseteq \bigcup_{i=0}^{t-1} \mathcal{I}^{\langle i\rangle} \cup \mathcal{M}^{\langle t-1\rangle} = \mathcal{M}$; thus, $\mathcal{M}$ is an invariant.

Remark 4. Alg. 4 works with both RMILP and BPT. However, since BPT is incomplete, we have to return `Unknown` instead of `False` in line 7 when using BPT (because ϕ may be violated by a spurious states in $\mathcal{I}^{\langle t\rangle}_{\mathcal{X}}$.)

5 EXPERIMENTAL EVALUATIONS

We evaluate[3] our approach by verifying unbounded and bounded $\text{LTL}_{\mathbb{R}}$ specifications for the Cart Pole, Pendulum, and Lunar Lander environments from the Gymnasium library [49], and the Simple Push environment from the Petting Zoo library [46]. For each environment, we consider memoryful RNN policies of varying complexity. We use R_i and $R_{2\times i}$ to denote neural policies with one and two recurrent layers of size i, respectively. These recurrent layers are followed by single linear layers matching the agents' action spaces. We denote with S_i and $S_{2\times i}$ the corresponding MN-MAS.

For MN-MAS with discrete action spaces, the policies are trained using recurrent Q-learning (DQN) [24, 37], while for continuous action spaces, we use Proximal Policy Optimisation (PPO) [42]. The environments' transition functions are non-linear and involve trigonometric functions; hence, we use PWL approximation of these [1, 14] to make them linearly definable. In all experiments, our implementation first attempts to verify using BPT; if unsuccessful, it uses RMILP with adaptive splitting. We have set a timeout of 10,000 seconds. In all tables, "–" signifies timeout.

[3]For all experiments, training and verification were performed on a workstation with a 16-core AMD 5955WX CPU, 256GB of DDR4 RAM, and an NVIDIA RTX 4090 GPU. The verification algorithms are implemented in Python using the NumPy library [23] and Gurobi optimiser [22]. The source code will be released on `github.com/mehini/Vern`.

Table 2: Runtimes (seconds) for Alg. 4, with a maximum search depth of 9, to verify $S \models \phi_\epsilon$ for different ϵ. Green, white, and red cells, denote that Alg. 4 returns `True`, `Unknown`, and `False`, respectively. All specifications are verified using RMILP. Superscripts † and ‡ indicate that minimal and maximal invariants are found, respectively. Subscripts denote the search depth where Alg. 4 has terminated.

	$\epsilon = \pi/40$	$\epsilon = \pi/30$	$\epsilon = \pi/20$	$\epsilon = \pi/10$
$S_{16} \models \phi_\epsilon$	2.7_2	2.3_1	2.0_1	2.3_1
$S_{32} \models \phi_\epsilon$	$41.3^{\dagger}_{3}$	$-_9$	7571.2_8	$-_9$
$S_{2\times16} \models \phi_\epsilon$	$9.8^{\ddagger}_{1}$	$5.3^{\ddagger}_{1}$	$9.3^{\ddagger}_{1}$	$11.7^{\dagger}_{2}$

Table 3: Runtimes (seconds) of Alg. 3 (`False` and `Unknown` instances) and BPMC (`True` instances), with a maximum search depth of 20, for verifying $S \models \psi_\epsilon$ for different ϵ. Subscripts denote the time steps where violations are found in `False` instances and the search depths k of Alg. 3 in other instances. The colour code is as in Tbl. 2. All instances are verified by first using BPT and then RMILP; however, BPT alone could not verify any of the instances. For `Unknown` instances, the table shows Alg. 3's time. Colour codes are the same as Tbl. 2.

	$\epsilon = 0.15$	$\epsilon = 0.10$	$\epsilon = 0.05$	$\epsilon = 0.01$
$S_{10} \models \psi_\epsilon$	60_{15}	32_{13}	86_{19}	183_{20}
$S_{20} \models \psi_\epsilon$	2156_{19}	6903_{20}	5314_{20}	6012_{20}
$S_{2\times10} \models \psi_\epsilon$	8626_{18}	$-_{20}$	$-_{20}$	$-_{20}$

