Verified Compositions of Neural Network Controllers for Temporal Logic Control Objectives

Jun Wang, Samarth Kalluraya and Yiannis Kantaros

Abstract— This paper presents a new approach to design verified compositions of Neural Network (NN) controllers for autonomous systems with tasks captured by Linear Temporal Logic (LTL) formulas. Particularly, the LTL formula requires the system to reach and avoid certain regions in a temporal/logical order. We assume that the system is equipped with a finite set of trained NN controllers. Each controller has been trained so that it can drive the system towards a specific region of interest while avoiding others. Our goal is to check if there exists a temporal composition of the trained NN controllers - and if so, to compute it - that will yield composite system behaviors that satisfy a user-specified LTL task for any initial system state belonging to a given set. To address this problem, we propose a new approach that relies on a novel integration of automata theory and recently proposed reachability analysis tools for NN-controlled systems. We note that the proposed method can be applied to other controllers, not necessarily modeled by NNs, by appropriate selection of the reachability analysis tool. We focus on NN controllers due to their lack of robustness. The proposed method is demonstrated on navigation tasks for aerial vehicles.

I. INTRODUCTION

Several methods have been proposed recently to train Neural Network (NN) controllers for autonomous systems. Such training methods include e.g., deep reinforcement learning (RL) [1] and model predictive control (MPC) [2]. Despite the high real-time performance of NN-driven systems, they typically lack safety and robustness guarantees as underscored by recent studies [3]. To address this limitation, various methods have been proposed to verify robustness properties of trained NN controllers [4], [5]. For instance, [5] addresses the problem of output range analysis of trained NNs given an input set. Safety verification of dynamical systems with feedback NN controllers has been studied as well in [6]–[12] and the references therein. Typically, these methods investigate if a dynamical system with a trained NN controller can satisfy a reach-avoid property given a set of possible initial states.

Common in all the above works is that they focus on learning or verifying the efficiency/safety of a single NN controller for a given task. However, the sample complexity and computational cost of learning a single NN controller increases drastically as the task complexity increases. Motivated by these limitations, compositional RL methods have been proposed recently that aim to learn a set of base NN controllers which are then composed to satisfy a complex task captured by temporal logics; see e.g., [13]–[15]. The key idea in these works is to decompose the task into simpler sub-tasks for which NN controllers can be learned, using RL, more efficiently. Then, these NN controllers are composed to satisfy the original task. Nevertheless, the resulting controllers often either lack safety guarantees or the provided guarantees are impractical (e.g., lower bounds on satisfaction probability) for safety-critical applications; also, these guarantees are typically specific to a fixed and given initial system state. To address this issue, in this paper we propose a new method to design verified temporal compositions of trained NN controllers for temporal logic tasks. Related is also the recent work in [16] which, however, unlike the above papers and ours, does not consider temporal logic tasks. For instance, in [16], the sub-tasks are revealed by the environment while in [13]–[15] the sub-tasks are ‘strategically’ selected to satisfy a temporal logic task.

Specifically, in this paper, we consider autonomous systems tasked with complex high-level missions captured by a fragment of Linear Temporal Logic (LTL), called co-safe LTL [17]. We assume that the system is governed by discrete-time linear time-varying dynamics and that the LTL task requires reaching and avoiding certain regions in a temporal/logical order. Also, the system has access to a finite set of already trained controllers modeled as NNs. Each controller is trained so that it can drive the system towards a specific region of interest while avoiding others. We do not make any assumptions about how these NNs have been trained; for instance they may have been trained using RL or MPC-based methods. Our goal is to check if there exists a temporal composition of these NN controllers - and if so, to compute it - that will yield composite system behaviors that always satisfy a user-specified LTL task for any initial system state belonging to a given set. To address this problem, we leverage automaton representations of LTL formulas as well as graph-search methods and existing reachability analysis for NN-driven systems [8]. We note that our approach can handle any other open-loop or feedback controllers that are not necessarily modeled as NNs by appropriate selection of the reachability analysis method. In this paper we focus on NN controllers due to their fragility to imperceptible input perturbations [3].

Contributions: First, we propose a new approach to design verified temporal compositions of NN controllers for co-safe LTL tasks. Second, we show correctness of the proposed method and discuss trade-offs between completeness and computational efficiency. Third, we demonstrate the efficiency of the proposed approach on several navigation tasks that involve aerial vehicles.
II. PROBLEM FORMULATION

Closed-loop system: We consider discrete-time linear systems defined as follows:
\[ x_{t+1} = f(x_t, u_t) = A_t x_t + B_t u_t + c_t, \]
where \( x_t \in \mathcal{X} \subseteq \mathbb{R}^d \) and \( u_t \in \mathcal{U} \subseteq \mathbb{R}^n \) denote the state and the control input of the system at time \( t \geq 0 \), respectively. Also, \( A_t \in \mathbb{R}^{d \times d}, B_t \in \mathbb{R}^{d \times n} \) are the system matrices while \( c_t \in \mathbb{R}^d \) is an exogenous input. We assume that state-space \( \mathcal{X} \) contains \( N > 0 \) sub-spaces denoted by \( \ell_i \in \mathcal{X} \) modelling regions of interest or unsafe areas. Also, we assume that at any time \( t \) the system can apply control inputs selected from a finite set of feedback controllers collected in the set \( \Xi = \{ \xi_i \}_{i=1}^N \), where \( \xi_i(x_t) : \mathcal{X} \to \mathbb{R}^n \) maps system states to control actions. We assume that the controller \( \xi_i \) is selected by the system when, given any initial state in \( \mathcal{X} \), the system state \( x_t \) needs to be driven towards the interior of \( \ell_i \). We consider cases where the controllers \( \xi_i(x_t) \) are parameterized by multi-layer feed-forward fully-connected neural networks (NNs). Such NN controllers can be implemented using available methods; see e.g., [18].

