Neurosymbolic Motion and Task Planning for Linear Temporal Logic Tasks

Xiaowu Sun, *Graduate Student Member, IEEE*, and Yasser Shoukry, *Senior Member, IEEE*

***Abstract*—This article presents a neurosymbolic framework to solve motion planning problems for mobile robots involving temporal goals. The temporal goals are described using temporal logic formulas, such as bounded linear temporal logic (LTL) and co-safe LTL to capture complex tasks. The proposed framework trains neural network (NN)-based planners that enjoy strong correctness guarantees when applying to unseen tasks, i.e., the exact task (including workspace, temporal logic formula, and errors in the dynamical models of the robot) is not available during the training of NNs. Our approach to achieving theoretical guarantees and computational efficiency is based on two insights. First, we incorporate a symbolic model into the training of NNs such that the resulting NN-based planner inherits the interpretability and correctness guarantees of the symbolic model. Moreover, the symbolic model serves as a discrete "memory," which is necessary for satisfying temporal logic formulas. Second, we train a library of NNs offline and combine a subset of the trained NNs into a single NN-based planner at runtime when a task is revealed. In particular, we develop a novel constrained NN training procedure, named formal NN training, to enforce that each NN in the library represents a "symbol" in the symbolic model. As a result, our neurosymbolic framework enjoys the scalability and flexibility benefits of machine learning and inherits the provable guarantees from control-theoretic and formal-methods techniques. We demonstrate the effectiveness of our framework in both simulations and on an actual robotic vehicle and show that our framework can generalize to unseen tasks where state-of-the-art meta-reinforcement learning techniques fail.**

***Index Terms*—Formal methods, meta-reinforcement learning, neural networks (NNs).**

I. Introduction

DEVELOPING intelligent machines with a considerable level of cognition dates to the early 1950 s. With the current rise of machine learning (ML) techniques, robotic platforms are witnessing a breakthrough in their cognition. Nevertheless, regardless of how many environments they were trained (or programmed) to consider, such intelligent machines will always face new environments, which the human designer failed to examine during the training phase. To circumvent the lack of autonomous systems to adapt to new environments, several researchers asked whether we could build autonomous agents that can learn how to learn. In other words, while conventional ML focuses on designing agents that can perform one task, the so-called meta-learning aims instead to solve the problem of designing agents that can generalize to different tasks that were not considered during the design or the training of these agents. For example, in the context of meta-reinforcement learning (meta-RL), given data collected from a multitude of tasks (e.g., changes in the environments, goals, and robot dynamics), meta-RL aims to combine all such experiences and use them to design agents that can quickly adapt to unseen tasks. While the current successes of meta-RL are undeniable, significant drawbacks of meta-RL in its current form are as follows:

1) *the lack of formal guarantees on its ability to generalize to unseen tasks*;
2) *the lack of formal guarantees with regards to its safety*;
3) *the lack of interpretability due to the use of black-box deep learning techniques*.

In this article, we focus on the problem of designing neural network (NN)-based task and motion planners that are guaranteed to generalize to unseen tasks, enjoy strong safety guarantees, and are interpretable. We consider agents who need to accomplish temporal goals captured by temporal logic formulas, such as bounded linear temporal logic (BLTL) [1] and co-safe LTL (scLTL) [2]. The use of BLTL/scLTL in task and motion planning has been widely studied (e.g., [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]) due to the ability of BLTL/scLTL formulas to capture complex goals, such as "eventually visit region A followed by a visit to region B or region C while always avoiding hitting obstacle D." On the one hand, motion and task planning using symbolic techniques enjoy the guarantees of satisfying task specifications in temporal logic. Nevertheless, these algorithms need an explicit model of the dynamic constraints of the robot and suffer from computational complexity whenever such dynamic constraints are highly nonlinear and complex. On the other hand, ML approaches are capable of training NN planners without the explicit knowledge of the dynamic constraints and scale favorably to highly nonlinear and complex dynamics. Nevertheless, these data-driven approaches suffer from the lack of safety and generalization guarantees. Therefore, in this work, we aim to design a novel *neurosymbolic* framework for motion and task planning by combining the benefits of symbolic control and ML techniques.

At the heart of the proposed framework is using a symbolic model to guide the training of NNs and restricting the behavior of NNs to "symbols" in the symbolic model. Specifically,

Manuscript received 15 September 2023; revised 27 February 2024; accepted 8 April 2024. Date of publication 22 April 2024; date of current version 6 May 2024. This paper was recommended for publication by Associate Editor S. L. Smith and Editor D. Hsu upon evaluation of the reviewers' comments. This work was sponsored by the NSF under Grant #CNS-2002405, Grant #CNS-2013824, and Grant #CNS-2313104. *(Corresponding author: Yasser Shoukry.)*

The authors are with the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA 92697 USA (e-mail: xiaowus@uci.edu; yshoukry@uci.edu).

Digital Object Identifier 10.1109/TRO.2024.3392079

our framework consists of offline (or training) and online (or runtime) phases. During the offline phase, we assume access to a "nominal" simulator that approximates the dynamic constraints of a robot. We assume no knowledge of the exact task (e.g., workspace, BLTL/scLTL formula, and exact dynamic constraints of a robot). We use this information to train a "library" of NNs through a novel NN training procedure, named formal NN training, which enforces each trained NN to represent a continuous piece-wise affine (CPWA) function from a chosen family of CPWA functions. Our focus on CPWA functions is motivated by the fact that any NN with a rectifier linear unit (ReLU) nonlinearity represents a CPWA function [15]. While standard NN training can not restrict the NN to a particular CPWA function, our proposed formal NN training enjoy such a guarantee. The exact task becomes available only during the online (or runtime) phase. Given the dynamic constraints of a robot, we compute a finite-state Markov decision process (MDP) as our symbolic model. Thanks to the formal NN training procedure, the symbolic model can be constructed so that each of the trained NNs in the library represents a transition in the MDP (and hence a symbol in this MDP). By analyzing this symbolic model, our framework selects NNs from the library and combines them into a single NN-based planner to perform the task and motion planning.

In summary, the main contributions of this article are as follows.

1) We propose a *neurosymbolic* framework that integrates ML and symbolic techniques in training NN-based planners for an agent to accomplish *unseen* tasks. Thanks to the use of a symbolic model, the resulting NN-based planners are guaranteed to satisfy the temporal goals described in linear temporal logic formulas, which cannot be satisfied by existing NN training algorithms.
2) We develop a formal training algorithm that restricts the trained NNs to specific local behavior. The training procedure combines classical gradient descent training of NNs with a novel *NN weight projection operator* that modifies the NN weights as little as possible to ensure the trained NN belongs to a chosen family of CPWA functions. We provide theoretical guarantees on the proposed *NN weight projection operator* in terms of correctness and upper bounds on the error between the NN before and after the projection.
3) We provide a theoretical analysis of the overall *neurosymbolic* framework. We show theoretical guarantees that govern the correctness of the resulting NN-based planners when generalizing to *unseen* tasks, including workspaces, temporal logic formulas, and errors in the robot's dynamical model.
4) We pursue the high performance of the proposed framework in fast adaptation to unseen tasks with efficient training. For example, we accelerate the training of NNs by employing ideas from transfer learning and constructing the symbolic model using a data-driven approach. We validate the effectiveness of the proposed framework on an actual robotic vehicle and demonstrate that our framework can generalize to unseen tasks where state-of-the-art meta-RL techniques are known to fail (e.g., when the tasks are chosen from across homotopy classes [16]).

The rest of this article is organized as follows. After the problem formulation in Section II, we present the formal NN training algorithm in Section III. In Section IV, we introduce the neurosymbolic framework that uses the formal NN training algorithm to obtain a library of NNs and combines them into a single NN-based planner at runtime. In Section V, we provide theoretical guarantees of the proposed framework. In Section VI, we present some key elements for performance improvement while maintaining the same theoretical guarantees. Experimental results are given in Section VII. Finally, Section VIII concludes this article. All proofs can be found in the appendix.

Comparison with the preliminary results: A preliminary version of this article was presented in [17]. In [17], we confined our goal to generating collision-free trajectories, whereas in this work, we consider agents that need to satisfy general temporal logic formulas, such as BLTL/scLTL. Also, we assume that temporal logic formulas and the exact robot dynamics are unavailable during the training of NNs. In this article, we present for the first time the formal NN training algorithm (see Section III). Moreover, we present a theoretical analysis of the proposed framework (see Section V). All the speedup techniques in Section VI, the implementation of our framework on an actual robotic vehicle, and the performance comparison with meta-RL algorithms are also new in this article.

Related work: The literature on the safe design of ML-based motion and task planners can be classified according to three broad approaches, namely, as follows.

1) Incorporating safety in the training of ML-based planners.
2) Post-training verification of ML models.
3) Online validation of safety and control intervention.

Representative examples of the first approach include reward-shaping [18], [19], Bayesian and robust regression [20], [21], [22], and policy optimization with constraints [23], [24], [25]. Unfortunately, these approaches do not provide provable guarantees about the safety of the trained ML-based planners.

To provide strong safety and reliability guarantees, several works in the literature focus on applying formal verification techniques (e.g., model checking) to verify pretrained ML models against formal safety properties. Representative examples of this approach include the use of SMT-like solvers [26], [27], [28], [29], [30], [31] and hybrid-system verification [32], [33], [34]. However, these techniques only assess a given ML-based planner's safety rather than design or train a safe agent.

Due to the lack of safety guarantees on the resulting ML-based planners, researchers proposed several techniques to *restrict* the output of the ML models to a set of safe control actions. Such a set of safe actions can be obtained through Hamilton–Jacobi analysis [35], [36] and barrier certificates [37], [38], [39], [40], [41], [42], [43]. Unfortunately, methods of this type suffer from being computationally expensive, specific to certain controller structures, or requiring assumptions on the system model. Other techniques in this domain include synthesizing a safety layer (shield) based on model predictive control with the assumption of safe terminal sets [44], [45], [46], logically-constrained

reinforcement learning [47], [48], [49], and Lyapunov methods [50], [51], [52] that focus on providing stability guarantees rather than safety or general temporal logic guarantees.

Another line of related work is the integration of logical reasoning with reinforcement learning. In particular, the work reported in [53], [54] focuses on transforming logical constraints (e.g., temporal logic mission constraints) into reward machines with the guarantee that finding a policy that achieves their global optima leads to the satisfaction of the logical mission constraints. Unfortunately, current RL algorithms do not enjoy guarantees in terms of achieving the global optima except when restricted to searching over a finite set of states and control actions. Moreover, these techniques assume the knowledge of the mission task during the training of the RL agent and do not enjoy generalization guarantees to unseen tasks. Another direction—to integrate logical reasoning with RL—is reported in [55], [56], [57], and [58], where the target is to compose different RL agents to satisfy temporal logic constraints. Unfortunately, the work in [55] and [58] is restricted to scenarios with finite state and control action spaces while the work in [56] and [57] does not guarantee the ability of the composed RL agents to generalize to all unseen LTL tasks unless additional RL controllers are learned. On the contrary, the framework proposed in this paper is designed for systems with continuous state and action spaces. Moreover, this framework guarantees the ability to generalize to unseen LTL tasks without the need to train additional NNs at the expense of performing some computations when the unseen LTL task is revealed. Finally, the work reported in [59] offers insights on how to perform transfer learning in RL between different temporal logic missions for finite state and action spaces and is restricted to the case when the logical formulas are "close enough." Unlike all these works, our approach can be applied to systems with continuous state and control action spaces, capable of training NNs with guarantees that allow them to be composable, and can perform transfer learning between any temporal logic tasks while satisfying strong guarantees that are not enjoyed by current transfer learning algorithms [60].

The idea of learning neurosymbolic models is studied in works [61], [62], [63] that use NNs to guide the synthesis of control policies represented as short programs. The algorithms in [61], [62], and [63] train an NN controller, project it to the space of program languages, analyze the short programs, and lift the programs back to the space of NNs for further training. These works focus on tasks given during the training of NNs, and the final controller is a short program. Another related work is reported in [64] and [65], which studies the problem of extracting a finite-state controller from a recurrent NN. Unlike the above works, we consider temporal logic specifications and unseen tasks, and our final planner is NNs in tandem with a finite-state MDP.

II. Problem Formulation

A. Notations

Let $\mathbb{R}$, $\mathbb{R}^+$, and $\mathbb{N}$ be the set of real numbers, positive real numbers, and natural numbers, respectively. For a nonempty set S, let 2^S be the power set of S, $\mathbf{1}_S$ be the indicator function of S, and $\text{Int}(S)$ be the interior of S. Furthermore, we use S^n to denote the set of all finite sequences of length $n \in \mathbb{N}$ of elements in S. The product of two sets is defined as $S_1 \times S_2 := \{(s_1, s_2) | s_1 \in S_1, s_2 \in S_2\}$. Let $\|x\|$ be the Euclidean norm of a vector $x \in \mathbb{R}^n$, $\|A\|$ be the induced two-norm of a matrix $A \in \mathbb{R}^{m\times n}$, and $\|A\|_{\max} = \max_{i,j}|A_{ij}|$ be the max norm of a matrix A. Any Borel space X is assumed to be endowed with a Borel σ-algebra denoted by $\mathcal{B}(X)$.

B. Assumptions and Information Structure

We consider a meta-RL setting that aims to train NNs for controlling a robot to achieve tasks that were unseen during training. To be specific, we denote a task by a tuple $\mathcal{T} = (g, \varphi, \mathcal{W}, X_0)$, where g captures the error in the robot's dynamical model (see Section II-C), φ is a BLTL/scLTL formula that defines the mission for a robot to accomplish (see Section II-D), $\mathcal{W}$ is a workspace (or an environment) in which a robot operates, and X_0 contains the initials states of a robot. During training, we assume the availability of an approximation of the dynamical model t (see Section II-C for details). The mission specification φ, the workspace $\mathcal{W}$, and the set of initial states X_0 are not available during training and only become available at runtime. Despite the limited knowledge of tasks during training, we aim to design provably correct NNs for unseen tasks $\mathcal{T}$.

C. Dynamical Model

We consider robotic systems that can be modeled as stochastic, discrete-time, nonlinear dynamical systems with a transition probability of the form

$$\Pr(x' \in A | x, u) = \int_A t(dx' | x, u) \tag{1}$$

where states of a robot $x \in X$ and control actions $u \in U$ are from continuous state and action spaces $X \subset \mathbb{R}^n$ and $U \subset \mathbb{R}^m$, respectively. In (1), we use $t : \mathcal{B}(X) \times X \times U \to [0, 1]$ to denote a stochastic kernel that assigns to any state $x \in X$ and action $u \in U$ a probability measure $t(\cdot|x, u)$. Then, $\Pr(x' \in A|x, u)$ is the probability of reaching a subset $A \in \mathcal{B}(X)$ in one-time step from state $x \in X$ under action $u \in U$. We assume that t consists of a priori known nominal model f and a model error g capturing the unmodeled dynamics. We assume the model error g is captured by a Gaussian process (GP) regression model $\mathcal{GP}(\mu_g, \sigma_g^2)$, where μ_g and σ_g^2 are the posterior mean and variance functions, respectively [66]. Hence, we can rewrite (1) as

$$\Pr(x' \in A | x, u) = \int_A \mathcal{N}(dx' | f(x, u) + \mu_g(x, u), \sigma_g^2(x, u)) \tag{2}$$

which is an integral of the normal distribution $\mathcal{N}(f(x, u) + \mu_g(x, u), \sigma_g^2(x, u))$ and hence can be easily computed.

We assume the nominal model f is available during the NN training phase, while the model-error g is evaluated at runtime, and hence the exact stochastic kernel t only becomes available at runtime. This allows us to apply the trained NN to various

robotic systems with different dynamics captured by the model error g.

Remark: We note that our algorithm does not require the knowledge of the function f in a closed-form/symbolic representation. Access to a simulator would suffice.

D. Temporal Logic Specification and Workspace

A well-known weakness of RL and meta-RL algorithms is the difficulty in designing reward functions that capture the exact intent of designers [47], [48], [67]. Agent behavior that scores high according to a user-defined reward function may not be aligned with the user's intention, which is often referred to as "specification gaming" [68]. To that end, we adopt the representation of an agent's mission in temporal logic specifications, which have been extensively demonstrated the capability to capture complex behaviors of robotic systems.

In particular, we consider mission specifications defined in either BLTL [1] or syntactically scLTL [2]. Let AP be a finite set of atomic propositions that describe a robotic system's states with respect to a workspace $\mathcal{W}$. For example, these atomic propositions can describe the location of a robot with respect to the obstacles to avoid and the goal location to achieve. Given AP, any BLTL formula can be generated according to the following grammar:

$$\varphi := \sigma \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \, \mathcal{U}_{[k_1,k_2]} \, \varphi_2$$

where $\sigma \in AP$ and time steps $k_1 < k_2$. Given the above grammar, we can define $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, false $= \varphi \wedge \neg\varphi$, and true $= \neg$false. Furthermore, the bounded-time *eventually* operator can be derived as $\Diamond_{[k_1,k_2]}\varphi =$ true $\mathcal{U}_{[k_1,k_2]}\, \varphi$ and the bounded-time *always* operator is given by $\Box_{[k_1,k_2]}\varphi = \neg\Diamond_{[k_1,k_2]}\neg\varphi$.