5.1 Cart Pole (Verifying Always)

In the Cart Pole environment [8], a pole is attached by an unactuated joint to a cart that is moving on a frictionless track. The pole is placed upright on the cart, and the goal is to balance the pole by applying forces to the cart from the left and right. At each time step, the agent has access to the cart's position $x \in [-4.8, 4.8]$ and the pole's angle $\theta \in [-2\pi/15, 2\pi/15]$ and can choose between pushing the cart to the left or right. We want to verify that the pole always remains balanced; thus, we consider unbounded specifications $\phi_\epsilon = \Box((\theta \leq \epsilon) \wedge (|x| \leq 1))$, for given angle bounds ϵ, when starting from the centre of the plane $x = 0$ while the pole is vertical, i.e., $\theta = \pi/2$. We trained three policies R_{16}, R_{32}, and $R_{2\times16}$, and used Alg. 4 to check whether their corresponding environments S_{16}, S_{32}, and $S_{2\times16}$ satisfy ϕ_ϵ for different ϵ. As reported in Tbl. 2, the $R_{2\times16}$ policy is guaranteed to keep the pole balanced indefinitely, and instances are easily verified, while R_{16} does not satisfy any of the constraints and the S_{16} environment is easily falsified. On the other hand, verifying the S_{32} environment has proven to be more challenging.

5.2 Pendulum (Verifying Eventually)

A pendulum is attached to a fixed joint at one end, while the other end is free. The pendulum starts in an uncertain position, and the goal is to swing it to an upright vertical position by applying torque on the free end. At each step, the agent has access to the free end's horizontal and vertical positions (x, y) and can apply a torque $a \in [-2, 2]$.

We verify that, for all possible system evolutions, the pole eventually reaches an upright position $y \geq 1 - \epsilon$ for different values of $\epsilon \in \{0.15, 0.1, 0.05, 0.01\}$, expressed by the unbounded eventually property $\psi_\epsilon = \Diamond(y \geq 1 - \epsilon)$. Again, we consider three RNNs with varying numbers of layers and nodes. As summarised in Tbl. 3, we see that R_{10} has the worst performance among the three policies.

5.3 Lunar Lander (Bounded Verification)

The Lunar Lander environment [33] consists of an agent whose goal is to land on a landing pad between two flags in a 2D representation of the moon The environment is non-deterministic due to wind. The action space $\mathcal{A} = \{\cup, \uparrow, \cup, \circ\}$ is discrete, consisting of "fire right orientation engine", "fire main engine", "fire left orientation engine", and "do nothing". The observation space $\mathcal{O} \subset \mathbb{R}^5$ consists of the horizontal and vertical coordinates of the lander $x, y \in [0, 1]$, its angle $\theta \in [-\pi, \pi]$, and two Boolean variables $l_1, l_2 \in \{0, 1\}$ that indicate if each leg is in contact with the ground. The agent is rewarded for landing on the landing pad and is penalised for crashing, moving out of the screen, and using the engines.

Here, we trained an RNN policy $R_{2\times32}$. To compare our approach against the existing methods, we verified $S_{2\times32} \models \phi$ using our proposed approach (RMILP combined with BPT and adaptive splitting) as well as pure MILP-based approaches of [4] and [28]. In fact, our implementation of [28] is the same as pure RMILP, and thus, this also serves as an ablation study on how BP and adaptive splitting speed up the verification.

As we see in Tbl. 4, BPT allows us to verify some instances in the early time steps almost instantly (the first rows of Tbl. 4c); however, it fails to provide a definite outcome for larger time bounds. Adaptive splitting significantly speeds up the verification and in larger time steps provides a speed up of around an order of magnitude compared to the pure RMILP, which is also the SoA [28] runtime.

5.4 Simple Spread (Multi-Agent)

The Simple Spread environment [46] consists of n agents and n landmarks. The goal is for the agents to cover all landmarks while avoiding collisions. All agents have the same observation space, $\mathcal{O} \subseteq \mathbb{R}^{4n}$, consisting of all agents and landmarks' 2D positions, and action space $\mathcal{A} = \{\leftarrow, \uparrow, \rightarrow, \downarrow, \circ\}$, consisting of "move left", "move up", "move right", "move down", or "do nothing" actions.

We consider two environments with different numbers of agents $n = 2, 4$ and two composite (bounded) $\text{LTL}_\mathbb{R}$ formulas: we verify whether the first agent, p_1, eventually reaches and remains in one of the landmarks (a stability property), and whether it always eventually reaches a landmark (a liveness property); i.e., we verify

$$S_n \models \xi_t = \Box^{\leq t}\Diamond^{\leq t}\Big(\bigvee_{i=1}^{n} \|p_1 - \ell_i\|_1 \leq 0.1\Big),$$

$$S_n \models \chi_t = \Diamond^{\leq t}\Box^{\leq t}\Big(\bigvee_{i=1}^{n} \|p_1 - \ell_i\|_1 \leq 0.1\Big),$$

where S_n is the MN-MAS with n agents and $t = 0, 1, \ldots, 9$. The agents' initial states are uncertain and can take values in $[-1, 1]^2$. Agents are controlled by $R_{2\times8}$ policies. The landmarks in S_2 and S_4 are at $(\pm 1/2, 0)$ and $(\pm 1/2, \pm 1/2)$. As summarised in Fig. 2, the verification time grows almost exponentially in the number of time steps t, and except $S_n \models \chi_t$, all other specifications are unsatisfiable.