Hereafter, with slight abuse of notation, we denote by \( \xi(t) \) the controller selected from \( \Xi \) at time \( t \). To ensure that NN output respects the input constraint, we consider a projection operator, denoted by \( \text{Proj}_{\xi_i} \), and define the control input as \( u_t = \text{Proj}_{\xi_i} \xi(t) \). We denote the closed-loop system with dynamics (1) and the projected NN control policy as:
\[ x_{t+1} = f_c(x_t) \]
Next, we define a high-level NN-based control strategy \( \xi \) as a temporal composition of the controllers in \( \Xi \).

Definition 2.1 (Control Strategy): A NN-based control strategy \( \xi \) is defined as a finite sequence of NN controllers selected from \( \Xi \), i.e., \( \xi = \mu(0), \mu(1), \mu(2), \ldots, \mu(K) \), for some finite \( K > 0 \), where \( \mu(k) \in \Xi \) for all \( k \in \{0, 1, \ldots, K\} \), and \( \mu(k) \) is applied for a finite horizon \( H_k \).

We note again that \( \mu(k) \) is a feedback controller from \( \Xi \). For instance, if \( \mu(0) = \xi_i \), for some \( i \in \{1, \ldots, N\} \), then the system applies the controller \( \xi_i(x_t) \), \( \forall t \in \{0, 1, \ldots, H_0\} \).

Given \( \xi \) and an initial state \( x_0 \), the corresponding closed-loop system (2) generates a finite sequence of system states, denoted by \( \tau(x_0) = x_0, x_1, \ldots, x_l, \ldots, x_F \), where \( F = \sum_{k=0}^{K} H_k \).

Linear Temporal Logic Properties: We define mission and safety properties for the system (1) using Linear Temporal Logic (LTL) as it allows to specify a wide range of high-level tasks [19], [20]. LTL consists of atomic propositions (i.e., Boolean variables), denoted by \( \mathcal{AP} \); Boolean operators, (i.e., conjunction \( \land \) and negation \( \neg \)), and two temporal operators, next \( \circ \) and until \( \mathcal{U} \). LTL formulas over a set \( \mathcal{AP} \) can be constructed based on the following grammar:
\[ \phi ::= \text{true} \mid \pi \mid \phi_1 \land \phi_2 \mid \neg \phi \mid \phi_1 \mathcal{U} \phi_2, \]
where \( \pi \in \mathcal{AP} \). For brevity we abstain from presenting the derivations of other Boolean and temporal operators, e.g., always \( \square, \) eventually \( \diamond, \) implication \( \Rightarrow, \) which can be found in [17]. Hereafter, we define the set \( \mathcal{AP} \) as \( \mathcal{AP} = \cup_i \{ \pi_i \} \), where \( \pi_i \) is an atomic predicate that is true when the system state \( x_t \) is within region \( \ell_i \). We restrict our attention to co-safe LTL properties that exclude the use of the ‘always’ operator. Co-safe LTL formulas are satisfied by discrete finite plans \( \tau \) defined as finite sequences of system states \( x_t \in \mathcal{X} \); i.e., \( \tau(x_0) = x_0, x_1, \ldots, x_l, \ldots, x_F \), where \( F > 0 \) denotes a finite horizon [17]. Given \( \xi \) and an initial state \( x_0 \), we say that the closed-loop system (2) satisfies \( \phi \), denoted by \( f_\xi \models \phi \), if (2) generates a sequence \( \tau(x_0) \) that satisfies \( \phi \).

Example 2.2: Examples of co-safe LTL specifications follow: (i) \( \phi = \diamond (\pi_1) \land (\neg \pi_2 \mathcal{U} \pi_3) \) captures a common reach-avoid property requiring the system to eventually reach the region of interest \( \ell_1 \) while avoiding in the meantime the unsafe region \( \ell_2 \); (ii) \( \phi = (\diamond \pi_1) \land (\diamond \pi_2) \land (\diamond \pi_3) \land (\diamond \pi_4) \land (\neg \pi_5 \mathcal{U} \pi_6) \) requires the system to eventually reach the regions \( \ell_1, \ell_2, \ell_3, \) and \( \ell_4 \), in any order, as long as \( \ell_4 \) is avoided until region \( \ell_1 \) is reached.

Problem 1: Given (i) a set of initial states \( X_0 \subseteq \mathcal{X} \); (ii) the system dynamics (1); (iii) a co-safe LTL property \( \phi \); and (iv) a finite set \( \Xi \) of NN controllers, check if there exists a NN-based control strategy \( \xi \) (Definition 2.1), so that \( f_\xi \models \phi \), for all \( x_0 \in X_0 \); if there exists such a \( \xi \), compute it as well.