Given a set of atomic propositions AP, the corresponding alphabet is defined as $\mathbb{A} := 2^{AP}$, and a finite (infinite) word ω is a finite (infinite) sequence of letters from the alphabet $\mathbb{A}$, i.e., $\omega = \omega^{(0)}\omega^{(1)}\ldots\omega^{(H)} \in \mathbb{A}^{H+1}$. The satisfaction of a word ω to a specification φ can be determined based on the semantics of BLTL [1]. Given a robotic system and an alphabet $\mathbb{A}$, let $L : X \to \mathbb{A}$ be a labeling function that assigns to each state $x \in X$ the subset of atomic propositions $L(x) \in \mathbb{A}$ that evaluate true at x. Then, a robotic system's trajectory ξ satisfies a specification φ, denoted by $\xi \models \varphi$, if the corresponding word satisfies φ, i.e., $L(\xi) \models \varphi$, where $\xi = x^{(0)}x^{(1)}\ldots x^{(H)} \in X^{H+1}$ and $L(\xi) = L(x^{(0)})L(x^{(1)})\ldots L(x^{(H)}) \in \mathbb{A}^{H+1}$. Similarly, we can consider scLTL specifications interpreted over infinite words based on the fact that any infinite word that satisfies a scLTL formula φ contains a finite "good" prefix such that all infinite words that contain the prefix satisfy φ [2].

Example 1 (Reach-avoid Specification): Consider a robot that navigates a workspace $\mathcal{W} = \{X_{\text{goal}}, O_1, \ldots, O_c\}$, where $X_{\text{goal}} \subset X$ is a set of goal states that the robot would like to reach and $O_1, \ldots, O_c \subset X$ are obstacles that the robot needs to avoid. The set of atomic propositions is given by $AP = \{x \in X_{\text{goal}}, x \in O_1, \ldots, x \in O_c\}$, where x is the state of the robot. Then, a reach-avoid specification can be expressed as $\varphi = \varphi_{\text{liveness}} \wedge \varphi_{\text{safety}}$, where $\varphi_{\text{liveness}} = \Diamond_{[0,H]}(x \in X_{\text{goal}})$ requires the robot to reach the goal X_{goal} in H time steps and $\varphi_{\text{safety}} = \Box_{[0,H]} \bigwedge_{i=1,\ldots,c} \neg(x \in O_i)$ specifies to avoid all the obstacles during the time horizon H. Let $\xi = x^{(0)}x^{(1)}\ldots x^{(H)}$ be a trajectory of the robot, then the reach-avoid specification φ is interpreted as

$$\xi \models \varphi_{\text{liveness}} \iff \exists k \in \{0, \ldots H\}, x^{(k)} \in X_{\text{goal}}$$

$$\xi \models \varphi_{\text{safety}} \iff \forall k \in \{0, \ldots H\}, \forall i \in \{1, \ldots, c\}, x^{(k)} \notin O_i.$$

E. Neural Network

To account for the stochastic behavior of a robot, we aim to design a state-feedback NN $\mathcal{NN} : X \to U$ that can achieve temporal motion and task specifications φ. An F-layer rectified linear unit (ReLU) NN is specified by composing F layer functions (or just layers). A layer l with $\mathfrak{i}_l$ inputs and $\mathfrak{o}_l$ outputs is specified by a weight matrix $W^{(l)} \in \mathbb{R}^{\mathfrak{o}_l \times \mathfrak{i}_l}$ and a bias vector $b^{(l)} \in \mathbb{R}^{\mathfrak{o}_l}$ as follows:

$$L^{\theta^{(l)}} : z \mapsto \max\{W^{(l)}z + b^{(l)}, 0\} \tag{3}$$

where the max function is taken element-wise, and $\theta^{(l)} \triangleq (W^{(l)}, b^{(l)})$ for brevity. Thus, an F-layer ReLU NN is specified by F layer functions $\{L^{\theta^{(l)}} : l = 1, \ldots, F\}$ whose input and output dimensions are composable: that is, they satisfy $\mathfrak{i}_l = \mathfrak{o}_{l-1}$, $l = 2, \ldots, F$. Specifically

$$\mathcal{NN}^{\theta}(x) = (L^{\theta^{(F)}} \circ L^{\theta^{(F-1)}} \circ \cdots \circ L^{\theta^{(1)}})(x) \tag{4}$$

where we index a ReLU NN function by a list of parameters $\theta \triangleq (\theta^{(1)}, \ldots, \theta^{(F)})$. As a common practice, we allow the output layer $L^{\theta^{(F)}}$ to omit the max function. For simplicity of notation, we drop the superscript θ in $\mathcal{NN}^{\theta}$ whenever the dependence on θ is obvious.

F. Main Problem

We consider training a finite set (or a library) of ReLU NNs (during the offline phase) and designing a selection algorithm (during the online phase) that can select the correct NNs once the exact task $\mathcal{T} = (g, \varphi, \mathcal{W}, X_0)$ is revealed at runtime. Before formalizing the problem under consideration, we introduce the following notion of NN composition.

Definition II.1: Given a set (or a library) of NNs $\mathfrak{NN} = \{\mathcal{NN}_1, \mathcal{NN}_2, \ldots, \mathcal{NN}_d\}$ along with an activation map $\Gamma : X \to \{1, \ldots, d\}$, the composed NN $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ is defined as: $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}(x) = \mathcal{NN}_{\Gamma(x)}(x)$.

In other words, the activation map Γ selects the NN that needs to be activated at each state $x \in X$. Let $\xi^x_{\mathcal{NN}_{[\mathfrak{NN},\Gamma]}}$ be a closed-loop trajectory of a robot that starts from the state $x \in X_0$ and evolves under the composed NN $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$. We define the problem of interest as follows.

Problem II.2: Given the nominal dynamics f, the state space X and the control space U, train a library of ReLU NNs $\mathfrak{NN}$ without the knowledge of the task $\mathcal{T} = (g, \varphi, \mathcal{W}, X_0)$. When the task $\mathcal{T} = (g, \varphi, \mathcal{W}, X_0)$ is revealed at runtime, compute an activation map Γ such that the composed NN $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ satisfies

the specification φ with probability at least p, i.e., $\Pr(\xi^x_{\mathcal{NN}_{[\mathfrak{MN},\Gamma]}} \models \varphi) \geq p$ for any $x \in X_0$.

G. Overview of the Neurosymbolic Framework

Our approach to designing the NN-based planner $\mathcal{NN}_{[\mathfrak{MN},\Gamma]}$ can be split into two stages: offline training and runtime selection. During the offline training phase, our algorithm obtains a library of networks $\mathfrak{MN}$. At runtime, and to fulfill *unseen* tasks using a *finite* set of NNs $\mathfrak{MN}$, our neurosymbolic framework bridges ideas from symbolic LTL-based planning and ML. Similar to symbolic LTL-based planning, our framework uses a hierarchical approach that consists of a "high-level" discrete planner and a "low-level" continuous controller [8], [9], [12]. The "high-level" discrete planner focuses on ensuring the satisfaction of the temporal logic specification. At the same time, the "low-level" controllers compute control actions that steer the robot to satisfy the "high-level" plan. Unlike symbolic LTL-based planners, our framework uses NNs as low-level controllers, thanks to their ability to handle complex nonlinear dynamic constraints. In particular, the "high-level" planner chooses the activation map Γ to activate particular NNs.

Nevertheless, to ensure the correctness of the proposed framework, it is essential to ensure that each NN in $\mathfrak{MN}$ satisfies some "formal" property. This "formal" property allows the high-level planner to abstract the capabilities of each of the NNs in $\mathfrak{MN}$ and hence choose the correct activation map Γ. To that end, in Section III, we formulate the subproblem of "formal NN training" that guarantees the trained NNs satisfy certain formal properties, and solve it efficiently by introducing a NN weight projection operator. The solution to the formal training is used in Section IV-A to obtain the library of networks $\mathfrak{MN}$ offline. The associated formal property of each NN is used in Section IV-B to design the activation map Γ.

III. Formal Training of NNs

In this section, we study the subproblem of training NNs that are guaranteed to obey certain behaviors. In addition to the classical gradient-descent update of NN weights, we propose a novel "projection" operator that ensures the resulting NN obeys the selected behavior. We provide a theoretical analysis of the proposed projection operator in terms of correctness and computational complexity.

A. Formulation of Formal Training

We start by recalling that every ReLU NN represents a CPWA function [15]. Let $\Psi_{\text{CPWA}} : X \to \mathbb{R}^m$ denote a CPWA function of the form

$$\Psi_{\text{CPWA}}(x) = K_i' x + b_i' \quad \text{if } x \in \mathcal{R}_i,\ i = 1, \ldots, L \tag{5}$$

where the collection of polytopic subsets $\{\mathcal{R}_1, \ldots, \mathcal{R}_L\}$ is a partition of the set $X \subset \mathbb{R}^n$ such that $\bigcup_{i=1}^L \mathcal{R}_i = X$ and $\text{Int}(\mathcal{R}_i) \cap \text{Int}(\mathcal{R}_j) = \emptyset$ if $i \neq j$. We call each polytopic subset $\mathcal{R}_i \subset X$ a linear region, and denote by $\mathbb{L}_{\Psi_{\text{CPWA}}}$ the set of linear regions associated to Ψ_{CPWA}, i.e., $\mathbb{L}_{\Psi_{\text{CPWA}}} = \{\mathcal{R}_1, \ldots, \mathcal{R}_L\}$. In this article, we confine our attention to CPWA controllers (and hence NN controllers) that are selected from a bounded polytopic set $\mathcal{P}^K \times \mathcal{P}^b \subset \mathbb{R}^{m \times n} \times \mathbb{R}^m$, i.e., we assume that $K_i' \in \mathcal{P}^K$ and $b_i' \in \mathcal{P}^b$. For simplicity of notation, we use $\mathcal{P}^{K \times b} \subset \mathbb{R}^{m \times (n+1)}$ to denote the polytopic set $\mathcal{P}^K \times \mathcal{P}^b$, and use $K_i(x)$ with a single parameter $K_i \in \mathcal{P}^{K \times b}$ to denote $K_i' x_i + b_i'$ with the pair $(K_i', b_i') = K_i$.

Let $\mathcal{P} \subseteq \mathcal{P}^{K \times b}$ be a bounded polytopic subset of the parameters K_i, then with some abuse of notation, we use the same notation $\mathcal{P}$ to denote the subset of CPWA functions whose parameters K_i are chosen from $\mathcal{P}$. In other words, a CPWA function $\Psi_{\text{CPWA}} \in \mathcal{P}$ if and only if $K_i \in \mathcal{P}$ at all linear regions $\mathcal{R}_i \in \mathbb{L}_{\Psi_{\text{CPWA}}}$, where the CPWA function Ψ_{CPWA} is in the form of (5).

Using this notation, we define the formal training problem that ensures the trained NNs belong to subsets of CPWA functions $\mathcal{P} \subseteq \mathcal{P}^{K \times b}$ as follows.

Problem III.1: Given a bounded polytopic subset $q \subseteq X$, a bounded subset of CPWA functions $\mathcal{P} \subseteq \mathcal{P}^{K \times b}$, and a cost functional J, find NN weights θ^* such that

$$\theta^* = \underset{\theta}{\textit{argmin}}\ J(\mathcal{NN}^\theta) \quad \textit{s.t.} \quad \mathcal{NN}^\theta|_q \in \mathcal{P}. \tag{6}$$

In Problem III.1, we use $\mathcal{NN}^\theta|_q$ to denote the restriction of $\mathcal{NN}^\theta$ to the subset q, i.e., $\mathcal{NN}^\theta|_q(x) = \mathcal{NN}^\theta(x)$ for $x \in q$. Consider the CPWA function $\mathcal{NN}^\theta$ is in the form of (5), then the constraint $\mathcal{NN}^\theta|_q \in \mathcal{P}$ requires that $K_i \in \mathcal{P}$ whenever the corresponding linear region $\mathcal{R}_i$ intersects the subset q, i.e.,

$$\mathcal{NN}^\theta|_q \in \mathcal{P} \iff K_i \in \mathcal{P} \quad \forall \mathcal{R}_i \in \{\mathcal{R} \in \mathbb{L}_{\mathcal{NN}^\theta} | \mathcal{R} \cap q \neq \emptyset\}. \tag{7}$$

Algorithm 1: Formal-Train $(q, \mathcal{P}, J)$.

1: Initialize neural network $\mathcal{NN}^\theta$, $i = 1$
2: **while** $i \leq$ max_iter **do**
3: $\quad \mathcal{NN}^\theta = \texttt{gradient} - \texttt{descent}(\mathcal{NN}^\theta, \mathcal{P}, J)$
4: $\quad \widehat{W}^{(F)}, \widehat{b}^{(F)} = \Pi_{\mathcal{P}}(\mathcal{NN}^\theta)$
5: $\quad$ Set the output layer weights of $\mathcal{NN}^\theta$ be $\widehat{W}^{(F)}, \widehat{b}^{(F)}$
6: $\quad i = i + 1$
7: **end while**
8: **Return** $\mathcal{NN}^\theta$

B. NN Weight Projection

To solve Problem III.1, we introduce a NN weight projection operator that can be incorporated into the training of NNs. Algorithm 1 outlines our procedure for solving Problem III.1. As a projected-gradient algorithm, Algorithm 1 alternates the gradient descent based training (line 3 in Algorithm 1) and the NN weight projection (lines 4 and 5 in Algorithm 1) up to a prespecified maximum iteration max_iter. Given a subset of CPWA functions $\mathcal{P} \subseteq \mathcal{P}^{K \times b}$, we denote by $\Pi_{\mathcal{P}}$ the NN weight projection operator that enforces a network $\mathcal{NN}^\theta$ to satisfy $\mathcal{NN}^\theta|_q \in \mathcal{P}$, i.e., the constraints (7). In the following, we formulate this NN weight projection operator $\Pi_{\mathcal{P}}$ as an optimization problem.

Consider a NN $\mathcal{NN}^\theta$ with F layers, including $F - 1$ hidden layers and an output layer. Let $W^{(F)}$ and $b^{(F)}$ be the weight

matrix and the bias vector of the output layer, respectively, i.e.,

$$\theta = \left(\theta^{(1)}, \ldots, \theta^{(F-1)}, \ (W^{(F)}, b^{(F)})\right). \tag{8}$$

Then, the NN weight projection $\Pi_{\mathcal{P}}$ updates the output layer weights $W^{(F)}, b^{(F)}$ to $\widehat{W}^{(F)}, \widehat{b}^{(F)}$ (lines 4 and 5 in Algorithm 1). As a result, the projected NN weights $\widehat{\theta}$ are given by

$$\widehat{\theta} = \left(\theta^{(1)}, \ldots, \theta^{(F-1)}, \ (\widehat{W}^{(F)}, \widehat{b}^{(F)})\right). \tag{9}$$

We formulate the NN weight projection operator $\Pi_{\mathcal{P}}$ as the following optimization problem:

$$\underset{\widehat{W}^{(F)}, \widehat{b}^{(F)}}{\operatorname{argmin}} \ \max_{x \in q} \|\mathcal{NN}^{\widehat{\theta}}(x) - \mathcal{NN}^{\theta}(x)\|_1 \tag{10}$$

$$\text{s.t. } \widehat{K}_i \in \mathcal{P}, \ \forall \mathcal{R}_i \in \{\mathcal{R} \in \mathbb{L}_{\mathcal{NN}^{\theta}} \mid \mathcal{R} \cap q \neq \emptyset\}. \tag{11}$$

In the constraints (11), we use $\widehat{K}_i$ to denote the affine function parameters of the CPWA function $\mathcal{NN}^{\widehat{\theta}}$.

The optimization problem (10) and (11) tries to minimize the change of the NN's outputs due to the weight projection, where the change is measured by the largest 1-norm difference between the outputs given by $\mathcal{NN}^{\widehat{\theta}}$ and $\mathcal{NN}^{\theta}$ across the subset $q \subseteq X$, i.e., $\max_{x \in q} \|\mathcal{NN}^{\widehat{\theta}}(x) - \mathcal{NN}^{\theta}(x)\|_1$. In the following two sections, we first upper bound the objective function (10) in terms of the change of the NN's weights, and then show that the optimization problem (10) and (11) can be solved efficiently.

C. Bounding the Change of Control Actions

First, we note that it is common to omit the ReLU activation functions from the NN's output layer. Since the proposed projection operator only modifies the output layer weights, it is straightforward to show that the NN weight projection operator does not affect the set of linear regions, i.e., $\mathbb{L}_{\mathcal{NN}^{\widehat{\theta}}} = \mathbb{L}_{\mathcal{NN}^{\theta}}$, but only updates the affine functions defined over these regions. The following proposition shows the relation between the change in the NN's outputs and the change made in the output layer weights. The proof of this proposition can be found in Appendix B.