Table 4: Runtimes (seconds) for $S_{2\times32} \models \varphi_{t,\epsilon} = \bigcirc^t(|\theta| < \epsilon)$ for different values of t and ϵ using the approach of (a) Akintunde et al., (b) Hosseini and Lomuscio, and (c) ours. Numbers indicate runtime (seconds). The white cells indicate that $S \models \varphi_{t,\epsilon}$, whilst the red cells indicate that $S \not\models \phi_{t,\epsilon}$. Instances marked by * are verified using BPT alone.

t	$\pi/30$	$\pi/20$	$\pi/10$	$\pi/30$	$\pi/20$	$\pi/10$	$\pi/30$	$\pi/20$	$\pi/10$
1	23.1	22.4	25.0	5.8	6.7	6.1	0.0*	0.0*	0.1*
2	195.3	210.5	196.8	12.5	13.5	19.6	0.1*	0.1*	0.1*
3	413.8	437.1	440.4	25.6	27.1	28.9	0.2*	14.1	14.5
4	1728.7	1724.6	1829.4	49.1	55.0	59.6	33.3	37.4	38.7
5	8648.9	7972.0	8295.2	77.7	80.4	83.5	50.4	49.6	55.5
6	–	–	–	92.9	99.7	113.5	73.1	72.3	80.0
7	–	–	–	147.6	159.9	185.8	100.7	99.2	107.8
8	–	–	–	335.9	393.9	340.2	193.4	287.4	211.0
9	–	–	–	776.0	946.8	767.2	234.8	475.7	289.6
10	–	–	–	1392.0	2241.2	1783.5	396.2	553.3	453.0

(a) RNSVerify [4] (b) RMILP [28] (c) Ours

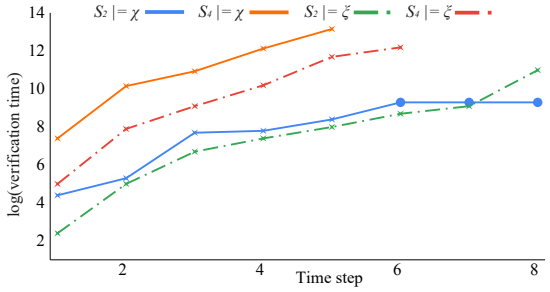


Figure 2: The base-2 logarithm of runtimes (seconds) for verifying ξ_t and χ_t using BPMC. All instances are verified using RMILP. Symbols • and x denote that the algorithm returned `True` and `False`, respectively.

CONCLUSIONS

We introduced the first framework for verifying full LTL specifications of multi-agent systems with memoryful neural agents under uncertainty by adapting well-established verification techniques. We evaluated the introduced algorithms on various widely used deep RL environments with single and multiple agents and verified bounded and unbounded $\text{LTL}_\mathbb{R}$ specifications. Compared to the SoA that consider fragments of $\text{LTL}_\mathbb{R}$, we improved the verification time by an order of magnitude. This work demonstrates how foundational verification techniques can be used for verifying the safety of MN-MAS, thus paving the way for the adoption of other advanced verification methods for the safety verification of MAS.

ACKNOWLEDGMENTS

This work was supported by supported by "REXASI-PRO" H-EU project, call HORIZON-CL4-2021-HUMAN01-01, Grant agreement ID: 101070028 and by the Engineering and Physical Sciences Research Council (EPSRC) under Award EP/W014785/2. Alessio Lomuscio acknowledges support from the Royal Academy of Engineering via a Chair of Emerging Technologies.

REFERENCES

[1] Michael Akintunde, Alessio Lomuscio, Lalit Maganti, and Edoardo Pirovano. 2018. Reachability Analysis for Neural Agent-Environment Systems. In *International Conference on Principles of Knowledge Representation and Reasoning, KR*. AAAI Press, 184–193.

[2] Michael E. Akintunde, Elena Botoeva, Panagiotis Kouvaros, and Alessio Lomuscio. 2020. Formal Verification of Neural Agents in Non-deterministic Environments. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*. IFAAMAS, 25–33.

[3] Michael E. Akintunde, Elena Botoeva, Panagiotis Kouvaros, and Alessio Lomuscio. 2020. Verifying Strategic Abilities of Neural-symbolic Multi-agent Systems. In *International Conference on Principles of Knowledge Representation and Reasoning, KR*. AAAI Press, 22–32.