Remark 2.3 (Controllers for LTL tasks): Several reinforcement learning (RL) methods have been proposed that can train a single controller, that can be parameterized by a NN, to satisfy an LTL task \( \phi \); see e.g., [1], [21]-[23] and the references therein. Verification of a single NN controller (as opposed to the set of controllers considered here) with respect to an LTL formula can be accomplished by existing reachability analysis tools for discrete-time systems; see e.g., [8]. Particularly, a sequence of reachable sets, capturing all possible system states at future time instants \( t \), needs to be computed under the considered NN controller and a set of initial states \( X_0 \). Then, it suffices to check if the atomic predicates that are satisfied across this sequence construct a word that can be accepted by an automaton of \( \phi \) [17].

III. VERIFIED COMPOSITIONS OF NN CONTROLLERS
FOR CO-SAFE LTL TASKS

Our approach to solve Problem 1 consists of the following steps. First, we translate the LTL formula \( \phi \) into a Deterministic Finite state Automaton (DFA); see Section III-A. Second, by leveraging the DFA, we decompose \( \phi \) into reach-avoid sub-tasks; see Sections III-B-III-C. Then, we apply graph-search methods combined with reachability analysis to check if there exists \( \xi \) so that \( f_\xi \models \phi \), for all \( x_0 \in X_0 \); see Section III-D. Trade-offs between completeness and computational efficiency are discussed in Section III-E.

A. From LTL formulas to DFA

First, we translate \( \phi \), constructed using \( \mathcal{AP} \), into a DFA defined as follows [17]; see also Fig. 1a.

Definition 3.1 (DFA): A Deterministic Finite state Automaton (DFA) \( D \) over \( \Sigma = 2^{\mathcal{AP}} \) is defined as a tuple \( D = (Q_D, q_D^0, \Sigma, \delta_D, q_D^F) \), where \( Q_D \) is the set of states, \( q_D^0 \in Q_D \) is the initial state, \( \Sigma \) is an alphabet, \( \delta_D : Q_D \times \Sigma \to Q_D \) is a deterministic transition relation, and \( q_D^F \in Q_D \) is the accepting/ final state.
Fig. 1: Fig 1a shows the DFA corresponding to $\phi = \Diamond(\pi_{\ell_2}) \land (\neg \pi_{\ell_2} U \pi_{\ell_1})$. Fig. 1b illustrates the reachability analysis over the DFA state space (see Section III-D).

To interpret a temporal logic formula over a sequence $\tau(x_0)$ generated by (2), we use a labeling function $L : X \to 2^{4P}$ that maps system states to symbols $\sigma \in 2^{4P}$. A finite sequence of states $\tau(x_0) = x_0, x_1, \ldots, x_F$ satisfies $\phi$ if the word $w = L(x_0)L(x_1) \ldots L(x_F)$ yields a DFA transition run, i.e., if starting from the initial state $q_D^0$, each element in $w$ yields a DFA transition so that the final state $q_D^F$ is reached [17]. Note that a DFA can be constructed using existing tools such as [24].

B. From DFA to Reach-Avoid Properties

Given any DFA state $q_D \in Q_D$, we compute a set $R_{q_D}$ that collects all DFA states that can be reached, in one hop, from $q_D$ using a symbol $\sigma$. In math, we have:

$$R_{q_D} = \{q_D' | q_D' = \delta_D(q_D, \sigma), \sigma \in \Sigma\}. \quad (3)$$

Then, given $q_D$ and for each DFA state $q_D' \in R_{q_D}$, we introduce the following definitions. We construct a set that collects the states $x \in X$ so that if the system state coincides with one of these states, then a symbol $\sigma = L(x) \in \Sigma$ enabling this DFA transition will be generated. We collect these states in the set $X_{q_D^0 \to q_D^f}$, i.e.,

$$X_{q_D^0 \to q_D^f} = \{x \in X | q_D' = \delta_D(q_D, L(x))\} \quad (4)$$

In what follows, for simplicity, we assume that for all $q_D \in Q_D$ there exists a feasible self-loop around every DFA state $q_D$; later, in Section III-C, we relax this assumption. Starting from any system state $x \in X$ and a DFA state $q_D$, transition from $q_D$ to $q_D' \in R_{q_D}$ will eventually occur after $H_{q_D}$ discretely time steps, if (i) the system state $x_t$ remains within $X_{q_D^0 \to q_D^f}$ for the next $H_{q_D} - 1$ steps, and (ii) at time $t = H_{q_D}$, we have that $x_{H_{q_D}} \in X_{q_D^0 \to q_D^f}$. Essentially, (i)-(ii) model a reach-avoid requirement. For instance, (i) may require a robot to stay within the obstacle-free space and (ii) may require a robot to eventually enter a region.

Example 3.2 (Reach-Avoid Properties): Consider the DFA in Fig. 1a. We have that $R_{q_D^0} = \{q_D^0, q_D, q_D^f\}$. Also, we have that $X_{q_D^0 \to q_D^f} = X \setminus (t_1 \cup t_2)$, $X_{q_D^0 \to q_D^f} = (X \setminus t_2) \cap t_1$, $X_{q_D^0 \to q_D^f} = t_2 \cap t_1$. Similarly, we have that $R_{q_D^0} = \{q_D^0, q_D^f\}$, $X_{q_D^0 \to q_D^f} = X \setminus t_2$, and $X_{q_D^0 \to q_D^f} = t_2$.