Proposition III.2: Consider two F-layer NNs $\mathcal{NN}^{\theta}$ and $\mathcal{NN}^{\widehat{\theta}}$ where θ and $\widehat{\theta}$ are as defined in (8) and (9). Then, the largest difference in the NNs' outputs across a subset $q \subseteq X$ is upper bounded as follows:

$$\begin{aligned} &\max_{x \in q} \|\mathcal{NN}^{\widehat{\theta}}(x) - \mathcal{NN}^{\theta}(x)\|_1 \\ &\leq \max_{x \in \text{Vert}\left(\mathbb{L}_{\mathcal{NN}^{\theta} \cap q}\right)} \sum_{i=1}^{m} \sum_{j=1}^{\mathfrak{o}_{F-1}} |\Delta W_{ij}^{(F)}| h_j(x) + \sum_{i=1}^{m} |\Delta b_i^{(F)}|. \end{aligned} \tag{12}$$

In Proposition III.2, m is the dimension of the NN's output, $\Delta W_{ij}^{(F)}$ and $\Delta b_i^{(F)}$ are the (i,j)th and the ith entry of $\Delta W^{(F)} = \widehat{W}^{(F)} - W^{(F)}$ and $\Delta b^{(F)} = \widehat{b}^{(F)} - b^{(F)}$, respectively. With the notation of layer functions (3), we use a single function $h : \mathbb{R}^n \to \mathbb{R}^{\mathfrak{o}_{F-1}}$ to represent all the hidden layers, i.e., $h(x) = (L_{\theta^{(F-1)}} \circ L_{\theta^{(F-2)}} \circ \cdots \circ L_{\theta^{(1)}})(x)$, where $\mathfrak{o}_{F-1}$ is the number of neurons in the $(F-1)$-layer (the last hidden layer). Furthermore, we use $\mathbb{L}_{\mathcal{NN}^{\theta} \cap q}$ to denote the intersected regions between the linear regions in $\mathbb{L}_{\mathcal{NN}^{\theta}}$ and the subset $q \subseteq X$, i.e., $\mathbb{L}_{\mathcal{NN}^{\theta} \cap q} = \{\mathcal{R} \cap q | \mathcal{R} \in \mathbb{L}_{\mathcal{NN}^{\theta}}, \mathcal{R} \cap q \neq \emptyset\}$. Let $\text{Vert}(\mathcal{R})$ be the set of vertices of a region $\mathcal{R}$, then $\text{Vert}(\mathbb{L}_{\mathcal{NN}^{\theta} \cap q}) = \bigcup_{\mathcal{R} \in \mathbb{L}_{\mathcal{NN}^{\theta} \cap q}} \text{Vert}(\mathcal{R})$ is the set of vertices of all regions in $\mathbb{L}_{\mathcal{NN}^{\theta} \cap q}$.

D. Efficient Computation of the NN Projection Operator

Now, we focus on how to compute the NN weight projection operator $\Pi_{\mathcal{P}}$ efficiently. In particular, Proposition III.2 proposes a direct way to solve the intended projection operator. In order to minimize the change of the NN's outputs (10) due to the weight projection, we minimize its upper bound given by (12). Accordingly, we compute the NN weight projection operator $\Pi_{\mathcal{P}}$ by solving following optimization problem:

$$\underset{\widehat{W}^{(F)}, \widehat{b}^{(F)}}{\operatorname{argmin}} \ \max_{x \in \text{Vert}\left(\mathbb{L}_{\mathcal{NN}^{\theta} \cap q}\right)} \sum_{i=1}^{m} \sum_{j=1}^{\mathfrak{o}_{F-1}} |\Delta W_{ij}^{(F)}| h_j(x) + \sum_{i=1}^{m} |\Delta b_i^{(F)}| \tag{13}$$

$$\text{s.t. } \widehat{K}_i \in \mathcal{P}, \ \forall \mathcal{R}_i \in \{\mathcal{R} \in \mathbb{L}_{\mathcal{NN}^{\theta}} \mid \mathcal{R} \cap q \neq \emptyset\}. \tag{14}$$

The next result establishes the computational complexity of solving the optimization problem above. The proof of the proposition is given in Appendix B.

Proposition III.3: The optimization problem (13) and (14) is a linear program.

While Proposition III.3 ensures that solving the optimization problem can be done efficiently, we note that identifying the set of linear regions $\mathbb{L}_{\mathcal{NN}^{\theta}}$ of a ReLU NN $\mathcal{NN}^{\theta}$ needs to enumerate the hyperplanes represented by $\mathcal{NN}^{\theta}$. For shallow NNs and other special NN architectures, this can be done in polynomial time (e.g., Ferlez and Shoukry [69] used a POSET for the enumeration). For general NNs, identifying linear regions may not be polynomial time, but there exist efficient tools such as NNENUM [70] that use star sets to enumerate all the linear regions. Moreover, as we will show in the following sections, each NN in the library $\mathfrak{NN}$ can contain a limited number of weights (and hence a limited number of linear regions), but their combination leads to NNs with a large number of linear regions and hence capable of implementing complex functions.

We conclude this section with the following result whose proof follows directly from Proposition III.3 and the equivalence in (7).

Theorem III.4: Given a bounded polytopic subset $q \subseteq X$ and a bounded subset of CPWA functions $\mathcal{P} \subseteq \mathcal{P}^{K \times b}$. Consider a NN $\mathcal{NN}^{\theta}$ whose output layer weights are given by the NN weight projection operator $\Pi_{\mathcal{P}}$ (i.e., the solution to (13) and (14)). Then, the network $\mathcal{NN}^{\theta}$ satisfies the constraint in (6), i.e., $\mathcal{NN}^{\theta}|_q \in \mathcal{P}$. Furthermore, the optimization problem (13) and (14) is a linear program.

IV. Neurosymbolic Learning Framework

As discussed in Section II-G, our approach to designing the NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ and solving Problem II.2 is split into two stages: offline training and runtime selection. During

the offline training phase, our algorithm obtains a library of networks $\mathfrak{NN}$, where each NN is trained using the formal training Algorithm 1. At runtime, when the exact task $\mathcal{T}=(g,\varphi,\mathcal{W},X_0)$ is observed, we use dynamic programming (DP) to compute an activation map Γ, which selects a subset of the trained NNs and combines them into a single planner. We provide details on these two stages in the following two sections separately.

A. *Offline Training of a Library* $\mathfrak{NN}$

Similar to standard LTL-based motion planners [9], [10], [11], [12], [13], [14], we partition the continuous state space $X\subset\mathbb{R}^n$ into a finite set of abstract states $\mathbb{X}=\{q_1,\ldots,q_N\}$, where each abstract state $q_i\in\mathbb{X}$ is an infinity-norm ball in $\mathbb{R}^n$ with a prespecified diameter $\eta_q\in\mathbb{R}^+$ (Theorem V.2 captures the effect of η_q on the correctness of the obtained controller and Section VI-C discusses how to choose η_q). The partitioning satisfies $X=\bigcup_{q\in\mathbb{X}}q$ and $\text{Int}(q_i)\cap\text{Int}(q_j)=\emptyset$ if $i\neq j$. Let $\text{abs}:X\to\mathbb{X}$ map a state $x\in X$ to the abstract state $\text{abs}(x)\in\mathbb{X}$ that contains x, i.e., $x\in\text{abs}(x)$, and $\text{ct}_{\mathbb{X}}:\mathbb{X}\to X$ map an abstract state $q\in\mathbb{X}$ to its center $\text{ct}_{\mathbb{X}}(q)\in X$, which is well-defined since abstract states are inifinity-norm balls. With some abuse of notation, we denote by q both an abstract state, i.e., $q\in\mathbb{X}$, and a subset of states, i.e., $q\subseteq X$.

As mentioned in the above section, we consider CPWA controllers (and hence NN controllers) selected from a bounded polytopic set (namely a controller space) $\mathcal{P}^{K\times b}\subset\mathbb{R}^{m\times(n+1)}$. We partition the controller space $\mathcal{P}^{K\times b}\subset\mathbb{R}^{m\times(n+1)}$ into a finite set of controller partitions $\mathbb{P}=\{\mathcal{P}_1,\ldots,\mathcal{P}_M\}$ with a prespecified grid size $\eta_{\mathcal{P}}\in\mathbb{R}^+$ (Theorem V.2 captures the effect of $\eta_{\mathcal{P}}$ on the correctness of the obtained controller and Section VI-C discusses how to choose $\eta_{\mathcal{P}}$). Each controller partition $\mathcal{P}_i\in\mathbb{P}$ is an infinity-norm ball centered around some $K_i\in\mathcal{P}^{K\times b}$ such that $\mathcal{P}^{K\times b}=\bigcup_{\mathcal{P}\in\mathbb{P}}\mathcal{P}$ and $\text{Int}(\mathcal{P}_i)\cap\text{Int}(\mathcal{P}_j)=\emptyset$ if $i\neq j$. Let $\text{ct}_{\mathbb{P}}:\mathbb{P}\to\mathcal{P}^{K\times b}$ map a controller partition $\mathcal{P}\in\mathbb{P}$ to its center $\text{ct}_{\mathbb{P}}(\mathcal{P})\in\mathcal{P}^{K\times b}$. As mentioned in Section III-A, we use the same notation $\mathcal{P}$ to denote both a subset of the parameters $K_i\in\mathcal{P}^{K\times b}$ and a subset of CPWA functions whose parameters K_i are chosen from $\mathcal{P}$.

Algorithm 2 outlines the training of a library of NNs $\mathfrak{NN}$. Without knowing the exact robot dynamics (i.e., the stochastic kernel t), the workspace $\mathcal{W}$, and the specification φ, we use the formal training Algorithm 1 to train one NN $\mathcal{NN}^{\theta}_{(q_j,\mathcal{P}_i)}$ corresponding to each combination of controller partitions $\mathcal{P}_i\in\mathbb{P}$ and abstract states $q_j\in\mathbb{X}$ (line 5 in Algorithm 2). Thanks to the NN weight projection operator $\Pi_{\mathcal{P}}$, the NNs $\mathcal{NN}^{\theta}_{(q_j,\mathcal{P}_i)}$ satisfy the constraint in (6), i.e., $\mathcal{NN}^{\theta}_{(q_j,\mathcal{P}_i)}|_{q_j}\in\mathcal{P}$. In the following, we use the notation $\mathcal{NN}_{(q,\mathcal{P})}$ by dropping the superscript θ for simplicity and refer to each NN $\mathcal{NN}_{(q_j,\mathcal{P}_i)}$ a local network.

To train the local NNs, we implement the training approach $\texttt{gradient}-\texttt{descent}$ (line 3 in Algorithm 1) based on proximal policy optimization (PPO) [71] with the reward function as follows:

$$r_i(x,u)=-w_1c(x,u)-w_2\|u-\kappa(\mathcal{P}_i)\| \tag{15}$$

where $w_1,w_2\in\mathbb{R}^+$ are prespecified weights. The function $c:X\times U\to\mathbb{R}$ is a user-defined state-action cost function. As an example, the cost function c can be the controller's energy $\|u\|^2$ or a loss function $\mathcal{L}(u,y)$ that measures the difference between the controller's output u and some desired output y (e.g., imitation learning). The second term in (15) motivates the NN to select control signals that are within the controller partition $\mathcal{P}_i$ by penalizing the difference between the control signals u and the center of the assigned controller partition $\mathcal{P}_i\in\mathbb{P}$, denoted by $\kappa(\mathcal{P}_i)=\text{ct}_{\mathbb{P}}(\mathcal{P}_i)$. Using the reward function r_i in (15), one can define the NN training cost functional J_i as the discounted, average reward over the trajectories of the dynamical system [71].

Algorithm 2: TRAIN-LIBRARY-NNS $(\mathbb{X},\mathbb{P},w_1,w_2,c(x,u))$.

1: $\mathfrak{NN}=\{\}$
2: **for** $q_j\in\mathbb{X}$ **do**
3: **for** $\mathcal{P}_i\in\mathbb{P}$ **do**
4: Construct the reward function r_i as in (15) and the corresponding J_i
5: $\mathcal{NN}_{(q_j,\mathcal{P}_i)}=\texttt{Formal}-\texttt{Train}(q_j,\mathcal{P}_i,J_i)$
6: $\mathfrak{NN}=\mathfrak{NN}\cup\{\mathcal{NN}_{(q_j,\mathcal{P}_i)}\}$
7: **end for**
8: **end for**
9: **Return** $\mathfrak{NN}$

We assume access to a "nominal" simulator (i.e., the nominal dynamics f in (2)) for updating the robot states. Algorithm 2 returns a library $\mathfrak{NN}$ of $M\times N$ local networks, where M and N are the number of abstract states and the number of controller partitions, respectively. In Section VI, we reduce the number of local networks that need to be trained by employing transfer learning.

B. *Runtime Selection of Local NNs*

In this section, we present our selection algorithm used at runtime when an arbitrary task $\mathcal{T}=(g,\varphi,\mathcal{W},X_0)$ is given. The selection algorithm assigns one local NN in the set $\mathfrak{NN}$ to each abstract state $\{q_1,\ldots,q_N\}$ in order to satisfy the given specification φ. Given a stochastic kernel t (composed of nominal dynamics f and model error g), our algorithm first computes a finite-state MDP that captures the closed-loop behavior of the robot under *all* possible CPWA controllers. Transitions in this finite-state MDP correspond to different subsets of CPWA functions in $\mathbb{P}=\{\mathcal{P}_1,\ldots,\mathcal{P}_M\}$. Thanks to the fact that the NNs in the library $\mathfrak{NN}$ were trained using the formal training algorithm (Algorithm 1), *each NN now represents a transition (symbol) in the finite-state MDP*. In other words, although NNs are hard to interpret due to their construction, the formal training algorithm ensures the one-to-one mapping between these black-box NNs and the transitions in the finite-state symbolic model.

Next, we use standard techniques in LTL-based motion planning to construct a finite-state automaton that captures the satisfaction of mission specifications φ. By analyzing the product between the finite-state MDP (that abstracts the robot dynamics) and the automaton corresponding to the specification φ, our algorithm decides which local networks in the set $\mathfrak{NN}$ need

to be activated. We present details on the selection algorithm in the three steps below.

Step 1. Compute symbolic model: We construct a finite-state MDP $\hat{\Sigma} = (\mathbb{X}, \mathbb{X}_0, \mathbb{P}, \hat{t})$ of the robotic system $\Sigma = (X, X_0, U, t)$ as follows:

1) $\mathbb{X} = \{q_1, \ldots, q_N\}$ is the set of abstract states;
2) $\mathbb{X}_0 = \{q \in \mathbb{X} \mid q \subseteq X_0\}$ is the set of initial states;
3) $\mathbb{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$ is the set of controller partitions;
4) The transition probability from state $q \in \mathbb{X}$ to state $q' \in \mathbb{X}$ with label $\mathcal{P} \in \mathbb{P}$ is given by

$$\hat{t}(q'|q, \mathcal{P}) = \int_{q'} t(dx'|z, \kappa(z)) \tag{16}$$

where $z = \text{ct}_{\mathbb{X}}(q)$ and $\kappa = \text{ct}_{\mathbb{P}}(\mathcal{P})$.

As explained in Section II-C, the integral (16) can be easily computed since the stochastic kernel $t(\cdot|x, u)$ is a normal distribution, and we show techniques to accelerate the construction of the symbolic model $\hat{\Sigma}$ in Section VI. Such finite symbolic models have been used heavily in state-of-the-art LTL-based controller synthesis. Nevertheless, and unlike state-of-the-art LTL-based controllers, the control alphabet in $\hat{\Sigma}$ is *controller partitions* (i.e., subsets of CPWA functions). This is in contrast to LTL-based controllers in the literature (e.g., [13], [14]) that use subsets of control signals as their control alphabet.

We emphasize that our trained NN controllers are used to control the robotic system Σ with continuous state and action spaces, and the theoretical guarantees that we provide in Section V are also for the robotic system Σ, not for the finite-state MDP $\hat{\Sigma}$. As the motivation to introduce the symbolic model $\hat{\Sigma}$, our approach provides correctness guarantees for the NN-controlled robotic system Σ through 1) analyzing the behavior of the finite-state MDP $\hat{\Sigma}$ (in this section), and 2) bounding the difference in behavior between the finite-state MDP $\hat{\Sigma}$ and the NN-controlled robotic system Σ (in Section V). Critical to the latter step is the ability to restrict the NN's behavior thanks to the formal training proposed in Section III.

Step 2. Construct product MDP: Given a mission specification φ encoded in BLTL or scLTL formula, we construct the equivalent deterministic finite-state automaton (DFA) $\mathcal{A}_\varphi = (S, S_0, \mathbb{A}, G, \delta)$ as follows:

1) S is a finite set of states;
2) $S_0 \subseteq S$ is the set of initial states;
3) $\mathbb{A}$ is an alphabet;
4) $G \subseteq S$ is the accepting set;
5) $\delta : S \times \mathbb{A} \to S$ is a transition function.

Such translation of BLTL and scLTL specifications to the equivalent DFA can be done using off-the-shelf tools (e.g., [72] and [73]).