[4] Michael E. Akintunde, Andreea Kevorchian, Alessio Lomuscio, and Edoardo Pirovano. 2019. Verification of RNN-Based Neural Agent-Environment Systems. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 6006–6013.

[5] Shaull Almagor, Udi Boker, and Orna Kupferman. 2016. Formally Reasoning About Quality. *J. ACM* 63, 3 (2016), 24:1–24:56.

[6] Shaull Almagor, Brynmor Chapman, Mehran Hosseini, Joël Ouaknine, and James Worrell. 2018. Effective Divergence Analysis for Linear Recurrence Sequences. In *International Conference on Concurrency Theory, CONCUR (LIPIcs, Vol. 118)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 42:1–42:15.

[7] Edoardo Bacci and David Parker. 2020. Probabilistic guarantees for safe deep reinforcement learning. In *Formal Modeling and Analysis of Timed Systems FORMATS*. Springer, 231–248.

[8] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. 1983. Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13, 5 (1983), 834–846.

[9] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. 2009. Bounded Model Checking. *Handbook of Satisfiability* 185, 99 (2009), 457–481.

[10] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of Adversarial Machine Learning. *Pattern Recognition* 84 (2018), 317–331.

[11] Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. 2020. Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 3291–3299.

[12] Mark Braverman. 2006. Termination of Integer Linear Programs. In *Computer Aided Verification, CAV (Lecture Notes in Computer Science, Vol. 4144)*. Springer, 372–385.

[13] Aykut Bulut and Ted K. Ralphs. 2021. On the Complexity of Inverse Mixed Integer Linear Optimization. *SIAM Journal on Optimization* 31, 4 (2021), 3014–3043.

[14] Claudia D'Ambrosio, Andrea Lodi, and Silvano Martello. 2010. Piecewise Linear Approximation of Functions of Two Variables in MILP Models. *Operations Research Letters* 38, 1 (2010), 39–46.

[15] Ruediger Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Automated Technology for Verification and Analysis, ATVA*. Springer, 269–286.

[16] Jiameng Fan, Chao Huang, Xin Chen, Wenchao Li, and Qi Zhu. 2020. ReachNN*: A Tool for Reachability Analysis of Neural-Network Controlled Systems. In *Automated Technology for Verification and Analysis, ATVA (Lecture Notes in Computer Science, Vol. 12302)*. Springer, 537–542.

[17] Zoubin Ghahramani and Geoffrey E. Hinton. 2000. Variational Learning for Switching State-Space Models. *Neural Computing* 12, 4 (2000), 831–864.

[18] Francesco Giacomarra, Mehran Hosseini, Nicola Paoletti, and Francesca Cairoli. 2025. Certified Guidance for Planning with Deep Generative Models. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*. IFAAMAS.

[19] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, et al. 2018. On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models. arXiv preprint arXiv:1810.12715.

[20] Igor Griva, Stephen G. Nash, and Ariela Sofer. 2008. *Linear and Nonlinear Optimization* (2nd ed.). SIAM.

[21] Albert Gu and Tri Dao. 2023. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. arXiv preprint arXiv:2312.00752.

[22] Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[23] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362.

[24] Matthew J. Hausknecht and Peter Stone. 2015. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*. AAAI Press, 29–37.

[25] Patrick Henriksen and Alessio R. Lomuscio. 2020. Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search. In *European Conference on Artificial Intelligence, ECAI*. IOS Press, 2513–2520.

[26] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[27] Mehran Hosseini. 2021. *On Termination and Divergence of Linear Programs*. Ph.D. Dissertation. University of Oxford.

[28] Mehran Hosseini and Alessio Lomuscio. 2023. Bounded and Unbounded Verification of RNN-Based Agents in Non-deterministic Environments. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*. IFAAMAS, 2382–2384.

[29] Mehran Hosseini, Joël Ouaknine, and James Worrell. 2019. Termination of Linear Loops over the Integers. In *International Colloquium on Automata, Languages, and Programming, ICALP (LIPIcs, Vol. 132)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 118:1–118:13.

[30] Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas, and Insup Lee. 2021. Verisig 2.0: Verification of Neural Network Controllers Using Taylor Model Preconditioning. In *International Conference on Computer Aided Verification, CAV (Lecture Notes in Computer Science, Vol. 12759)*. Springer, 249–262.

[31] Yuval Jacoby, Clark W. Barrett, and Guy Katz. 2020. Verifying Recurrent Neural Networks using Invariant Inference. In *Automated Technology for Verification and Analysis, ATVA (Lecture Notes in Computer Science, Vol. 12302)*. Springer, 57–74.