C. Verifying Reach-Avoid Properties

In what follows, we discuss how to check whether a transition from a DFA state $q_D$ to $q_D' \neq q_D$ can be enabled; later, we will discuss how this can be used to verify LTL properties. Specifically, we want to verify that given an initial set of system states associated with $q_D$, denoted by $X_{q_D^0}$, the previously discussed conditions (i)-(ii) can be satisfied; the detailed construction of $X_{q_D^0}$ will be discussed in Section III-D. Notice $X_{q_D^0 \to q_D^f}$ may contain more than one region of interest $t_i$. As discussed in Section II, for each region of interest $t_i$ in $X_{q_D^0 \to q_D^f}$, the system selects the corresponding NN controller $\xi_i \in \Xi$. Hereafter, we collect all NN controllers associated with $X_{q_D^0 \to q_D^f}$ in a set denoted by $\Xi_{q_D^0 \to q_D^f} \subseteq \Xi$. In math, we want to show that there exists a finite horizon $H_{q_D}$ and at least one NN controller $\xi \in \Xi_{q_D^0 \to q_D^f}$, so that if the system evolves as per $x_{t+1} = f(x_t, \xi)$ then the following two conditions hold for all possible initial system states in $X_{q_D^0}^0$: (i) $x_t \in X_{q_D^0 \to q_D^f}, \forall t \in [0, H_{q_D} - 1]$ and (ii) $x_{H_{q_D}} \in X_{q_D^0 \to q_D^f}$. If such a horizon $H_{q_D}$ and controller $\xi \in \Xi_{q_D^0 \to q_D^f}$ exist, then by definition of the set $X_{q_D^0 \to q_D^f}$ in (4), we have that within the time interval $[0, H_{q_D} - 1]$, the transition from $q_D$ to $q_D'$ (self-loop) is enabled, and at the time step $t = H_{q_D}$ the transition from $q_D$ to $q_D'$ occurs. In this case, we say that the DFA transition from $q_D$ to $q_D'$ is verified to be safe when the system starts anywhere within $X_{q_D^0}$ and applies the NN controller $\xi$.

To reason about safety of a DFA transition, we leverage existing reachability analysis tools that can compute forward reachable sets $R_t(X_{q_D^0}, \xi) \subseteq X_{q_D^0}$ collecting all possible states $x$ of the system may reach after applying a feedback NN controller $\xi$ for $t$ time steps while starting anywhere in $X_{q_D^0}$. Given such reachable sets, it suffices to check if there exists a finite horizon $H_{q_D}$ and at least one controller $\xi \in \Xi_{q_D^0 \to q_D^f}$ such that the reachable sets satisfy the following two conditions: (i) $R_t(X_{q_D^0}, \xi) \subseteq X_{q_D^0 \to q_D^f}, \forall t \in [0, \ldots, H_{q_D} - 1]$ and (ii) $R_{H_{q_D}}(X_{q_D^0}, \xi) \subseteq X_{q_D^0 \to q_D^f}$. If both conditions hold, we verify that the DFA transition from $q_D$ to $q_D'$ is safe given the initial set of states $X_{q_D^0}$ and the controller $\xi$ [6].

Construction of exact reachable sets is computationally intractable. Thus, instead, we compute over-approximated reachable sets, denoted hereafter by $R_t(X_{q_D^0}, \xi) \subseteq \Xi_{q_D^0}(X_{q_D^0}, \xi)$, using tools that can handle systems of the form (1) with NN controllers [8]. If the following two conditions are satisfied

$$R_t(X_{q_D^0}, \xi) \subseteq X_{q_D^0 \to q_D^f}, \forall t \in [0, \ldots, H_{q_D} - 1]$$

then we say that the considered DFA transition is verified to be safe given the initial set of states $X_{q_D^0}$ and a feedback NN controller $\xi$. Finally, if there is no self-loop for $q_D$, then $X_{q_D^0 \to q_D^f}$ cannot be defined. In this case, such a transition from $q_D$ to $q_D'$ is verified to be safe if (6) holds for $H_{q_D} = 1$.

Example 3.3 (Verifying Reach-Avoid Properties (cont)): Consider the DFA in Fig. 1a and, specifically, the transition from $q_D^0$ to $q_D^f$. To reach $q_D^f$ from $q_D^0$, the available controllers are $\Xi_{q_D^0 \to q_D^f} = \{\xi_1, \xi_2\}$ by construction of the set

$^{1}$In case $X_{q_D^0 \to q_D^f}$ does not contain any region $t_i$, then $\Xi_{q_D^0 \to q_D^f} = \emptyset$ by definition of $\Xi$; see Ex. 3.3.

$^{2}$In practice, reachable sets over a large enough horizon $H$ are computed. If there is not reachable set $R_t$, for some $t \in \{0, 1, \ldots, H\}$ that satisfies (6), then we say that the system fails to reach this region of interest. This is in accordance with related works; see e.g., [6], [8].
DFS Algorithm

way store and recover from memory the sets $V_q$ that the system can be when it reaches currently visited node in $D$ of DFA transitions that are verified to be safe. This process $D$ verifed to be safe. For the transition $q_D$ from $q_D$ to $q_D'$, we have that $E_{q_D 	o q_D'} = \{ \xi \}$. Verification of this transition will be discussed in Ex. 3.4. Finally, we have $E_{q_D 	o q_D'} = \emptyset$ and $E_{q_D 	o q_D} = \{ \xi \}$.