Given the finite-state MDP capturing the robot dynamics $\hat{\Sigma} = (\mathbb{X}, \mathbb{X}_0, \mathbb{P}, \hat{t})$ and the DFA $\mathcal{A}_\varphi = (S, S_0, \mathbb{A}, G, \delta)$ of the mission specification φ, we construct the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi = (\mathbb{X}^\otimes, \mathbb{X}_0^\otimes, \mathbb{P}, \mathbb{X}_G^\otimes, \hat{t}^\otimes)$ as follows:

1) $\mathbb{X}^\otimes = \mathbb{X} \times S$ is a finite set of states;
2) $\mathbb{X}_0^\otimes = \{(q_0, \delta(s_0, \hat{L}(q_0)))|q_0 \in \mathbb{X}_0, s_0 \in S_0\}$ is the set of initial states, where $\hat{L} : \mathbb{X} \to \mathbb{A}$ is the labeling function that assigns to each abstract state $q \in \mathbb{X}$ the subset of atomic propositions $\hat{L}(q) \in \mathbb{A}$ that evaluate $true$ at q;
3) $\mathbb{P}$ is the set of controller partitions;
4) $\mathbb{X}_G^\otimes = \mathbb{X} \times G$ is the accepting set;
5) The transition probability from state $(q, s) \in \mathbb{X}^\otimes$ to state $(q', s') \in \mathbb{X}^\otimes$ under $\mathcal{P} \in \mathbb{P}$ is given by:

$$\hat{t}^\otimes(q', s'|q, s, \mathcal{P}) = \begin{cases} \hat{t}(q'|q, \mathcal{P}) & \text{if } s' = \delta(s, \hat{L}(q')) \\ 0 & \text{else.} \end{cases}$$

Step 3. Select local NNs by DP: Once constructed the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$, the next step is to assign one local network $\mathcal{NN}_{(q,\mathcal{P})} \in \mathfrak{NN}$ to each abstract state $q \in \mathbb{X}$. In particular, the selection of NNs aims to maximize the probability of the finite-state MDP $\hat{\Sigma}$ satisfying the given specification φ. This can be formulated as finding the optimal policy that maximizes the probability of reaching the accepting set $\mathbb{X}_G^\otimes$ in the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$. To that end, we define the optimal value functions $\hat{V}_k^* : \mathbb{X}^\otimes \to [0, 1]$ that map a state $(q, s) \in \mathbb{X}^\otimes$ to the maximum probability of reaching the accepting set $\mathbb{X}_G^\otimes$ in $H - k$ steps from the state (q, s). When $k = 0$, the optimal value function $\hat{V}_0^*$ yields the maximum probability of reaching the accepting set $\mathbb{X}_G^\otimes$ in H steps, i.e., the maximum probability of $\hat{\Sigma}$ satisfying φ. The optimal value functions can be solved by the following DP recursion:

$$\hat{Q}_k(q, s, \mathcal{P}) = \mathbf{1}_G(s) + \mathbf{1}_{S\setminus G}(s) \sum_{(q',s') \in \mathbb{X}^\otimes} \hat{V}_{k+1}^*(q', s') \hat{t}^\otimes(q', s'|q, s, \mathcal{P}) \tag{17}$$

$$\hat{V}_k^*(q, s) = \max_{\mathcal{P} \in \mathbb{P}} \hat{Q}_k(q, s, \mathcal{P}) \tag{18}$$

with the initial condition $\hat{V}_H^*(q, s) = \mathbf{1}_G(s)$ for all $(q, s) \in \mathbb{X}^\otimes$, where $k = H - 1, \ldots, 0$. The recursion in (17) and (18) is solved for all the states $(q, s) \in \mathbb{X}^\otimes$ and controller partitions $\mathcal{P} \in \{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$.

Algorithm 3 summarizes the above three steps for selecting local NNs. Given a task $\mathcal{T} = (g, \varphi, \mathcal{W}, X_0)$ at runtime, Algorithm 3 first computes the symbolic model $\hat{\Sigma}$ based on the stochastic kernel t, translates the mission specification φ to a DFA $\mathcal{A}_\varphi$ using off-the-shelf tools, and constructs the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$ (lines 1–3 in Algorithm 3). Then, Algorithm 3 solves the optimal policy for the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$ using the DP recursion (17) and (18) (lines 4–20 in Algorithm 3). At time step k, the optimal controller partition $\mathcal{P}^*$ at state $(q, s) \in \mathbb{X}^\otimes$ is given by the maximizer of $\hat{Q}_k(q, s, \mathcal{P})$ (line 16 in Algorithm 3). The last step is to assign a corresponding NN to be applied given the robot states $x \in X$ and the DFA states $s \in S$. To that end, let

$$\Gamma_k(x, s) = \hat{\Gamma}_k(\text{abs}(x), s)$$

where $\hat{\Gamma}_k$ maps the product MDP's states $(q, s) \in \mathbb{X}^\otimes$ to NN's indices $(q, \mathcal{P}^*)$ (line 17 in Algorithm 3). In other words, given the robot states $x \in X$ and the DFA states $s \in S$ at time step k, we first find the abstract state $q \in \mathbb{X}$ that contains x, i.e., $q = \text{abs}(x)$, and then use the NN $\mathcal{NN}_{(q,\mathcal{P}^*)} \in \mathfrak{NN}$ to control the robot at x, where $\hat{\Gamma}_k(q, s) = (q, \mathcal{P}^*)$. Recall that the NNs

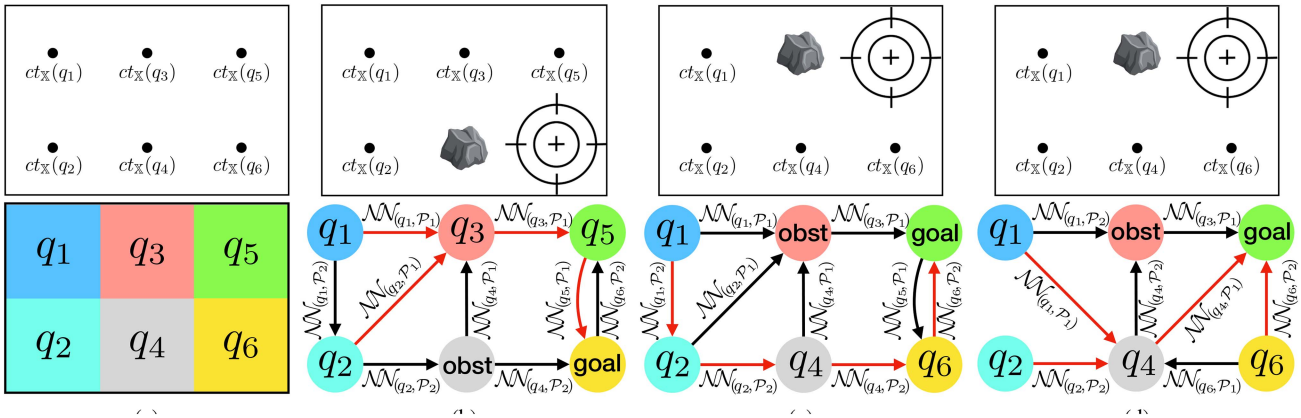


Fig. 1. Toy example of a robot that navigates a 2-D workspace and needs to satisfy reach-avoid specifications $\varphi = \varphi_{\text{liveness}} \wedge \varphi_{\text{safety}}$ (see more details in Section IV-C).

Algorithm 3: Runtime-Select $(\mathcal{T} = (g, \varphi, \mathcal{W}, X_0))$.

Compute the symbolic model $\hat{\Sigma} = (\mathbb{X}, \mathbb{X}_0, \mathbb{P}, t)$
Translate φ to a DFA $\mathcal{A}_\varphi = (S, S_0, \mathbb{A}, G, \delta)$
Construct the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$
for $(q, s) \in \mathbb{X}^\otimes$ **do**
$\hat{V}^*_H(q, s) = \mathbf{1}_G(s)$
end for
$k = H - 1$
while $k \geq 0$ **do**
  for $(q, s) \in \mathbb{X}^\otimes$ and $\mathcal{P} \in \{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$ **do**
    if $s \in G$ **then**
      $\hat{Q}_k(q, s, \mathcal{P}) = 1$
    else
      $\hat{Q}_k(q, s, \mathcal{P}) = \sum_{(q', s') \in \mathbb{X}^\otimes} \hat{V}^*_{k+1}(q', s') \hat{t}^\otimes(q', s' | q, s, \mathcal{P})$
    end if
    $\hat{V}^*_k(q, s) = \max_{\mathcal{P} \in \mathbb{P}} \hat{Q}_k(q, s, \mathcal{P})$
    $\mathcal{P}^* = \operatorname{argmax}_{\mathcal{P} \in \mathbb{P}} \hat{Q}_k(q, s, \mathcal{P})$
    $\hat{\Gamma}_k(q, s) = (q, \mathcal{P}^*)$
  end for
  $k = k - 1$
end while
Return $\{\hat{\Gamma}_k\}_{k \in \{0, \ldots, H-1\}}, \hat{V}^*_0, \hat{\Sigma} \otimes \mathcal{A}_\varphi$

in $\mathfrak{MN}$ are indexed as $(q, \mathcal{P})$ and hence the function $\Gamma(x, s) = \hat{\Gamma}_k(\text{abs}(x), s)$ computes such indices.

C. Toy Example

We conclude this section by providing a toy example in Fig. 1. Consider a mobile robot that navigates a 2-D workspace. We partition the state space $X \subset \mathbb{R}^2$ into six abstract states $\mathbb{X} = \{q_1, \ldots, q_6\}$ and discretize the controller space $\mathcal{P}^{K \times b}$ into two controller partitions $\mathbb{P} = \{\mathcal{P}_1, \mathcal{P}_2\}$. Fig. 1(a) shows the state space (top) and the abstract states $q_1, \ldots, q_6$ resulted from the partitioning (bottom), where the centers of abstract states are $\text{ct}_{\mathbb{X}}(q_1), \ldots, \text{ct}_{\mathbb{X}}(q_6)$.

During the offline training (Section IV-A), we use the formal training Algorithm 1 to obtain a library $\mathfrak{MN}$ consisting of 12 NNs, i.e., $\mathfrak{MN} = \{\mathcal{NN}_{(q_i, \mathcal{P}_j)} | i \in \{1, \ldots, 6\}, j \in \{1, 2\}\}$.

We consider three different tasks $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ that only become available at runtime after all the NNs in $\mathfrak{MN}$ have been trained. Fig. 1(b)–(d) shows the workspaces for these three tasks, respectively. The specifications for these three tasks are $\varphi_1 = \Diamond_{[0,3]}(x \in q_6) \wedge \Box_{[0,3]} \neg (x \in q_4)$, $\varphi_2 = \Diamond_{[0,4]}(x \in q_5) \wedge \Box_{[0,4]} \neg (x \in q_3)$, and $\varphi_3 = \Diamond_{[0,3]}(x \in q_5) \wedge \Box_{[0,3]} \neg (x \in q_3)$, respectively. Finally, the three tasks have different robot dynamics t. Fig. 1(b)–(d) also depicts the transitions in the resulting symbolic models, where we assume that all the transition probabilities $\hat{t}$ are 1 for simplicity [the transition probabilities $\hat{t}$ are computed as the integral of t in (16)]. Thanks to the formal training Algorithm 1, the NNs in $\mathfrak{MN}$ are guaranteed to be members of the CPWA functions in $\{\mathcal{P}_1, \mathcal{P}_2\}$. Hence, we label the transitions in the MDPs in Fig. 1(b)–(d) using $\mathcal{NN}_{(q_i, \mathcal{P}_j)}$ instead of $\{\mathcal{P}_1, \mathcal{P}_2\}$. While the transitions in the MDPs in Fig. 1(b) and (c) are the same, the MDP in Fig. 1(d) is different from that in Fig. 1(b) and (c) due to the difference in the robot dynamics in this task.

When the tasks $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ become available, we use the runtime selection algorithm (Algorithm 3) to obtain the selection functions Γ_k. In Fig. 1(b)–(d), the selected NNs are the labels of the transitions marked in red. For example, in Fig. 1(b), our algorithm selects $\mathcal{NN}_{(q_1, \mathcal{P}_1)}$ to be used at all states $x \in q_1$. It is clear from the figures that the selected NNs are guaranteed to satisfy the given specifications φ_1, φ_2, and φ_3, respectively, regardless of the difference in the workspaces and robot dynamics.

V. Theoretical Guarantees

In this section, we study the theoretical guarantees of the proposed approach. We first provide a probabilistic guarantee for our NN-based planners on satisfying mission specifications given at runtime, then bound the difference between the NN-based planner and the optimal controller that maximizes the

probability of satisfying the given specifications. The proof of the theoretical guarantees (Theorems V.1 and V.2) can be found in Appendix A.

A. Generalization to Unseen Tasks

For an arbitrary task $\mathcal{T} = (g, \varphi, \mathcal{W}, X_0)$, let $\mathcal{NN}_{[\mathfrak{MN},\Gamma]}$ be the corresponding NN-based planner, where the library of networks $\mathfrak{MN}$ is trained by Algorithm 2 without knowing the task $\mathcal{T}$, and the activation map Γ denotes the time-dependent functions Γ_k obtained from Algorithm 3. As a key feature of $\mathcal{NN}_{[\mathfrak{MN},\Gamma]}$, the activation map Γ selects NNs based on both the robot states and the states of the $\mathcal{A}_\varphi$ DFA. This allows the NN-based planner $\mathcal{NN}_{[\mathfrak{MN},\Gamma]}$ to take into account the specification φ by tracking states of the DFA $\mathcal{A}_\varphi$. In comparison, a single state-feedback NN $\mathcal{NN}: X \to U$ is not able to track the DFA states and hence cannot be trained to satisfy BLTL or scLTL specifications in general.

We denote by $\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{MN},\Gamma]}}$ the closed-loop trajectory of a robot under the NN-based planner $\mathcal{NN}_{[\mathfrak{MN},\Gamma]}$ with the robot starting from state $x \in X_0$ and the DFA $\mathcal{A}_\varphi$ starting from state $s \in S_0$. Notice that though the symbolic model $\hat{\Sigma}$ is a finite-state MDP, the NN-based planner $\mathcal{NN}_{[\mathfrak{MN},\Gamma]}$ is used to control the robotic system Σ with continuous state and action spaces. The following theorem provides a probabilistic guarantee for the NN-controlled robotic system to satisfy mission specifications given at runtime.

Theorem V.1: Let $\hat{V}_0^*$ be the optimal value function returned by Algorithm 3. For arbitrary states $x \in X_0$ and $s \in S_0$, the probability of the closed-loop trajectory $\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{MN},\Gamma]}}$ satisfying the given mission specification φ is bounded as follows:

$$\left|\Pr\left(\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{MN},\Gamma]}} \models \varphi\right) - \hat{V}_0^*(q,s)\right| \leq HZ\Delta^{\mathcal{NN}} \tag{19}$$

where $q = \text{abs}(x)$ and

$$\Delta^{\mathcal{NN}} = \max_{i\in\{1,\ldots,N\}} \left(\Lambda_i \eta_q + B_i L_i \eta_q + \sqrt{m(n+1)}\mathcal{L}_X B_i \eta_{\mathcal{P}}\right). \tag{20}$$

Recall that η_q and $\eta_{\mathcal{P}}$ are the grid sizes used for partitioning the state space and the controller space, respectively. The upper bound $HZ\Delta^{\mathcal{NN}}$ in Theorem V.1 can be arbitrarily small by tuning the grid sizes η_q and $\eta_{\mathcal{P}}$. In (19) and (20), H is the time horizon, $N = |\mathbb{X}|$ is the number of abstract states, and $Z = |S|$ is the number of the $\mathcal{A}_\varphi$ DFA states. The parameters Λ_i and B_i are given by $\Lambda_i = \int_X \lambda_i(y)\mu(dy)$ and $B_i = \int_X \beta_i(y)\mu(dy)$, where $\lambda_i(y)$ and $\beta_i(y)$ are the Lipschitz constants of the stochastic kernel $t : \mathcal{B}(X) \times X \times U \to [0,1]$, i.e. $\forall x, x' \in q_i\ \forall u \in U$

$$|t(dy|x',u) - t(dy|x,u)| \leq \lambda_i(y)\|x' - x\|\mu(dy)$$

and $\forall x \in q_i\ \forall u, u' \in U$

$$|t(dy|x,u') - t(dy|x,u)| \leq \beta_i(y)\|u' - u\|\mu(dy).$$

Furthermore, L_i is the Lipschitz constant of the local NNs at abstract state $q_i \in \mathbb{X}$, i.e., $\forall \mathcal{P} \in \mathbb{P}\ \forall x, x' \in q_i$:

$$\|\mathcal{NN}_{(q_i,\mathcal{P})}(x) - \mathcal{NN}_{(q_i,\mathcal{P})}(x')\| \leq L_i\|x - x'\|.$$

Finally, $\sup_{x\in X}\|x\| \leq \mathcal{L}_X$, $\sup_{K\in\mathcal{P}^{\kappa\times b}}\|K\| \leq \mathcal{L}_{\mathcal{P}}$, and n, m are the dimensions of $X \subset \mathbb{R}^n$, $U \subset \mathbb{R}^m$, respectively.

B. Optimality Guarantee

Next, we compare our NN-based planner $\mathcal{NN}_{[\mathfrak{MN},\Gamma]}$ with the optimal controller (not necessarily a NN) that maximizes the probability of satisfying the given specification φ. To that end, we provide an upper bound on the difference in the probabilities of satisfying φ without explicit computing of the optimal controller. Let $\mathcal{C}^*_\varphi : X \times S \to U$ be the optimal controller and $\xi^{(x,s)}_{\mathcal{C}^*_\varphi}$ be the closed-loop trajectory of the robotic system $\Sigma = (X, X_0, U, t)$ controlled by $\mathcal{C}^*_\varphi$. Similar to the NN-based planner $\mathcal{NN}_{[\mathfrak{MN},\Gamma]}$, the optimal controller $\mathcal{C}^*_\varphi$ applies to the robotic system Σ with continuous state and action spaces, and takes the DFA states $s \in S$ into consideration when computing control actions. Synthesizing the optimal controller $\mathcal{C}^*_\varphi$ for a mission specification φ is computationally prohibitive due to the continuous state and action spaces. Without explicitly computing $\mathcal{C}^*_\varphi$, the following theorem tells how close our NN-based planner $\mathcal{NN}_{[\mathfrak{MN},\Gamma]}$ is to the optimal controller $\mathcal{C}^*_\varphi$ in terms of satisfying the specification φ. By tuning the grid sizes η_q and $\eta_{\mathcal{P}}$, our NN-based planner $\mathcal{NN}_{[\mathfrak{MN},\Gamma]}$ can be arbitrarily close to the optimal controller $\mathcal{C}^*_\varphi$.