[32] Guy Katz, Derek A Huang, Duligur Ibeling, et al. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Computer Aided Verification, CAV (Lecture Notes in Computer Science, Vol. 11561)*. Springer, 443–452.

[33] Oleg Klimov. 2016. OpenAI Gym, Lunar Lander.

[34] Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. 2019. POPQORN: Quantifying Robustness of Recurrent Neural Networks. In *International Conference on Machine Learning, ICML*. PMLR, 3468–3477.

[35] Panagiotis Kouvaros and Alessio Lomuscio. 2021. Towards Scalable Complete Verification of ReLU Neural Networks via Dependency-based Branching. In *International Joint Conference on Artificial Intelligence, IJCAI*. ijcai.org, 2643–2650.

[36] Alessio Lomuscio and Lalit Maganti. 2017. An Approach to Reachability Analysis for Feed-Forward ReLU Neural Networks. arXiv preprint arXiv:1706.07351.

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. arXiv preprint arXiv:1312.5602.

[38] Pierre El Mqirmi, Francesco Belardinelli, and Borja G León. 2021. An Abstraction-based Method to Check Multi-Agent Deep Reinforcement-Learning Behaviors. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*. ACM, 474–482.

[39] Joël Ouaknine and James Worrell. 2014. Positivity Problems for Low-Order Linear Recurrence Sequences. In *Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*. SIAM, 366–379.

[40] Amir Pnueli. 1977. The Temporal Logic of Programs. In *Annual Symposium on Foundations of Computer Science, FOCS*. IEEE Computer Society, 46–57.

[41] Joshua Riley, Radu Calinescu, Colin Paterson, Daniel Kudenko, and Alec Banks. 2021. einforcement Learning with Quantitative Verification for Assured Multi-Agent Policies. In *nternational Conference on Agents and Artificial Intelligence, ICAART*. SciTePress.

[42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.

[43] Chelsea Sidrane, Amir Maleki, Ahmed Irfan, and Mykel J. Kochenderfer. 2022. OVERT: An Algorithm for Safety Verification of Neural Network Control Policies for Nonlinear Systems. *Journal of Machine Learning Research* 23, 117 (2022), 1–45.

[44] David Silver, Julian Schrittwieser, Karen Simonyan, et al. 2017. Mastering the Game of Go without Human Knowledge. *Nature* 550, 7676 (2017), 354–359.

[45] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing Properties of Neural Networks. In *International Conference on Learning Representations, ICLR*. OpenReview.net.

[46] J Terry, Benjamin Black, Nathaniel Grammel, et al. 2021. Pettingzoo: Gym for Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems, NeurIPS*. Curran Associates, Inc., 15032–15043.

[47] Ashish Tiwari. 2004. Termination of Linear Programs. In *Computer Aided Verification, CAV (Lecture Notes in Computer Science, Vol. 3114)*. Springer, 70–82.

[48] Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations, ICLR*. OpenReview.net.

[49] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, et al. 2023. Gymnasium. https://zenodo.org/record/8127025

[50] 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 *Computer Aided Verification, CAV (Lecture Notes in Computer Science, Vol. 12224)*. Springer, 3–17.

[51] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, et al. 2019. Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning. *Nature* 575, 7782 (2019), 350–354.

[52] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *USENIX*

Security Symposium. USENIX Association, 1599–1614.
[53] Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory Networks. arXiv preprint arXiv:1410.3916.
[54] Matthew Wicker, Luca Laurenti, Andrea Patane, Nicola Paoletti, Alessandro Abate, and Marta Kwiatkowska. 2021. Certification of iterative predictions in Bayesian neural networks. In *Uncertainty in Artificial Intelligence, UAI*. PMLR, 1713–1723.
[55] Matthew Wicker, Luca Laurenti, Andrea Patane, Nicola Paoletti, Alessandro Abate, and Marta Kwiatkowska. 2024. Probabilistic Reach-Avoid for Bayesian Neural Networks. *Artificial Intelligence* (2024), 104132.
[56] Rui Yan, Gabriel Santos, Xiaoming Duan, David Parker, and Marta Kwiatkowska. 2022. Finite-Horizon Equilibria for Neuro-Symbolic Concurrent Stochastic Games. In *Uncertainty in Artificial Intelligence*. PMLR, 2170–2180.
[57] Rui Yan, Gabriel Santos, Gethin Norman, David Parker, and Marta Kwiatkowska. 2022. Strategy Synthesis for Zero-Sum Neuro-Symbolic Concurrent Stochastic Games. arXiv preprint arXiv:2202.06255.