D. From Verification of Reach-Avoid Properties to Verification of LTL formulas

To verify that the system satisfies $\phi$, it suffices to check that the final DFA state can be reached from the initial state by enabling a sequence of DFA transitions that are verified to be safe; see also Section III-A. To check this we rely on applying graph-search methods over the DFA state while verifying on-the-fly safety of DFA transitions using reachability analysis; see Section III-C. Specifically, first we view the DFA as a directed graph $D = (V, E)$ with vertices $V$ and edges $E$ that are determined by the set of states and transitions of the DFA. As discussed in Section III-C, to verify safety of DFA transition, an initial set of states is needed denoted by $X_{q_D}$. This set captures all possible states in $X$ that the system may have when it reaches a DFA state $q_D$. As a result, $X_{q_D}$ depends on the previous DFA states that the system has gone through to reach $q_D$. To simplify the proposed algorithm, we pre-process $D$ so that each node in $D$ that can be reached through multiple paths (excluding self-loops) originating from $q_D$ is replicated so that each replica can be reached through a unique path (excluding self-loops). For each vertex $q_D \in V$, we define sets that collect its incoming and outgoing edges denoted by $E_{in}^q$ and $E_{out}^q$, respectively. If the number of incoming edges (excluding self-loops) for $q_D$ is greater than 1, i.e., $|E_{in}^q| > 1$ we create $|E_{in}^q|$ copies of $q_D$ denoted by $q_D$. Each node $q_D$ has only one incoming edge which is selected to be the $i$-th edge in $E_{in}^q$, denoted by $E_{in}^q(i)$, while its outgoing edges remain the same as in the original node $q_D$. Then we add all copies to the graph and remove the original nodes $q_D$. We denote the resulting graph by $D'$.

Next, we apply a Depth-first search (DFS) method over $D'$ to see if $q_D$ can be reached from $q_D$ through a sequence of DFA transitions that are verified to be safe. This process is summarized in Alg. 1. The inputs to this algorithm are the graph $D'$, the set $\Xi$ of NN controllers and the system dynamics (1) (line 1). In what follows, we denote by $q_D$ the currently visited node in $D'$. The set of all possible states in $X$ that the system can be when it reaches $q_D$ is denoted by $X_{q_D}$. We initialize $E_{in}^q$ as $q_D = \emptyset$ and $X_{q_D} = \emptyset$. Also, we define a set $V_{vis}$ that collects all nodes in $D'$ that have been visited and a sequence of nodes that points to the current path from $q_D$ towards $q_D$. They are initialized as $V_{vis} = \{ q_D \}$ and $q = q_D$. Also we initialize the control strategy $\xi$ as an empty sequence. We also define a function $g : V \to X$ that maps a DFA state $q_D \in V$ to the corresponding set $X_{q_D}$; this function that is constructed on-the-fly is needed only as way store and recover from memory the sets $X_{q_D}$ (lines 2-3).

Given $q_D$, we randomly select a next state $q_{next} \in R_{q_D}$. Then we apply reachability analysis over the transition from $q_D$ to $q_{next}$ using the NN controllers $\Xi_{q_D \to q_{next}} \subseteq \Xi$ and the initial set of states $X_{q_D}$ (lines 5-6); see Section III-C. If the transition is verified to be safe, then we append $q_{next}$ to $q$. Also, we append the controller $\xi \in \Xi_{q_D \to q_{next}}$ for which this transition is safe to $\xi$ that denotes the current control strategy to reach $q_{next}$ from $q_D$ (lines 7-8). The corresponding horizon $H_{q_D}$ should be stored as well; we abstain from this for simplicity of presentation. The final reachable set $R_{q_D} = R_{q_D} \setminus q_{next}$ becomes the set $X_{q_D}$; see Ex. 3.4 and Fig. 1b as line (8). Also, $g_{max}$ is constructed on-the-fly as $g_{max}(q_{next}) = X_{q_{next}}$ and we replace $q_{cur}$ with $q_{max}$ (line 9). If the transition $q_D \to q_{next}$ is not verified to be safe, we remove $q_{next}$ from the set $R_{q_D}$ (see (3)). Then we keep taking the last element in $q$ and $\xi$, denoted by $q$ (end) and $\xi$ (end), while $q$ (end) is assigned to $q_{cur}$ until we find another state in $R_{q_D}$ that has not been visited yet (lines 12-16). The above process is repeated until $q_{cur}$ is updated to $q_{next}$. In this case, we have found a NN control strategy $\xi$ that satisfies $\phi$ for all initial states $x_0 \in X_0$ (line 11). If $q$ is empty yet we fail to find another state in $R_{q_D}$ that is not visited, then the proposed method cannot find a feasible path from $q_D$ to $q_{next}$ (even though it may exist; see Section III-E) (lines 17-18). We note that any other graph-search method in conjunction with reachability analysis can be used as well.

Example 3.4 (LTL Verification (cont)): We continue Ex. 3.3; see also Fig. 1b. To verify the DFA transition from $q_{cur} = q_D$ to $q_{next} = q_{next}$, we initialize $X_{q_D} = R_{q_D}(X_0, \xi)$. By computing $R_{q_D}(X_0, \xi)$, we verify that this DFA transition is safe. Thus, there exists $\xi = \xi_1, \xi_2$ where $\xi_1$ and $\xi_2$ are applied for 3 and 2 time steps so that $F_1 = \phi$, for all $x_0 \in X_0$.

E. Correctness & Completeness

 Proposition 3.5 (Correctness): The proposed method is correct, i.e., the computed $\xi$ solves Problem 1.