Theorem V.2: For arbitrary states $x \in X_0$ and $s \in S_0$, the difference in the probabilities of the closed-loop trajectories $\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{MN},\Gamma]}}$ and $\xi^{(x,s)}_{\mathcal{C}^*_\varphi}$ satisfying the given mission specification φ is upper bounded as follows:

$$\left|\Pr\left(\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{MN},\Gamma]}} \models \varphi\right) - \Pr\left(\xi^{(x,s)}_{\mathcal{C}^*_\varphi} \models \varphi\right)\right| \leq HZ(\Delta^{\mathcal{NN}} + \Delta^*) \tag{21}$$

where $\Delta^{\mathcal{NN}}$ is given by (20) and

$$\Delta^* = \max_{i\in\{1,\ldots,N\}} \left(\Lambda_i \eta_q + B_i \mathcal{L}_{\mathcal{P}} \eta_q + 2\sqrt{m(n+1)}\mathcal{L}_X B_i \eta_{\mathcal{P}}\right). \tag{22}$$

VI. Effective Adaptation

In this section, we focus on practical issues of the proposed approach and present some key elements for performance improvement while maintaining the same theoretical guarantees as Section V. First, we show that the proposed composition of NNs leads to an effective way to adapt previous learning experiences to unseen tasks. In particular, instead of training the whole library of NNs $\mathfrak{MN}$ in Algorithm 2, we only train a subset of networks $\mathfrak{MN}_{\text{part}} \subseteq \mathfrak{MN}$ based on tasks provided for training. Obtaining this subset $\mathfrak{MN}_{\text{part}}$ can be viewed as a systematic way to store learning experiences, which are adapted to unseen tasks via transfer learning (see Section VI-A). Second, we propose a data-driven approach to accelerate the construction of the symbolic model $\hat{\Sigma}$ (see Section VI-B). Finally, we comment on the choice of grid sizes η_q and $\eta_{\mathcal{P}}$ for partitioning the state and action spaces (see Section VI-C).

A. Accelerate by Transfer Learning

Consider a meta-RL problem with a set of training tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$ that are provided for training NNs in the hope

of fast adaptation to unseen tasks $\mathcal{T}_{\text{test}}$ during the test phase, where each task is a tuple $\mathcal{T} = (g, \varphi, \mathcal{W}, X_0)$ as defined before. We consider the problem of how to leverage the learning experiences from the training tasks to accelerate the learning of the unseen test tasks. Our intuition is that when the training tasks have enough variety, the local behavior for fulfilling a test task $\mathcal{T}_{\text{test}}$ should be close to the local behavior for fulfilling some training task $\mathcal{T}_{\text{train}} \in \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$. In other words, the controller needed by a robot to fulfill the test task $\mathcal{T}_{\text{test}}$ should be close to the controller used for fulfilling some training task $\mathcal{T}_{\text{train}} \in \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$, where the training task $\mathcal{T}_{\text{train}}$ can be *different* in different subsets of the state space X. This is more general than the prevalent assumption in the meta-RL literature that the test task's controller is close to the *same* training task's controller everywhere in the state space. As a result, our approach requires less variety of the training tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$ for fast adaptation to unseen tasks.

The form of the composed NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ provides a systematic way to store learning experiences from all the training tasks and enables to select which training task should be adapted to the test task based on the current state of the robot. Given a set of training tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$, Algorithm 4 trains a subset of local networks $\mathfrak{NN}_{\text{part}} \subseteq \mathfrak{NN}$ suggested by the training tasks. For each training task $\mathcal{T}_{\text{train}} \in \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$, Algorithm 4 first calls `Runtime-Select` (i.e., Algorithm 3) to compute the corresponding activation maps $\hat{\Gamma}_k$ (line 3 in Algorithm 4). The activation maps $\hat{\Gamma}_k$ are then used to determine which local networks $\mathcal{NN}_{(q,\mathcal{P})}$ need to be trained at each state $(q, s) \in \mathbb{X}^{\otimes}$ of the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$ (line 5 in Algorithm 4). The local NNs are trained using the method `Formal-Train` given by Algorithm 1 (line 7 in Algorithm 4). Compared with Algorithm 2 that trains all the NNs to obtain the library $\mathfrak{NN}$, Algorithm 4 reduces the number of NNs need to be trained by leveraging the training tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$.

During the test phase, we adapt previous learning experiences stored in the subset of networks $\mathfrak{NN}_{\text{part}}$ to test tasks $\mathcal{T}_{\text{test}}$ by employing transfer learning. In particular, if a local NN needed by the test task $\mathcal{T}_{\text{test}}$ has not been trained, we fast learn it by fine-tuning the "closest" NN to it in the subset $\mathfrak{NN}_{\text{part}}$. Thanks to the fact that each local network $\mathcal{NN}_{(q,\mathcal{P})}$ is associated with an abstract state $q \in \mathbb{X}$ and a controller partition $\mathcal{P} \in \mathbb{P}$, we can define the distance between two local networks $\mathcal{NN}_{(q_1,\mathcal{P}_1)}$ and $\mathcal{NN}_{(q_2,\mathcal{P}_2)}$ as follows:

$$\begin{aligned}\text{Dist}\left(\mathcal{NN}_{(q_1,\mathcal{P}_1)}, \mathcal{NN}_{(q_2,\mathcal{P}_2)}\right) &= \alpha_1 \|\text{ct}_{\mathbb{X}}(q_1) - \text{ct}_{\mathbb{X}}(q_2)\| \\ &+ \alpha_2 \|\text{ct}_{\mathbb{P}}(\mathcal{P}_1) - \text{ct}_{\mathbb{P}}(\mathcal{P}_2)\|_{\max} \end{aligned} \tag{23}$$

with prespecified weights $\alpha_1, \alpha_2 \in \mathbb{R}^+$. Given a test task $\mathcal{T}_{\text{test}}$, Algorithm 5 first computes the corresponding activation maps $\hat{\Gamma}_k$ (line 1 in Algorithm 5), and then selects local networks $\mathcal{NN}_{(q,\mathcal{P})}$ to be applied at each time step until reaching the product MDP's accepting set $\mathbb{X}_G^{\otimes}$ (lines 3 and 4 in Algorithm 5). If the needed network $\mathcal{NN}_{(q,\mathcal{P})}$ has not been trained, Algorithm 5 initializes the missing network $\mathcal{NN}_{(q,\mathcal{P})}$ using the weights of the closest network $\mathcal{NN}_{(q^*,\mathcal{P}^*)}$ to it in the subset $\mathfrak{NN}_{\text{part}}$, where the distance metric between NNs is given by (23) (lines 5–7 in Algorithm 5). After that, the algorithm trains the missing network $\mathcal{NN}_{(q,\mathcal{P})}$ using PPO with only a few episodes for fine-tuning (line 8 in Algorithm 5). Thanks to the NN weight projection operator $\Pi_{\mathcal{P}}$, the resulting NN-based planner enjoys the same theoretical guarantees presented in Section V (lines 9 and 10 in Algorithm 5).

Algorithm 4: Train-Transfer $(\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}, J)$.

1: $\mathfrak{NN}_{\text{part}} = \{\}$
2: **for** $\mathcal{T}_{\text{train}} \in \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$ **do**
3: $\quad\hat{\Gamma}_k, \hat{V}_0^*, \hat{\Sigma} \otimes \mathcal{A}_\varphi = \texttt{Runtime} - \texttt{Select}(\mathcal{T}_{\text{train}})$
4: $\quad$**for** $(q, s) \in \mathbb{X}^{\otimes}, k \in \{0, \ldots, H-1\}$ **do**
5: $\quad\quad(q, \mathcal{P}) = \hat{\Gamma}_k(q, s)$
6: $\quad\quad$**if** $\mathcal{NN}_{(q,\mathcal{P})} \notin \mathfrak{NN}_{\text{part}}$ **then**
7: $\quad\quad\quad\mathcal{NN}_{(q,\mathcal{P})} = \texttt{Formal} - \texttt{Train}(q, \mathcal{P}, J)$
8: $\quad\quad\quad\mathfrak{NN}_{\text{part}} = \mathfrak{NN}_{\text{part}} \cup \{\mathcal{NN}_{(q,\mathcal{P})}\}$
9: $\quad\quad$**end if**
10: $\quad$**end for**
11: **end for**
12: **Return** $\mathfrak{NN}_{\text{part}}$

Algorithm 5: Runtime-Transfer $(\mathcal{T}_{\text{test}}, \mathfrak{NN}_{\text{part}}, J, x, s)$.

1: $\hat{\Gamma}_k, \hat{V}_0^*, \hat{\Sigma} \otimes \mathcal{A}_\varphi = \texttt{Runtime} - \texttt{Select}(\mathcal{T}_{\text{test}})$
2: $k = 0, q = \text{abs}(x)$
3: **while** $(q, s) \notin \mathbb{X}_G^{\otimes}$ **do**
4: $\quad(q, \mathcal{P}) = \hat{\Gamma}_k(q, s)$
5: $\quad$**if** $\mathcal{NN}_{(q,\mathcal{P})} \notin \mathfrak{NN}_{\text{part}}$ **then**
6: $\quad\quad\mathcal{NN}_{(q^*,\mathcal{P}^*)} = \underset{\mathcal{NN}_{(q_1,\mathcal{P}_1)} \in \mathfrak{NN}_{\text{part}}}{\text{argmin}} \text{Dist}(\mathcal{NN}_{(q_1,\mathcal{P}_1)}, \mathcal{NN}_{(q,\mathcal{P})})$
7: $\quad\quad\mathcal{NN}_{(q,\mathcal{P})} = \texttt{initialize}(\mathcal{NN}_{(q^*,\mathcal{P}^*)})$
8: $\quad\quad\mathcal{NN}_{(q,\mathcal{P})} = \texttt{PPO} - \texttt{update}(\mathcal{NN}_{(q,\mathcal{P})}, J)$
9: $\quad\quad\widehat{W}^{(F)}, \widehat{b}^{(F)} = \Pi_{\mathcal{P}}(\mathcal{NN}_{(q,\mathcal{P})})$
10: $\quad\quad$Set $\mathcal{NN}_{(q,\mathcal{P})}$ output layer weights be $\widehat{W}^{(F)}, \widehat{b}^{(F)}$
11: $\quad\quad\mathfrak{NN}_{\text{part}} = \mathfrak{NN}_{\text{part}} \cup \{\mathcal{NN}_{(q,\mathcal{P})}\}$
12: $\quad$**end if**
13: $\quad u = \mathcal{NN}_{(q,\mathcal{P})}(x)$
14: $\quad$Apply action u, observe the new state x
15: $\quad q = \text{abs}(x), s = \delta(s, L(x))$
16: $\quad k = k + 1$
17: **end while**

B. Data-Driven Symbolic Model

Recall that in Algorithm 3, after knowing the robot dynamics (i.e., the stochastic kernel t), the first step is to construct the symbolic model $\hat{\Sigma} = (\mathbb{X}, \mathbb{X}_0, \mathbb{P}, \hat{t})$ (line 1 in Algorithm 3). The construction of $\hat{\Sigma}$ requires to compute the transition probabilities $\hat{t}(q'|q, \mathcal{P}) = \int_{q'} t(dx'|z, \kappa(z))$ with all controller partitions $\mathcal{P} \in \mathbb{P}$ at each abstract state $q \in \mathbb{X}$, where $z = \text{ct}_{\mathbb{X}}(q)$, $\kappa = \text{ct}_{\mathbb{P}}(\mathcal{P})$. Reducing the computation of transition probabilities is tempting when the number of controller partitions is large, especially if the stochastic kernel $t(\cdot|x, u)$ is not a normal distribution and needs numerical integration. In this section, we accelerate the construction of $\hat{\Sigma}$ in a data-driven manner.

Algorithm 6: Construct-Symbol-Model $(\mathcal{T}, \mathcal{D}, \mathbb{X}, \mathbb{P}, I)$.

1: $\mathcal{NN} = \texttt{imitation} - \texttt{learning}(\mathcal{D})$
2: **for** $q \in \mathbb{X}$ **do**
3: $u^* = \mathcal{NN}(z)$, where $z = \text{ct}_{\mathbb{X}}(q)$
4: $P_q = \{\}$
5: **for** $i = 1, \ldots, I$ **do**
6: $\mathcal{P}^* = \underset{\mathcal{P} \in \mathbb{P} \setminus P_q}{\text{argmin}} \|\kappa(z) - u^*\|$, s.t. $\kappa = \text{ct}_{\mathbb{P}}(\mathcal{P}), z = \text{ct}_{\mathbb{X}}(q)$
7: $P_q = P_q \cup \{\mathcal{P}^*\}$
8: **end for**
9: Compute $\hat{t}(q'|q, \mathcal{P})$ with $\mathcal{P} \in P_q$
10: **end for**
11: **Return** $\hat{\Sigma}$

For a given task $\mathcal{T}$, we consider our algorithm has access to a set of expert-provided trajectories $\mathcal{D} = \{\xi_1, \xi_2, \ldots, \xi_c\}$, such as human demonstrations, that fulfill the task $\mathcal{T}$. Instead of computing all the transition probabilities $\hat{t}(q'|q, \mathcal{P})$, we use the set of expert trajectories $\mathcal{D}$ to guide the computation of transitions. The resulting symbolic model can be viewed as a symbolic representation of the expert trajectories in $\mathcal{D}$.

In Algorithm 6, we first use imitation learning to train a NN $\mathcal{NN}$ by imitating the expert trajectories in $\mathcal{D}$ (line 1 in Algorithm 6). Although the NN $\mathcal{NN}$ trained using a limited dataset $\mathcal{D}$ may not always fulfill the task $\mathcal{T}$, the network $\mathcal{NN}$ contains relevant control actions that can be used to obtain the final controller. In particular, at each abstract state $q \in \mathbb{X}$, we only compute transition probabilities $\hat{t}(q'|q, \mathcal{P})$ with controller partitions $\mathcal{P}$ suggested by the network $\mathcal{NN}$. To be specific, let u^* be the control actions given by the network $\mathcal{NN}$ at the centers of abstract states $q \in \mathbb{X}$ (line 3 in Algorithm 6). Then, Algorithm 6 selects a subset $P_q \subseteq \mathbb{P}$ consists of I controller partitions that yield control actions close to the NN's output u^*, where $I \in \mathbb{N}$ is a user-defined parameter (lines 4–8 in Algorithm 6). Finally, Algorithm 6 computes a symbolic model $\hat{\Sigma}$ with only transitions under the controller partitions in the subset P_q (line 9 in Algorithm 6). The symbolic model $\hat{\Sigma}$ contains more transitions by increasing the parameter I at the cost of computational efficiency. The choice of I can be adaptively determined as discussed in the next section.

C. Adaptive Partitioning

Recall that during the offline training, we partition the state space $X \subset \mathbb{R}^n$ and the controller space $\mathcal{P}^{K \times b} \subset \mathbb{R}^{m \times (n+1)}$ using the prespecified parameters η_q and $\eta_{\mathcal{P}}$, respectively (see Section IV-A). In this section, we comment on the choice of the grid sizes η_q and $\eta_{\mathcal{P}}$. In particular, our framework can directly incorporate the discretization techniques from the literature of abstraction-based controller synthesis (e.g., [74] and [75]). To that end, we provide a simple yet efficient example of adaptive partitioning in Algorithm 7, which enables the update of grid sizes η_q and $\eta_{\mathcal{P}}$ at runtime using transfer learning.

The first part of Algorithm 7 aims to partition the state and controller spaces such that the resulting probabilities $\hat{V}_0^*(q, s)$ of satisfying the specification φ are greater than the prespecified threshold p at all initial states $(q, s) \in \mathbb{X}_0 \times S_0$ (lines 1–7 in Algorithm 7). In particular, if the probability $\hat{V}_0^*(q, s)$ is less than p at some state $(q, s) \in \mathbb{X}_0 \times S_0$, Algorithm 7 decreases the current grid sizes η_q and $\eta_{\mathcal{P}}$ by half and increases the parameter I (line 6 in Algorithm 7). After having such a partitioning of the state and controller spaces, Algorithm 7 trains the corresponding locals networks by fine-tuning the NNs in the provided library of networks $\mathfrak{NN}_{\text{part}}$ (lines 8–18 in Algorithm 7). The following theoretical guarantee for the resulting NN-based planner to satisfy the given specification φ directly follows Theorem V.1.

Algorithm 7: Adapt-Partition $(\mathcal{T}, \mathcal{D}, \mathfrak{NN}_{\text{part}}, J, \eta_q, \eta_{\mathcal{P}}, I)$.

1: **while** $\hat{V}^*_{\min} < p$ **do**
2: $\mathbb{X} = \texttt{partition}(X, \eta_q), \mathbb{P} = \texttt{partition}(\mathcal{P}^{K \times b}, \eta_{\mathcal{P}})$
3: $\hat{\Sigma} = \texttt{Construct} - \texttt{Symbol} - \texttt{Model}(\mathcal{T}, \mathcal{D}, \mathbb{X}, \mathbb{P}, I)$
4: $\hat{\Gamma}_k, \hat{V}_0^*, \hat{\Sigma} \otimes \mathcal{A}_\varphi = \texttt{Runtime} - \texttt{Select}(\mathcal{T})$
5: $\hat{V}^*_{\min} = \min_{(q,s) \in \mathbb{X}_0 \times S_0} \hat{V}_0^*(q, s)$
6: $\eta_q = \eta_q/2,\ \eta_{\mathcal{P}} = \eta_{\mathcal{P}}/2,\ I = 2I$
7: **end while**
8: **for** $(q, s) \in \mathbb{X}^{\otimes}, k \in \{0, \ldots, H-1\}$ **do**
9: $(q, \mathcal{P}) = \hat{\Gamma}_k(q, s)$
10: **if** $\mathcal{NN}_{(q,\mathcal{P})} \notin \mathfrak{NN}_{\text{part}}$ **then**
11: $\mathcal{NN}_{(q^*,\mathcal{P}^*)} = \underset{\mathcal{NN}_{(q_1,\mathcal{P}_1)} \in \mathfrak{NN}_{\text{part}}}{\text{argmin}} \text{Dist}(\mathcal{NN}_{(q_1,\mathcal{P}_1)}, \mathcal{NN}_{(q,\mathcal{P})})$
12: $\mathcal{NN}_{(q,\mathcal{P})} = \texttt{initialize}(\mathcal{NN}_{(q^*,\mathcal{P}^*)})$
13: $\mathcal{NN}_{(q,\mathcal{P})} = \texttt{PPO} - \texttt{update}(\mathcal{NN}_{(q,\mathcal{P})}, J)$
14: $\widehat{W}^{(F)}, \widehat{b}^{(F)} = \Pi_{\mathcal{P}}(\mathcal{NN}_{(q,\mathcal{P})})$
15: Set $\mathcal{NN}_{(q,\mathcal{P})}$ output layer weights be $\widehat{W}^{(F)}, \widehat{b}^{(F)}$
16: $\mathfrak{NN}_{\text{part}} = \mathfrak{NN}_{\text{part}} \cup \{\mathcal{NN}_{(q,\mathcal{P})}\}$
17: **end if**
18: **end for**
19: **Return** $\mathfrak{NN}_{\text{part}}, \{\hat{\Gamma}_k\}_{k \in \{0,\ldots,H-1\}}$

Corollary VI.1: Consider Algorithm 7 returns a library of local networks $\mathfrak{NN}_{\text{part}}$ and an activation map Γ (denoting the functions $\hat{\Gamma}_k$). Then, the NN-based planner $\mathcal{NN}_{[\mathfrak{NN}_{\text{part}}, \Gamma]}$ satisfying $\Pr(\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{NN}_{\text{part}}, \Gamma]}} \models \varphi) \geq p - \varepsilon$ for any $x \in X_0$ and $s \in S_0$, where $\varepsilon = HZ\Delta^{\mathcal{NN}}$ and $\Delta^{\mathcal{NN}}$ is given by (20).

VII. Results

We evaluated the proposed framework both in simulation and on a robotic vehicle. All experiments were executed on a single Intel Core i9 2.4-GHz processor with 32 GB of memory. For all the reported experiments, the NNs were trained using an Adam optimizer with an adaptive learning rate, PPO clipping of 0.2, discount factor of 0.9, and batch size of 256. Our open-source implementation of the proposed neurosymbolic framework can be found online.[1]

[1][Online]. Available:https://github.com/rcpsl/Neurosymbolic_planning

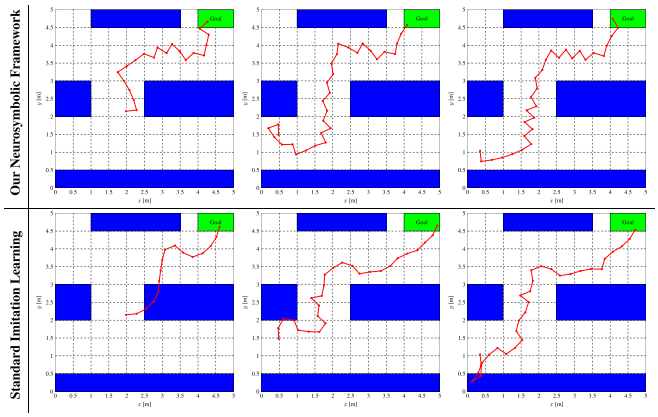


Fig. 2. Upper row shows trajectories resulting from NN-based planners trained using our framework. The lower row shows trajectories under the control of NNs trained by standard imitation learning, where the NN architectures are (left) 2 hidden layers with 10 neurons per layer, (middle) 2 hidden layers with 64 neurons per layer, and (right) 3 hidden layers with 128 neurons per layer. With the same initial states (two subfigures in the same column), only NN-based planners trained by our framework lead to collision-free trajectories.

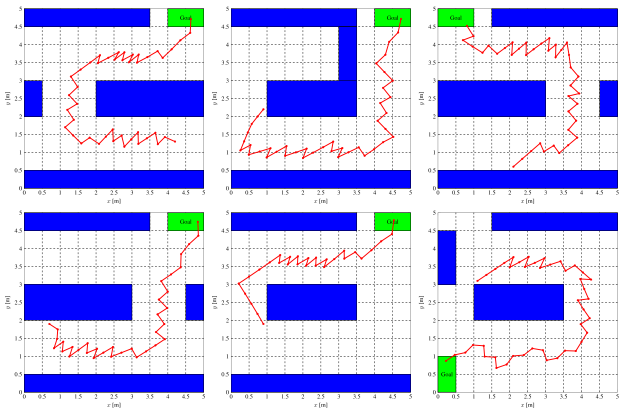


Fig. 3. Upper row shows trajectories in workspaces $\mathcal{W}_1$, $\mathcal{W}_3$, and $\mathcal{W}_5$, and the lower row corresponds to workspaces $\mathcal{W}_2$, $\mathcal{W}_4$, and $\mathcal{W}_6$. The subset of local networks $\mathfrak{NN}_{\text{part}}$ is trained in workspace $\mathcal{W}_1$ and the rest five workspaces are given at runtime. Trajectories in all the workspaces satisfy both the safety specification φ_{safety} (blue areas are obstacles) and the liveness specification $\varphi_{\text{liveness}}$ for reaching the goal (green area).

A. Controller Performance in Simulation

Consider a wheeled robot with the state vector $x = [\zeta_x, \zeta_y, \theta]^\top \in X \subset \mathbb{R}^3$, where ζ_x and ζ_y denote the coordinates of the robot and θ is the heading direction. The priori known nominal model f in the form of (2) is given by

$$\begin{aligned}\zeta_x^{(t+\Delta t)} &= \zeta_x^{(t)} + \Delta t\ v\ \cos(\theta^{(t)})\\ \zeta_y^{(t+\Delta t)} &= \zeta_y^{(t)} + \Delta t\ v\ \sin(\theta^{(t)})\\ \theta^{(t+\Delta t)} &= \theta^{(t)} + \Delta t\ u^{(t)}\end{aligned} \tag{24}$$

where the speed $v = 0.3$ m/s and the time step $\Delta t = 1$s. We train NNs to control the robot, i.e., $u^{(t)} = \text{NN}(x^{(t)})$, $\text{NN} \in \mathcal{P}^{K\times b} \subset \mathbb{R}^{1\times 4}$ with the controller space $\mathcal{P}^{K\times b}$ being a hyperrectangle.

As the first step of our framework, we discretized the state space $X \subset \mathbb{R}^3$ and the controller space $\mathcal{P}^{K\times b} \subset \mathbb{R}^{1\times 4}$ as described in Section IV-A. Specifically, we partitioned the range of heading direction $\theta \in [0, 2\pi)$ uniformly into 8 intervals, and the partitions in the x, y dimensions are shown as the dashed lines in Fig. 2. We uniformly partitioned the controller space $\mathcal{P}^{K\times b}$ into 240 hyperrectangles.

Study#1—Comparison against standard NN training for a fixed task: This study aims to compare the proposed framework against standard NN training when the task is known during training time. We aim to show the ability of our framework to guarantee the safety and correctness of achieving the task compared with standard NN training. To that end, we considered the workspace shown in Fig. 2 and a simple reach-avoid specification, i.e., reach the goal area (green) while avoiding the obstacles (blue).

We collected data by observing the control actions of an expert controller (model predictive controller) operating in this workspace while varying the initial position of the robot. We trained several NNs using imitation learning. In particular, we trained the NNs to minimize the regression loss between the control actions taken by the expert controller and those of the NN in a supervised learning manner. We trained a wide range of NN architectures using different numbers of episodes to achieve the best performance.

We then trained a library of NNs $\mathfrak{NN}$ using Algorithm 2, and we used the dataset—used to train NNs with imitation learning—to accelerate the runtime selection as detailed in Algorithm 6 (recall that line 1 in Algorithm 6 uses imitation-learning).

We report the trajectories of the proposed neurosymbolic framework in the first row of Fig. 2 and the results of the top performing NNs obtained from imitation learning in the second row of Fig. 2. As shown in the figure, we were able to find initial states from which the imitation-learning-based NNs failed to guarantee the safety of the robot (and hence failed to satisfy the mission goals). However, as shown in the figure (and supported by our theoretical analysis in Theorem V.2), our framework was capable of always achieving the mission goals and steering the robot safely to the goal.

Study#2—Generalization to unseen workspace/tasks using transfer learning: This experiment aims to study our framework's ability to generalize to unseen tasks even when the library of NNs is not complete. In other words, the trained local networks in $\mathcal{NN}$ cannot cover all possible transitions in the symbolic model, and hence a transfer learning needs to be performed during the runtime selection phase.

During the offline training, we trained a subset of local networks $\mathfrak{NN}_{\text{part}}$ by following Algorithm 4 in Section VI-A. Specifically, the local NNs are trained in the workspace $\mathcal{W}_1$ (the first subfigure in the upper row of Fig. 3). The set $\mathfrak{NN}_{\text{part}}$ consists of 658 local NNs, where each local NN has only one hidden layer with six neurons. We used PPO implemented in Keras [76] to train each local NN for 800 episodes, and projected the NN weights at the end of training. The total time for training and projecting weights of the 658 local networks in $\mathfrak{NN}_{\text{part}}$ is 2368 s.

At runtime, we tested the trained NN-based planner in five unseen workspaces $\mathcal{W}_i$, $i = 2, \ldots, 7$, and the corresponding

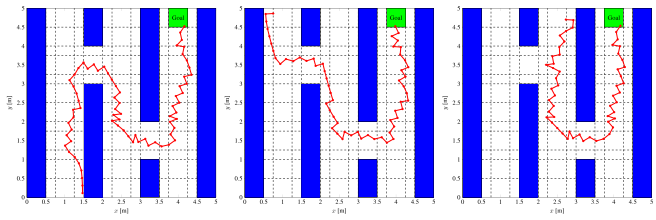


Fig. 4. Three different trajectories from workspace $\mathcal{W}_7$ corresponding to three different initial conditions. The subset of local networks $\mathfrak{NN}_{\text{part}}$ is trained in workspace $\mathcal{W}_1$ while workspace $\mathcal{W}_7$ is given at runtime. Trajectories in all the workspaces satisfy both the safety specification φ_{safety} (blue areas are obstacles) and the liveness specification $\varphi_{\text{liveness}}$ for reaching the goal (green area).

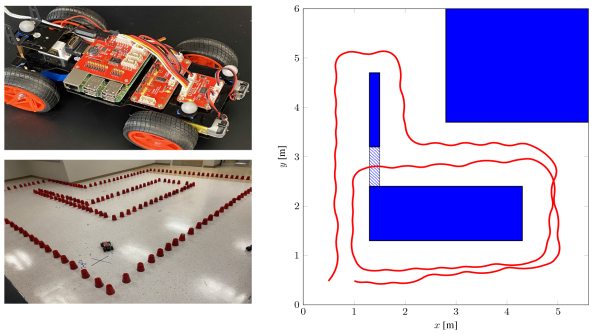


Fig. 5. (Left) PiCar and workspace. (Right) The PiCar's trajectory (red) for two loops, where the striped blue obstacle is removed after the first loop.

trajectories are shown in Figs. 3 and 4. While the workspaces $\mathcal{W}_2 \ldots \mathcal{W}_6$ contain several obstacles that are similar to the one used during training, workspace $\mathcal{W}_7$ is significantly different in terms of the number of obstacles and their alignment. For each of the workspaces, our framework computes an activation map Γ that assigns a controller partition $\mathcal{P} \in \mathbb{P}$ to each abstract state $q \in \mathbb{X}$ through dynamical programming (Algorithm 3 in Section IV-B). The local NNs corresponding to the assigned controller partitions may not have been trained offline. If this was the case, we follow Algorithm 5 that employs transfer learning to learn the missing NNs at runtime efficiently. Specifically, after initializing a missing NN using its closest NN in the set $\mathfrak{NN}_{\text{part}}$, we trained it for 80 episodes, much less than the number of episodes used in the offline training. For example, for the workspace $\mathcal{W}_2$ (the first subfigure in the lower row of Fig. 3), the length of the corresponding trajectory is 35 steps and 28 local NNs used along the trajectory are not in the set $\mathfrak{NN}_{\text{part}}$. Our algorithm efficiently trains these 28 local NNs in 10.5 s showing our framework's capability in real-time applications.

B. Actual Robotic Vehicle

We tested the proposed framework on a small robotic vehicle called PiCar, which carries a Raspberry Pi that runs the NNs trained by our framework. We used a Vicon motion capture system to measure the states of the PiCar in real-time. Fig. 5 (left) shows the PiCar and our experimental setup. We modeled the PiCar's dynamics using the rear-wheel bicycle drive [77] and used GP regression to learn the model error. We emphasize that all local NNs are trained using data collected from the simple bicycle drive model without having any data collected from the actual robotic vehicle. Data collected from the robotic vehicle are only used to obtain the GP regression model g. The results in the next two studies show the capability of our framework to generalize to model errors that were unseen during the training time.

Study#3—Dynamic changes in the workspace: We study the ability of our framework to adapt, at runtime, to changes in the workspace. This is critical in cases when the workspace is dynamic and changes over time. To that end, we trained NNs in the workspace shown in Fig. 5 (right). The part of the obstacle colored in striped blue was considered an obstacle during the training but was removed at runtime after the PiCar finished running the first loop. Thanks to the DP recursion that selects the optimal NNs at runtime (Algorithm 3 in Section IV-B), the PiCar was capable of updating its optimal selection of NNs and found a better trajectory to achieve the mission.

We reemphasize that partitioning the state space, controller space, training the library of local NNs, and building the MDP are all computed offline. At runtime, when environmental changes take place, we update the transitions in the MDP to reflect those changes. Such MDP update is computationally straightforward and results in negligible computation overhead. All the computation time is used in rerunning Algorithm 3 to reassign the local NNs to each abstract state after the environmental changes take place. For this study, Algorithm 3 consumed 0.47 s, which can handle environmental changes that occur with 2 Hz frequency.

Study#4—Comparison against meta-RL in terms of generalization to unseen tasks: This study aims to show our framework's ability to generalize to unseen tasks, even in scenarios that are known to be hard for state-of-the-art meta-RL algorithms. We conducted our second experiment with the workspaces in Fig. 6. In particular, the four subfigures in the first row of Fig. 6 are the workspaces considered for training. These four training workspaces differ in the y-coordinate of the two obstacles (blue areas). During runtime, we use the workspaces shown in the second/third row of Fig. 6. Specifically, the first subfigure in the second/third rows of Fig. 6 corresponds to a workspace that has appeared in training. The rest three subfigures in the second/third row of Fig. 6 are unseen workspaces, i.e., they are not present in training and only become known at runtime. Indeed, as demonstrated in [16], existing meta-RL algorithms are limited by the ability to adapt across homotopy classes (in Fig. 6, the training tasks and the unseen tasks are in different homotopy classes since trajectories satisfying a training task cannot be continuously deformed to trajectories satisfying an unseen task without intersecting the obstacles).