Proof: This result holds by construction of the proposed
method. Particularly, we ensure that the closed-loop system (2) driven by $\xi = \mu(0), \ldots, \mu(K)$, where each $\mu(k)$ is applied for $H_k$ time-steps, generates trajectories $\tau(x_0) = x_0, \ldots, x_F$, where $F = \sum_{k=0}^{H_k}$ that satisfy the system dynamics and the LTL formula $\phi$, $\forall x_0 \in X_0$.

Remark 3.6 (Computational Efficiency vs Completeness): In general, our method is not complete in the sense that it may not find a control strategy $\xi$ that satisfies $\phi$, for all $x_0 \in X_0$, even though such a strategy exists. This is due to the fact that (a) Alg. 1 computes over-approximated reachable sets and that (b) it does not exhaustively search over all possible combinations of NN control actions that the system can apply when a new DFA state is reached. As for (b), for instance, given a initial set of states for $q_D^{\text{out}}$, Alg. 1 reasons about safety of a transition from $q_D^{\text{out}}$ to $q_D^{\text{in}}$, using only controllers selected from $\Xi_{\text{out}}$. If this transition is unsafe, then it is discarded. However, this transition may become feasible if the initial set of states changes, which can happen by applying a controller from $\Xi_{\text{out}}$ to $q_D^{\text{out}}$, for all $\hat{\ell}$ time steps. Additionally, as soon as Alg. 1 finds a $\xi \in \Xi_{\text{out}}$ for which the corresponding transition is safe, it proceeds to new DFA transitions. However, that transition may be safe for other controllers in $\xi \in \Xi_{\text{in}}$, as well, where each one yields a different initial set for subsequent DFA transitions affecting their safety. The proposed method can be extended to account for these additional control actions at the expense of increasing its computational cost. We note that such trade-offs are quite common in related works; see e.g., [25]. The proposed method is complete if (i) there are no self-loops in the DFA states, or if $\Xi_{\text{out}} = \emptyset$ for all states (see e.g., Ex. 3.3); (ii) $|\Xi_{\text{out}}| = 1$, for all $q_D \in Q_D \setminus \{q_D^F\}$; (iii) the reachable sets are accurately computed.

IV. EXPERIMENTS

In this section, we demonstrate our framework on an unmanned aerial vehicle (UAV) with various LTL properties.

System Dynamics: We consider a UAV with dynamics as in (1) where the matrices $A_i, B_i, c_i$ are defined as in [8]. The UAV state is defined as $x = [p_x; p_y; p_z; v_x; v_y; v_z] \in \mathbb{R}^6$ capturing the position and velocity. The control input $u \in \mathbb{R}^3$ is a function of pitch, roll and thrust.

NN controllers: In what follows, we define LTL specifications that require the UAV to visit certain disjoint regions of interest $\ell_i \in \mathbb{R}^3$. We highlight that the $\ell_i$ regions are defined only over the UAV position, i.e., $\ell_i \subseteq \Omega$. Also, since we assume disjoint regions, the DFA can be pruned by removing infeasible DFA transitions reducing the computational cost for verification [20]. We train the NN controllers similarly to [8]; nevertheless, any other method (e.g., RL) can be used to train them. Specifically, to train $\xi_i$, we leverage nonlinear Model Predictive Control (MPC) methods. First we generate a random set of states $x \in \mathbb{R}^6$. Starting from each one of these states, we generate a sequence of pairs of states and control inputs that drive the UAV towards the interior of $\ell_i$ using an off-the-shelf MPC solver [26]. Each pair constitutes a data point in a training dataset. Using this dataset we train feedforward NN controllers $\xi_i$ with 2 hidden layers, 30 neurons/layer, and ReLU activation functions.

Case Study I: In this case study, we consider the following LTL formula $\phi = (\Diamond \pi_1^a) \land (\Diamond \pi_2^a) \land (\neg(\pi_1^a \lor \pi_2^a) \lor \pi_3^a) \land (\neg(\pi_1^a \lor \pi_2^a) \lor \pi_4^a)$ requiring the UAV to eventually visit the regions $\ell_1$ and $\ell_2$, in any order, while avoiding the obstacles $\ell_3$ and $\ell_4$. This formula corresponds to the DFA shown in Fig. 2a; notice that the transition from $q_D^0$ to $q_D^1$ was pruned and, therefore, not considered during verification since it requires the UAV to be present in more than one region simultaneously. To train $\xi_1$ and $\xi_2$ we collected 160000 data-points for each controller. Given these trained NN controllers and an initial set $X_0$ of states defined as an ellipsoid centered at $[3.5;3.5;2.9]$ with shape matrix of diag$[0.5^2;0.5^2;0.5^2]$.

We check if there exists a sequence of control actions that satisfy $\phi$. Particularly, first it investigates the DFA transition from $q_D^0$ to $q_D^1$. This transition requires the UAV to stay within the obstacle-free space (i.e., avoid the obstacles $\ell_3$ and $\ell_4$) and eventually reach $\ell_2$. The corresponding reachable sets for this DFA transition are shown in Fig. 3a. Notice that the reachable sets $R_i(X_0, \xi_2)$ are fully outside the obstacle regions for $t = 1, 2$ while at $t = 3$ the corresponding reachable set is fully inside $\ell_2$. Thus, this DFA transition is verified to be safe. Next, the DFA transition from $q_D^1$ to $q_D^2$ is considered requiring the robot to reach $\ell_1$ while avoiding the obstacle regions. The set of initial states for this transition is $X_0^0 = R_3(X_0, \xi_2)$. After computing reachable sets $R_i(X_0^0, \xi_1)$ (not shown), we can see that for $t = 1, 2$ they are outside the obstacles while $R_3(X_0^0, \xi_1)$ is inside $\ell_1$ verifying that this DFA transition is also safe. Thus, there exists a control strategy $\xi = \xi_2, \xi_1$ where $H_1 = H_2 = 3$ such that $x_0 \models \phi$ for all $x_0 \in X_0$.