We show the PiCar's trajectories under the NN-based planner trained by our neurosymbolic framework in the second row of Fig. 6. By following Algorithm 5 with transfer learning, the PiCar's trajectories satisfy the reach-avoid specifications in all four workspaces, including the three unseen ones. Thanks to the fact that our NN-based planner is composed of local networks, our framework enables easy adaptation across homotopy classes

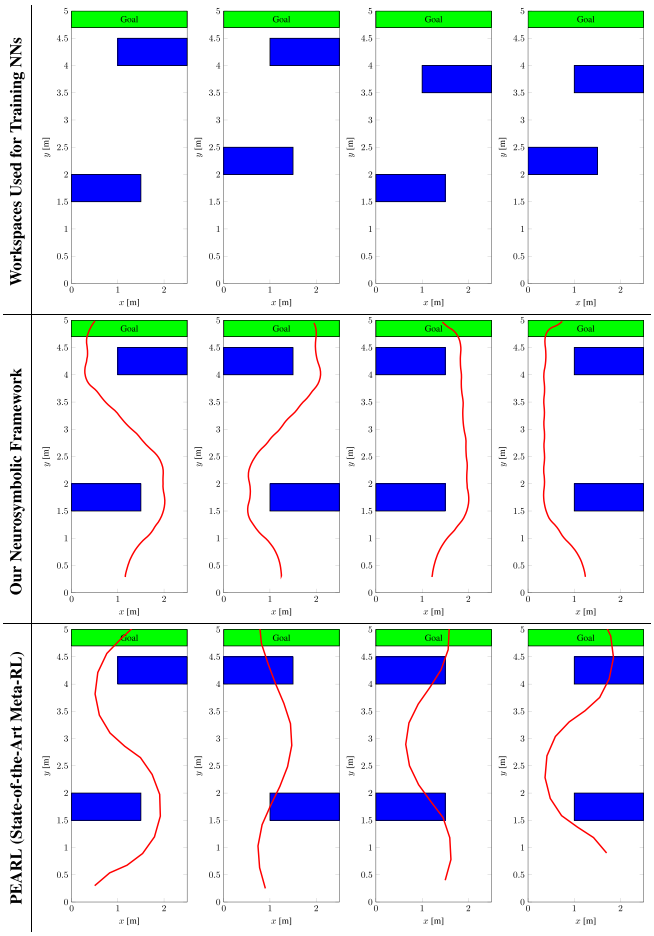


Fig. 6. Performance comparison between our neurosymbolic framework and a state-of-the-art meta-RL algorithm PEARL. The first row shows the four workspaces used for training NNs. The second row shows the PiCar's trajectories under the NN-based planner trained by our neurosymbolic framework. All the trajectories satisfy reach-avoid specifications even in unseen workspaces. The third row shows trajectories resulting from NN controllers trained by PEARL, where the trajectory is only safe in the training workspace (the first subfigure in the third row) but unsafe in the three unseen workspaces (the rest three subfigures in the third row).

by updating the activation map Γ based on the revealed task (Algorithm 3).

As a comparison, we assessed NN controllers trained by a state-of-the-art meta-RL algorithm PEARL [78] in the above workspaces. Given the four training workspaces (the first row of Fig. 6), we use PEARL to jointly learn a probabilistic encoder [79] (three hidden layers with 20 neurons per layer) and a NN controller (three hidden layers with 30 neurons per layer). The probabilistic encoder accumulates information about tasks into a vector of probabilistic context variables $z \in \mathbb{R}^5$, and the NN controller $\mathcal{NN}$ takes both the robot states x and the context variables z as input and outputs control actions $\mathcal{NN}(x, z)$.

When applying the trained NN controller to a task (either a training task or an unseen task) at runtime, PEARL needs to first update the posterior distribution of the context variables $z \in \mathbb{R}^5$ by collecting trajectories from the corresponding task. The third row of Fig. 6 shows trajectories under the control of NNs trained by PEARL. Specifically, the first subfigure in the third row of Fig. 6 corresponds to a workspace that has appeared in training, and the presented trajectory is obtained after updating the posterior distribution of z with 2 trajectories collected from this workspace. The other three subfigures in the third row of Fig. 6 show trajectories in unseen workspaces, where the trajectories cannot be safe even after updating the posterior distribution of z with 100 trajectories collected from the corresponding unseen workspace. By comparing trajectories resulting from our neurosymbolic framework and PEARL (the second and third rows in Fig. 6), NN-based planners trained by our algorithm show the capability of adapting to unseen tasks that can be very different from training tasks.

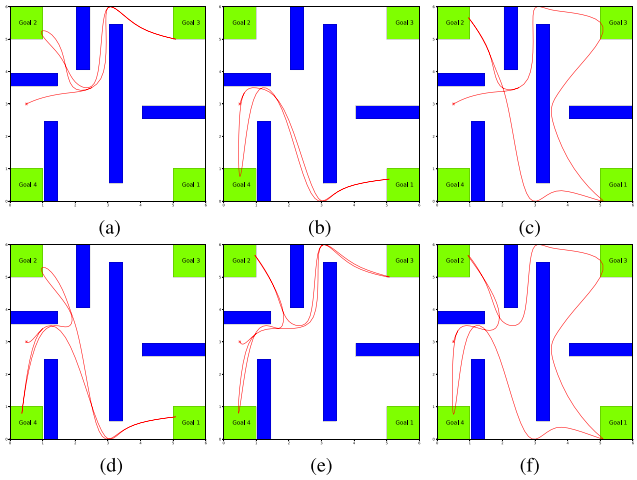


Fig. 7. Six different LTL patrolling missions where the robot needs to visit some goals (or rooms) infinitely often. Blue areas represent obstacles, and green areas represent goals. NN training took place in an empty workspace with no obstacles or goals. patrolling goals (room numbers) are reported in Table III.

C. Scalability Study

We study the scalability of our framework with respect to partition granularity, system dimension, and complexity of the LTL mission. In this experiment, we used the workspace shown in Fig. 7, constructed the symbolic models $\hat{\Sigma}$ and assigned controller partitions by following Algorithm 3. To conveniently increase the system dimension, we consider a chain of integrators represented as the linear system $x^{(t+1)} = Ax^{(t)} + Bu^{(t)}$, where $A \in \mathbb{R}^{n \times n}$ is the identity matrix and $u^{(t)} \in \mathbb{R}^2$. Note that our algorithm is not aware of the linearity of the dynamics constraints nor is exploiting this fact. The algorithm has access to a simulator [the function f in (2)] that it can use to construct the symbolic model $\hat{\Sigma}$. To construct the symbolic models $\hat{\Sigma}$ efficiently, we adopt Algorithm 6 and only consider local controller partitions by setting the range parameter I be 25.

Table I reports the execution time that grows with the increasing number of abstract states and controller partitions. We observed that the average training time for one local NN gets smaller as we increase the number of abstract states. In particular, the average time to train one local NN is 2.575, 1.983, and 0.905 s for the case of 1000, 2197, and 4096 abstract states, respectively. We hypothesize that the smaller the grid size—and

TABLE I
SCALABILITY OF ALGORITHM 3 WITH RESPECT TO PARTITION GRANULARITY

Number of Abstract States	Number of Controller Partitions	Build Symbolic Model $\hat{\Sigma}$ [s]	Assign Controller Partitions [s]	Success Rate
1000	100	10.1	21.8	96.4 %
1000	324	11.3	69.8	96.98 %
1000	900	13.3	193.2	97.64 %
2197	100	41.6	74.2	97.01 %
2197	324	44.7	227.5	97.12 %
2197	900	51.3	673.45	97.58 %
4096	100	145.6	383.8	99.92 %
4096	324	151.2	1210.64	99.93 %
4096	900	164.6	3444.43	99.98 %

TABLE II
SCALABILITY OF ALGORITHM 3 WITH RESPECT TO SYSTEM DIMENSION

System Dimension n	Number of Abstract States	Build Symbolic Model $\hat{\Sigma}$ [s]	Assign Controller Partitions [s]
2	324	2.1	1.8
4	1296	9.4	10.4
6	4096	70.3	62.9
8	16384	311.2.	158.4
10	59049	1581.9	441.7

TABLE III
SCALABILITY OF ALGORITHM 3 WITH RESPECT TO MISSION COMPLEXITY

	Patrolling Goals	Build Symbolic Model $\hat{\Sigma}$ [s]	Assign Controller Partitions [s]
(a)	Goal #2 & Goal #3	9.4	2.03
(b)	Goal #1 & Goal #4	9.4	2.59
(c)	Goal #1 & Goal #2	9.4	4.72
(d)	Goal #1, Goal #2, & Goal #4	9.4	11.38
(e)	Goal #2, Goal #3, & Goal #4	9.4	10.74
(f)	Goal #1, Goal #2, Goal #3, & Goal #4	9.4	24.66

hence the higher the number of abstract states and the smaller the volume of the abstract state—the lower the complexity of the local NN is needed to control the system in this abstract state. Moreover, Theorems V.1 and V.2 points to a tradeoff between the performance of the proposed NN controller—in terms of satisfying mission specifications—and the granularity of the state space and controller space discretization parameters. To that end, we trained the local NNs using the linear dynamics above and then added Gaussian noise with zero mean and a variance of 0.75 to each state dynamics while building the symbolic model. We randomly sampled 50 initial states and observed the success in avoiding obstacles and reaching Goal 1 (as shown in Fig. 7). We reported the success rate in the last column of Table I. As supported by the theoretical analysis in Theorems V.1 and V.2, increasing the granularity of the state space and controller space partitioning leads to an increase in the success rate of the proposed NN controller at the expense of higher execution times.

In Table II, we show the scalability by increasing the system dimension n. As expected, the complexity of training the local NNs, building the symbolic model, and assigning the controller partitions increases with the system dimension.

Finally, Table III tabulates the execution time as we increase the complexity of the LTL mission. As shown in Fig. 7, the workspace has four goals and five obstacles. We consider patrolling tasks in which the robot needs to visit some goals "infinitely often" while avoiding the obstacles. The location of the obstacles and goals were not known during the training of the local NNs. To increase the mission's complexity, we increased the number of goals patrolled from 2 to 4 and selected the patrolling goals randomly. We increased the horizon of the BLTL formula until a solution was found. As given in Table III, the execution time needed to build the symbolic model is unaffected by the LTL mission. Nevertheless, more complex missions require longer horizons to find a solution, resulting in longer execution times for assigning controller partitions.

The execution times reported in Tables I–III show that our algorithm can handle high-dimensional systems and complex LTL missions in a reasonable amount of time. Although we conducted all the experiments on a single CPU core, we note that our framework is highly parallelizable. For example, both computing transition probabilities in the symbolic model $\hat{\Sigma}$ and training local networks $\mathcal{NN}_{(q,\mathcal{P})}$ can be parallelized.

VIII. CONCLUSION, LIMITATIONS, AND TRADEOFFS

This article proposed a neurosymbolic framework of motion and task planning for mobile robots with respect to temporal logic formulas. By incorporating a symbolic model into the training of NNs and restricting the behavior of NNs, the resulting NN-based planner can be generalized to unseen tasks with correctness guarantees. Compared with existing techniques, our framework results in provably correct NN-based planners, removing the need for online monitoring, predictive filters, barrier functions, or post-training formal verification. An interesting topic for future research is extending the framework to multiple agents with high-bandwidth sensor perception of the environment.

The proposed framework comes with some limitations. First, the projection operator assumes the NN represents a CPWA function. While this assumption covers any NN with a ReLU activation unit, it excludes NNs with traditional nonlinearities (e.g., tanh and sigmoid). Second, the framework depends on extending existing abstraction-based algorithms to the case when the controller is a NN. Hence, our framework inherits some properties of these algorithms. In particular, our framework can handle only fragments of LTL formulas since the optimality analysis (generalized in Theorem V.2) assumes finite-horizon tasks. Our framework can potentially benefit from the new advances in solving infinite-horizon optimal control problems with LTL constraints [80]. Similarly, our framework inherits the deficiencies of abstraction-based control regarding its scalability to high-dimensional state and control spaces due to the need to partition (or discretize) the state and controller spaces. Our framework can potentially benefit from sampling-based/abstraction-free techniques [8], [81], [82] through integrating the projection operator $\Pi_{\mathcal{P}}$ in the sampling process. These three limitations point toward future research opportunities for integrating ideas from LTL-based motion planning with ML to achieve the flexibility of ML with the rigor of LTL-based planners.

The discretization granularity—captured by the parameters η_q and $\eta_{\mathcal{P}}$ needs to be carefully tuned. In particular, these two parameters offer a tradeoff between the discretization granularity and the performance of our framework, both in terms of the probability of satisfying the mission specifications and execution time. This tradeoff is captured in Theorems V.1 and V.2 and reflected in the scalability results in Table I. Moreover, the

correctness of this tradeoff relies on the implicit assumption that the error in the dynamical model g (the GP regression kernel) is in itself correct. Finally, while the local NNs do not need to be retrained (or fine-tuned) to generalize to unseen tasks, our neurosymbolic framework requires additional algorithmic computations to readjust the activation map Γ for each task. This computational overhead entails solving an optimal control problem over the discretized MDP. These additional computations weaken the notion of "generalization to unseen tasks."

APPENDIX A
SECTION III PROOFS

In this appendix, we provide proofs of Section III.

A. Proof of Proposition III.2

Proof: Let $h:\mathbb{R}^n \to \mathbb{R}^{\mathfrak{o}_{F-1}}$ represent all the hidden layers, then the NNs before and after the change of the output layer weights are given by $\mathcal{NN}^{\theta}: x \mapsto W^{(F)}h(x)+b^{(F)}$ and $\mathcal{NN}^{\widehat{\theta}}: x \mapsto \widehat{W}^{(F)}h(x)+\widehat{b}^{(F)}$, respectively. The change in the NN's outputs is bounded as follows:

$$\max_{x\in q} \|\mathcal{NN}^{\widehat{\theta}}(x)-\mathcal{NN}^{\theta}(x)\|_1 \tag{25}$$

$$= \max_{x\in q} \sum_{i=1}^{m}\left|\sum_{j=1}^{\mathfrak{o}_{F-1}} \Delta W_{ij}^{(F)} h_j(x)+\Delta b_i^{(F)}\right| \tag{26}$$

$$\le \max_{x\in q} \sum_{i=1}^{m}\sum_{j=1}^{\mathfrak{o}_{F-1}} |\Delta W_{ij}^{(F)}| h_j(x)+\sum_{i=1}^{m}|\Delta b_i^{(F)}| \tag{27}$$

$$= \max_{x\in \text{Vert}(\mathbb{L}_{\mathcal{NN}^{\theta}\cap q})} \sum_{i=1}^{m}\sum_{j=1}^{\mathfrak{o}_{F-1}} |\Delta W_{ij}^{(F)}| h_j(x)+\sum_{i=1}^{m}|\Delta b_i^{(F)}| \tag{28}$$

where (26) directly follows the form of $\mathcal{NN}^{\theta}$ and $\mathcal{NN}^{\widehat{\theta}}$, (27) swaps the order of taking the absolute value and the summation, and uses the fact that the hidden layers satisfy $h(x)\ge 0$ due to the ReLU activation function. When x is restricted to each linear region of $\mathcal{NN}^{\theta}$, the hidden layer function h is affine, and hence (27) is a linear program whose optimal solution is attained at extreme points. Therefore, in (28), the maximum can be taken over a *finite* set of states that are vertices of the linear regions in $\mathbb{L}_{\mathcal{NN}^{\theta}\cap q}$. ■

B. Proof of Proposition III.3

Proof: We write the optimization problem (13) and (14) in its equivalent epigraph form

$$\min_{\widehat{W}^{(F)},\widehat{b}^{(F)},t,s_{ij},v_i} \quad t \quad \text{such that}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{\mathfrak{o}_{F-1}} s_{ij}h_j(x)+\sum_{i=1}^{m} v_i \le t,\ \forall x\in \text{Vert}(\mathbb{L}_{\mathcal{NN}^{\theta}\cap q}) \tag{29}$$

$$|\widehat{W}_{ij}^{(F)}-W_{ij}^{(F)}|\le s_{ij},\ i=1,\dots,m,\ j=1,\dots,\mathfrak{o}_{F-1} \tag{30}$$

$$|\widehat{b}_i^{(F)}-b_i^{(F)}|\le v_i,\ i=1,\dots,m \tag{31}$$

$$\widehat{K}_i\in\mathcal{P},\ \forall \mathcal{R}_i\in\{\mathcal{R}\in\mathbb{L}_{\mathcal{NN}^{\theta}}\mid \mathcal{R}\cap q\neq\emptyset\}. \tag{32}$$

The inequalities in (29) are affine since the hidden layer function h is known and does not depend on the optimization variables. The number of inequalities in (29) is finite since the set of vertices $\text{Vert}(\mathbb{L}_{\mathcal{NN}^{\theta}\cap q})$ is finite. To see the constraints (32) are affine, consider the NN $\mathcal{NN}^{\widehat{\theta}}: x\mapsto \widehat{W}^{(F)}h(x)+\widehat{b}^{(F)}$ with the output layer weights $\widehat{W}^{(F)}$, $\widehat{b}^{(F)}$ and the hidden layer function h. The CPWA function $\mathcal{NN}^{\widehat{\theta}}$ can also be written in the form of (5), i.e., $\mathcal{NN}^{\widehat{\theta}}: x\mapsto \widehat{K}_i(x)$ at each linear region $\mathcal{R}_i\in\mathbb{L}_{\mathcal{NN}^{\theta}}$, where we use the notation $\widehat{K}_i(x)$ to denote $\widehat{K}_i'x+\widehat{b}_i'$. Since the hidden-layer function h restricted to each linear region $\mathcal{R}_i\in\mathbb{L}_{\mathcal{NN}^{\theta}}$ is a known affine function of x, the parameters $\widehat{K}_i$ affinely depend on $\widehat{W}^{(F)}$ and $\widehat{b}^{(F)}$. Therefore, the constraints $\widehat{K}_i\in\mathcal{P}$ are affine constraints of $\widehat{W}^{(F)}$ and $\widehat{b}^{(F)}$. ■

APPENDIX B
SECTION V PROOFS

In this appendix, we provide proofs of Theorems V.1 and V.2 in Section V. Let $\Sigma=(X,X_0,U,t)$ be a robotic system with continuous state and action spaces and $\mathcal{A}_{\varphi}=(S,S_0,\mathbb{A},G,\delta)$ be the DFA of a mission specification φ. Similar to the product MDP $\hat{\Sigma}\otimes\mathcal{A}_{\varphi}$, the product between Σ and $\mathcal{A}_{\varphi}$ is given by $\Sigma\otimes\mathcal{A}_{\varphi}=(X^{\otimes},X_0^{\otimes},U,X_G^{\otimes},t^{\otimes})$, where