Case Study II: In this case study, we consider an LTL formula $\phi = (\Diamond \pi_2^f \lor \pi_3^f) \land (\neg(\pi_1^f \lor \pi_2^f) \lor \pi_4^f)$ requiring the UAV to eventually visit the regions $\ell_1$ and $\ell_2$, in any order, while avoiding the obstacles $\ell_3$ and $\ell_4$. This formula corresponds to the DFA shown in Fig. 2b; notice that the transition from $q_D^0$ to $q_D^1$ was pruned and, therefore, not considered during verification since it requires the UAV to be present in more than one region simultaneously. To train $\xi_1$ and $\xi_2$ we collected 160000 data-points for each controller. Given these trained NN controllers and an initial set $X_0$ of states defined as an ellipsoid centered at $[3.5;3.5;2.9]$ with shape matrix of diag$[0.5^2;0.5^2;0.5^2]$.

We check if there exists a sequence of control actions that satisfy $\phi$. Particularly, first it investigates the DFA transition from $q_D^0$ to $q_D^1$. This transition requires the UAV to stay within the obstacle-free space (i.e., avoid the obstacles $\ell_3$ and $\ell_4$) and eventually reach $\ell_2$. The corresponding reachable sets for this DFA transition are shown in Fig. 3a. Notice that the reachable sets $R_i(X_0, \xi_2)$ are fully outside the obstacle regions for $t = 1, 2$ while at $t = 3$ the corresponding reachable set is fully inside $\ell_2$. Thus, this DFA transition is verified to be safe. Next, the DFA transition from $q_D^1$ to $q_D^2$ is considered requiring the robot to reach $\ell_1$ while avoiding the obstacle regions. The set of initial states for this transition is $X_0^0 = R_3(X_0, \xi_2)$. After computing reachable sets $R_i(X_0^0, \xi_1)$ (not shown), we can see that for $t = 1, 2$ they are outside the obstacles while $R_3(X_0^0, \xi_1)$ is inside $\ell_1$ verifying that this DFA transition is also safe. Thus, there exists a control strategy $\xi = \xi_2, \xi_1$ where $H_1 = H_2 = 3$ such that $x_0 \models \phi$ for all $x_0 \in X_0$.
Fig. 4: Case Study III: Verification of the DFA transition from \( q_0 \) to \( q_1 \). The red ellipsoid corresponds to a reachable set being fully inside the region to avoid.

requiring the UAV to eventually visit either region \( \ell_1 \) or \( \ell_2 \), while avoiding obstacles \( \ell_3 \) and \( \ell_4 \). This formula corresponds to the DFA shown in Figure 2b. The NN controller \( \xi \) is the same as in the previous case study while \( \ell_1 \) was trained using only 100 datapoints to mimic a poorly designed controller. The initial set of states \( X_0 \) is defined as an ellipsoid centered at \( [0.4; 0.4; 0.3] \) with shape matrix of diag\( [0.5^2; 0.5^2; 0.5^2] \). Given the DFA, we investigate the transition from \( q_0 \) to \( q_1 \) which requires the UAV to reach either \( \ell_1 \) or \( \ell_2 \) while avoiding both \( \ell_3 \) and \( \ell_4 \). In other words, we have that \( X_{q_0 \rightarrow q_1} = \{ \Omega \setminus (\ell_3 \cup \ell_4) \} \) and \( X_{q_1 \rightarrow q_2} = \{ \ell_1 \cup \ell_2 \} \). Thus, we have that \( X_{q_0 \rightarrow q_2} = \{ \xi, \ell_2 \} \). First, we check if this DFA transition is safe using \( \xi \). The generated reachable sets are shown in Figure 3b; only the first 4 reachable sets are shown. We observed that \( \ell_1 \) could not be reached within a large enough number of iterations. Thus, we cannot reason about safety of this DFA transition under \( \xi \). Thus, next we consider the controller \( \xi_2 \) and we constructed the sets \( \mathcal{R}_i(X_0, \xi_2) \) (not shown) demonstrating that all obstacles are avoided while at \( t = 6 \) the reachable set is fully inside \( \ell_2 \). Thus, we verify that there exists \( \xi = \xi_2 \), where \( \xi_2 \) is applied for 6 steps, so that \( f_\xi = \phi \), for all \( X_0 \in X_0 \).

**Case Study III:** We revisit the LTL formula considered in Ex. 3.2-3.4 with DFA shown in Fig. 1a. The controllers \( \xi_1 \) and \( \xi_2 \) are the same as in case study I while \( X_0 \) is defined as an ellipsoid centered at \( [3.4; 3.4; 3.1] \) with shape matrix of diag\( [0.5^2; 0.5^2; 0.5^2] \). To check whether the DFA transition from \( q_0 \) to \( q_1 \) is safe, we check if the system can reach \( \ell_1 \) while avoiding \( \ell_2 \). By constructing the reachable sets, we notice that \( \ell_1 \) cannot be reached without entering \( \ell_2 \) first; see Fig. 4. Thus, we cannot find a \( \xi = \phi \); see Rem. 3.6.

**V. CONCLUSION**

This paper proposed a new method to design verified compositions of NN controllers for co-safe LTL tasks. We showed its efficiency in navigation tasks for aerial vehicles.

**REFERENCES**

[1] Q. Gao, D. Hajinezhad, Y. Zhang, Y. Kantaros, and M. M. Zavlanos, “Reduced variance deep reinforcement learning with temporal logic specifications,” in 10th ACM/IEEE International Conference on Cyber-Physical Systems, 2019, pp. 237–248.

[2] V. Rubies-Royo, D. Fridovich-Keil, S. Herbert, and C. J. Tomlin, “A classification-based approach for approximate reachability,” in 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp. 7697–7704.

[3] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, “Adversarial attacks on neural network policies,” arXiv preprint arXiv:1702.02284, 2017.

[4] M. Fazlyab, M. Morari, and G. J. Pappas, “Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming,” IEEE Trans on Automatic Control, 2020.

[5] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, “Output range analysis for deep feedforward neural networks,” in NASA Formal Methods Symposium. Springer, 2018, pp. 121–138.

[6] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu, “Reachnn: Reachability analysis of neural-network-controlled systems,” ACM Transactions on Embedded Computing Systems (TECS), vol. 18, no. 5, pp. 1–22, 2019.

[7] X. Sun, H. Khedr, and Y. Shoukry, “Formal verification of neural network controlled autonomous systems,” in International Conference on Hybrid Systems: Computation and Control, 2019, pp. 147–156.

[8] H. Hu, M. Fazlyab, M. Morari, and G. J. Pappas, “Reach-sdp: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming,” 59th IEEE Conference on Decision and Control (CDC), pp. 5929–5934, December 2020.

[9] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, “NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems,” Computer Aided Verification, vol. 12224, pp. 3 – 17, 2020.

[10] S. Dutta, X. Chen, and S. Sankaranarayanan, “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, 2019, pp. 157–168.

[11] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. Pappas, and I. Lee, “Verisig 2.0: Verification of neural network controllers using taylor model preconditioning,” in International Conference on Computer Aided Verification. Springer, 2021, pp. 249–262.

[12] S. Sun, Y. Zhang, X. Luo, P. Vlantis, M. Pajic, and M. M. Zavlanos, “Formal verification of stochastic systems with relu neural network controllers,” in 2022 International Conference on Robotics and Automation (ICRA). IEEE, 2022, pp. 6800–6806.

[13] K. Jothimurugan, S. Bansal, O. Bastani, and R. Alur, “Compositional reinforcement learning from logical specifications,” Advances in Neural Information Processing Systems, vol. 34, 2021.

[14] G. N. Tasse, D. Jarvis, S. James, and B. Rosman, “Skill machines: Temporal logic composition in reinforcement learning,” arXiv preprint arXiv:2205.12532, 2022.

[15] C. Neary, C. Verginis, M. Cubuktepe, and U. Topcu, “Verifiable and compositional reinforcement learning systems,” in Proceedings of the International Conference on Automated Planning and Scheduling, vol. 32, 2022, pp. 615–623.

[16] R. Ivanov, K. Jothimurugan, S. Hsu, S. Vaidya, R. Alur, and O. Bastani, “Compositional learning and verification of neural network controllers,” ACM Trans. Embed. Comput. Syst., vol. 20, no. 5s, sep 2021. [Online]. Available: https://doi.org/10.1145/3477023

[17] C. Baier and J.-P. Katoen, Principles of model checking. MIT press Cambridge, 2008, vol. 26202649.

[18] S. W. Chen, T. Wang, N. Atanasov, V. Kumar, and M. Morari, “Large scale model predictive control with neural networks and primal active sets,” Automatica, vol. 135, p. 109947, 2022.

[19] K. Leahy, D. Zhou, C.-I. Vasile, K. Oikonomopoulos, M. Schwager, and C. Bela, “Persistent surveillance for unmanned aerial vehicles subject to charging and temporal logic constraints,” Autonomous Robots, vol. 40, no. 8, pp. 1363–1378, 2016.

[20] Y. Kantaros and M. M. Zavlanos, “Stilus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems,” International Journal of Robotics Research, 2020.

[21] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak, “Omega-regular objectives in model-free reinforcement learning,” TACAS, 2018.

[22] Y. Kantaros, “Accelerated reinforcement learning for temporal logic control objectives,” in IEEE/RSJ International Conference on Intelligent Robots and Systems, Kyoto, Japan, October 2022.

[23] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic, “Control synthesis from linear temporal logic specifications using model-free reinforcement learning,” in 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020, pp. 10 349–10 355.

[24] F. Fugitti, “L1ld2fa,” March 2019. [Online]. Available: https://github.com/whitemech/l1ld2fa

[25] K. Leahy, A. Jones, and C. I. Vasile, “Fast decomposition of temporal logic specifications for heterogeneous teams,” IEEE Robotics and Automation Letters, 2022.

[26] N. Andrei, “A sqp algorithm for large-scale constrained optimization: Snopt,” in Continuous nonlinear optimization for engineering applications in GAMS technology. Springer, 2017, pp. 317–330.