1) $X^{\otimes}=X\times S$ is the state space,
2) $X_0^{\otimes}=\{(x_0,\delta(s_0,L(x_0)))|x_0\in X_0,s_0\in S_0\}$ is the set of initial states, where $L:X\to\mathbb{A}$ is the labeling function that assigns to each state $x\in X$ the subset of atomic propositions $L(x)\in\mathbb{A}$ that evaluate $true$ at x,
3) $U\subset\mathbb{R}^m$ is the control action space,
4) $X_G^{\otimes}=X\times G$ is the accepting set, and
5) The stochastic kernel $t^{\otimes}$ is given by

$$t^{\otimes}(dx',s'|x,s,u)=\begin{cases} t(dx'|x,u) & \text{if } s'=\delta(s,L(x')) \\ 0 & \text{else.}\end{cases}$$

A. Proof of Theorem V.1

Proof: Given the NN-based planner $\mathcal{NN}_{[\mathfrak{M},\Gamma]}$ obtained using our framework, we define functions $V_k^{\mathcal{NN}}:X^{\otimes}\to[0,1]$ that map a state $(x,s)\in X^{\otimes}$ to the probability of reaching the accepting set $X_G^{\otimes}$ in $H-k$ steps from the state (x,s) and under the control of $\mathcal{NN}_{[\mathfrak{M},\Gamma]}$. With this notation, we have $V_0^{\mathcal{NN}}(x,s)=\Pr(\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{M},\Gamma]}}\models\varphi)$ since reaching the accepting set $X_G^{\otimes}$ in H steps in the product MDP $\Sigma\otimes\mathcal{A}_{\varphi}$ is equivalent to Σ satisfying φ. In the following, we show that for any $x\in q$ and $k=0,\dots,H$:

$$|V_k^{\mathcal{NN}}(x,s)-\hat{V}_k^{*}(q,s)|\le (H-k)Z\Delta^{\mathcal{NN}} \tag{33}$$

which yields (19) by letting $k=0$. By the definition of $V_k^{\mathcal{NN}}$, the probabilities of reaching the accepting set $X_G^{\otimes}$ under the NN-based planner $\mathcal{NN}_{[\mathfrak{M},\Gamma]}$ can be expressed as

$$V_k^{\mathcal{NN}}(x,s)=\mathbf{1}_G(s)$$
$$+\mathbf{1}_{S\setminus G}(s)\sum_{s'\in S}\int_X V_{k+1}^{\mathcal{NN}}(x',s')t^{\otimes}(dx',s'|x,s,\mathcal{NN}(x)) \tag{34}$$

with the initial condition $V_H^{\mathcal{NN}}(x,s)=\mathbf{1}_G(s)$. In the stochastic kernel $t^{\otimes}$ in (34), we use $\mathcal{NN}$ to denote the local network selected by the activation map Γ_{k+1} at the state (x,s) for simplicity. Although solving (34) is intractable due to the continuous state space, we can bound the difference between $V_k^{\mathcal{NN}}$ and $\hat{V}_k^*$ as (33) by induction.

For the base case $k=H$, (33) trivially holds since $V_H^{\mathcal{NN}}(x,s)=\mathbf{1}_G(s)$ and $\hat{V}_H^*(q,s)=\mathbf{1}_G(s)$. For the induction hypothesis, suppose for $k+1$ it holds that

$$|V_{k+1}^{\mathcal{NN}}(x,s)-\hat{V}_{k+1}^*(q,s)|\le (H-k-1)Z\Delta^{\mathcal{NN}}. \tag{35}$$

Let $\bar{V}_k^*$ be a piecewise constant interpolation of $\hat{V}_k^*$ defined by $\bar{V}_k^*(x,s)=\hat{V}_k^*(q,s)$ for any $x\in q$ and any $s\in S$. Then

$$\begin{aligned}&|V_k^{\mathcal{NN}}(x,s)-\hat{V}_k^*(q,s)|\\&\le |V_k^{\mathcal{NN}}(x,s)-V_k^{\mathcal{NN}}(z,s)|+|V_k^{\mathcal{NN}}(z,s)-\bar{V}_k^*(z,s)|\end{aligned} \tag{36}$$

where $z=\mathrm{ct}_{\mathbb{X}}(q)$ and $x\in q$. For the first term on the RHS

$$\begin{aligned}&|V_k^{\mathcal{NN}}(x,s)-V_k^{\mathcal{NN}}(z,s)|\\&=|\mathbf{1}_G(s)+\mathbf{1}_{S\setminus G}(s)\sum_{s'\in S}\int_X V_{k+1}^{\mathcal{NN}}(x',s')t^{\otimes}(dx',s'|x,s,\mathcal{NN}(x))\\&\quad-\mathbf{1}_G(s)+\mathbf{1}_{S\setminus G}(s)\sum_{s'\in S}\int_X V_{k+1}^{\mathcal{NN}}(x',s')t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z))|\\&\le\sum_{s'\in S}\int_X V_{k+1}^{\mathcal{NN}}(x',s')|t^{\otimes}(dx',s'|x,s,\mathcal{NN}(x))\\&\quad-t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z)|\\&\le Z\int_X|t(dx'|x,\mathcal{NN}(x))-t(dx'|z,\mathcal{NN}(z)|\\&\le Z\int_X|t(dx'|x,\mathcal{NN}(x))-t(dx'|z,\mathcal{NN}(x))|\\&\quad+|t(dx'|z,\mathcal{NN}(x))-t(dx'|z,\mathcal{NN}(z))|\\&\le Z\Lambda_i\|x-z\|+ZB_i\|\mathcal{NN}(x)-\mathcal{NN}(z)\|\\&\le Z\Lambda_i\eta_q+ZB_iL_i\eta_q.\end{aligned} \tag{37}$$

For the second term on the RHS of (36)

$$\begin{aligned}&|V_k^{\mathcal{NN}}(z,s)-\bar{V}_k^*(z,s)|\\&=|\mathbf{1}_G(s)+\mathbf{1}_{S\setminus G}(s)\sum_{s'\in S}\int_X V_{k+1}^{\mathcal{NN}}(x',s')t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z))\\&\quad-\mathbf{1}_G(s)+\mathbf{1}_{S\setminus G}(s)\max_{\mathcal{P}\in\mathbb{P}}\sum_{(q',s')\in\mathbb{X}^{\otimes}}\hat{V}_{k+1}^*(q',s')\hat{t}^{\otimes}(q',s'|q,s,\mathcal{P})|\end{aligned} \tag{38}$$

$$\begin{aligned}&\le|\sum_{s'\in S}\int_X V_{k+1}^{\mathcal{NN}}(x',s')t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z))\\&\quad-\sum_{s'\in S}\sum_{q'\in\mathbb{X}}\hat{V}_{k+1}^*(q',s')\hat{t}^{\otimes}(q',s'|q,s,\mathcal{P}^*)|\end{aligned} \tag{39}$$

$$\begin{aligned}&\le|\sum_{s'\in S}\int_X V_{k+1}^{\mathcal{NN}}(x',s')t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z))\\&\quad-\sum_{s'\in S}\int_X\bar{V}_{k+1}^*(x',s')t^{\otimes}(dx',s'|z,s,\mathrm{ct}_{\mathbb{P}}(\mathcal{P}^*)(z))|\end{aligned} \tag{40}$$

$$\begin{aligned}&\le\sum_{s'\in S}\int_X|V_{k+1}^{\mathcal{NN}}(x',s')-\bar{V}_{k+1}^*(x',s')|t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z))\\&\quad+\sum_{s'\in S}\int_X\bar{V}_{k+1}^*(x',s')|t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z))\\&\quad-t^{\otimes}(dx',s'|z,s,\mathrm{ct}_{\mathbb{P}}(\mathcal{P}^*)(z))|\end{aligned} \tag{41}$$

$$\le(H-k-1)Z\Delta^{\mathcal{NN}}+Z\sqrt{m(n+1)}\mathcal{L}_XB_i\eta_{\mathcal{P}} \tag{42}$$

where (38) uses the DP recursion (17) and (18), in (39) $\mathcal{P}^*$ denotes the maximizer, and (40) uses the definition of $\hat{t}$ in (16) with $z=\mathrm{ct}_{\mathbb{X}}(q)$. In (42), we use the induction hypothesis (35), and the inequality $\|K(x)-K'(x)\|\le\|K-K'\|\|x\|\le\sqrt{m(n+1)}\|K-K'\|_{\max}\mathcal{L}_X\le\sqrt{m(n+1)}\eta_{\mathcal{P}}\mathcal{L}_X$, where $\|K-K'\|_{\max}\le\eta_{\mathcal{P}}$ since the local network $\mathcal{NN}$ selected by the activation map Γ represents a CPWA function from the maximizer $\mathcal{P}^\star$, i.e., $K,K'\in\mathcal{P}^\star\subset\mathbb{R}^{m\times(n+1)}$. Substitute (37) and (42) into (36) yields (33).

B. Proof of Theorem V.2

Proof: Let functions $V_k^*:X^{\otimes}\to[0,1]$ map a state $(x,s)\in X^{\otimes}$ to the probability of reaching the accepting set $X_G^{\otimes}$ in $H-k$ steps from the state (x,s) and under the optimal controller $\mathcal{C}_\varphi^*:X\times S\to U$. Then, $V_0^*(x,s)=\Pr(\xi_{\mathcal{C}_\varphi^*}^{(x,s)}\models\varphi)$ since reaching the accepting set $X_G^{\otimes}$ in H steps in the product MDP $\Sigma\otimes\mathcal{A}_\varphi$ is equivalent to Σ satisfying φ. The optimal probabilities of reaching the accepting set $X_G^{\otimes}$ can be expressed using DP recursion

$$\begin{aligned}Q_k(x,s,u)=&\mathbf{1}_G(s)\\&+\mathbf{1}_{S\setminus G}(s)\sum_{s'\in S}\int_X V_{k+1}^*(x',s')t^{\otimes}(dx',s'|x,s,u)\end{aligned} \tag{43}$$

$$V_k^*(x,s)=\max_{u\in U}Q_k(x,s,u). \tag{44}$$

Although solving V_k^* and the corresponding optimal controller $\mathcal{C}_\varphi^*$ is intractable due to the continuous state and action spaces, we can bound the difference between V_k^* and $\hat{V}_k^*$ by induction similar to the proof of (33) in Theorem V.1. We skip the details and directly give the following bound:

$$|V_k^*(x,s)-\hat{V}_k^*(q,s)|\le(H-k)Z\Delta^* \tag{45}$$

where $x\in q$ and Δ^* is given by (22). With (33) and (45), we have

$$\begin{aligned}&|V_k^{\mathcal{NN}}(x,s)-V_k^*(x,s)|\\&\le|V_k^{\mathcal{NN}}(x,s)-\hat{V}_k^*(q,s)|+|V_k^*(x,s)-\hat{V}_k^*(q,s)|\\&\le(H-k)Z(\Delta^{\mathcal{NN}}+\Delta^*)\end{aligned} \tag{46}$$

which yields (21) by letting $k=0$. ■

REFERENCES

[1] A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan, "Linear encodings of bounded LTL model checking," *Log. Methods Comput. Sci.*, vol. 2, no. 5:5, pp. 1–64, 2006.

[2] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods System Des.*, vol. 19, pp. 291–314, 2001.

[3] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive, high-level robot control," *IEEE Robot. Autom. Mag.*, vol. 18, no. 3, pp. 65–74, Sep. 2011.

[4] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for robots: Guarantees and feedback for robot behavior," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 1, pp. 211–236, 2018.

[5] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *IEEE Robot. Autom. Mag.*, vol. 18, no. 3, pp. 55–64, Sep. 2011.

[6] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *Int. J. Robot. Res.*, vol. 37, no. 1, pp. 104–136, 2018.

[7] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Motion and action planning under LTL specifications using navigation functions and action description language," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 240–245.

[8] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 2689–2696.

[9] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Proc. IEEE 44th Conf. Decis. Control*, 2005, pp. 4885–4890.

[10] G. E. Fainekos, S. G. Loizou, and G. J. Pappas, "Translating temporal logic to controller specifications," in *Proc. 45th IEEE Conf. Decis. Control*, 2006, pp. 899–904.

[11] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2007, pp. 3116–3121.

[12] Y. Shoukry et al., "Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming," in *Proc. IEEE 56th Annu. Conf. Decis. Control*, 2017, pp. 1132–1137.

[13] P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach*. Berlin, Germany: Springer, 2009.

[14] C. Belta, B. Yordanov, and E. A. Gol, *Formal Methods for Discrete-Time Dynamical Systems*, vol. 15. Berlin, Germany: Springer, 2017.

[15] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2924–2932.

[16] Z. Cao, M. Kwon, and D. Sadigh, "Transfer reinforcement learning across homotopy classes," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 2706–2713, Apr. 2021.

[17] X. Sun, W. Fatnassi, U. S. Cruz, and Y. Shoukry, "Provably safe model-based meta reinforcement learning: An abstraction-based approach," in *Proc. 60th IEEE Conf. Decis. Control*, 2021, pp. 2963–2968.

[18] Y. Jiang, S. Bharadwaj, B. Wu, R. Shah, U. Topcu, and P. Stone, "Temporal-logic-based reward shaping for continuing learning tasks," in *Proc. 33rd AAAI Conf. Artif. Intell.*, 2020, pp. 7995–8003.

[19] W. Saunders, G. Sastry, A. Stuhlmueller, and O. Evans, "Trial without error: Towards safe reinforcement learning via human intervention," in *Proc. 17th Int. Conf. Auton. Agents MultiAgent Syst.*, 2018, pp. 2067–2069.

[20] F. Berkenkamp, A. Krause, and A. P. Schoellig, "Bayesian optimization with safety constraints: Safe and automatic parameter tuning in robotics," *Mach. Learn.*, vol. 112, pp. 3713–3747, 2023.

[21] A. Liu, G. Shi, S.-J. Chung, A. Anandkumar, and Y. Yue, "Robust regression for safe exploration in control," in *Proc. Mach. Learn. Res.*, 2020, pp. 1–6.

[22] P. Pauli, A. Koch, J. Berberich, and F. Allgöwer, "Training robust neural networks using Lipschitz bounds," *IEEE Control Syst. Lett.*, vol. 6, pp. 121–126, 2022.

[23] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 22–31.

[24] M. Turchetta, F. Berkenkamp, and A. Krause, "Safe exploration in finite Markov decision processes with Gaussian processes," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4312–4320.

[25] L. Wen, J. Duan, S. E. Li, S. Xu, and H. Peng, "Safe reinforcement learning for autonomous vehicles through parallel constrained policy optimization," in *Proc. IEEE 23rd Int. Conf. Intell. Transp. Syst.*, 2020, pp. 1–7.

[26] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *Proc. NASA Formal Methods Symp.*, 2018, pp. 121–138.

[27] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *Found. Trends Optim.*, Hanover, MA, USA: Now Publishers Inc., vol. 4, no. 34, pp. 244–404, Feb. 2021, doi: 10.1561/2400000035.

[28] X. Sun, H. Khedr, and Y. Shoukry, "Formal verification of neural network controlled autonomous systems," in *Proc. 22nd ACM Int. Conf. Hybrid Syst.: Computation Control*, 2019, pp. 147–156.

[29] H. Khedr, J. Ferlez, and Y. Shoukry, "PEREGRiNN: Penalized-relaxation greedy neural network verifier," in *Proc. Int. Conf. Comput. Aided Verification*, 2021, pp. 287–300.

[30] J. Ferlez, H. Khedr, and Y. Shoukry, "Fast BATLLNN: Fast box analysis of two-level lattice neural networks," in *Proc. 25th ACM Int. Conf. Hybrid Syst.: Computation Control*, 2022, pp. 1–11.

[31] U. S. Cruz and Y. Shoukry, "NNLander-VeriF: A neural network formal verification framework for vision-based autonomous aircraft landing," in *Proc. NASA Formal Methods Symp.*, 2022, pp. 213–230.

[32] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas, "Efficient and accurate estimation of Lipschitz constants for deep neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2019, pp. 11423–11434.

[33] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: Verifying safety properties of hybrid systems with neural network controllers," in *Proc. 22nd ACM Int. Conf. Hybrid Syst.: Computation Control*, 2019, pp. 169–178.

[34] W. Xiang, D. M. Lopez, P. Musau, and T. T. Johnson, "Reachable set estimation and verification for neural network models of nonlinear dynamic systems," in *Safe, Autonomous and Intelligent Vehicles*. Berlin, Germany: Springer, 2019, pp. 123–144.

[35] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *IEEE Trans. Autom. Control*, vol. 64, no. 7, pp. 2737–2752, Jul. 2019.

[36] K.-C. Hsu, V. Rubies-Royo, C. J. Tomlin, and J. F. Fisac, "Safety and liveness guarantees through reach-avoid reinforcement learning," in *Proc. Robot.: Sci. Syst.*, 2021, Paper 077.

[37] A. Abate, D. Ahmed, A. Edwards, M. Giacobbe, and A. Peruffo, "FOSSIL: A software tool for the formal synthesis of Lyapunov functions and barrier certificates using neural networks," in *Proc. 24th ACM Int. Conf. Hybrid Syst.: Computation Control*, 2021, pp. 1–11.

[38] S. Chen, M. Fazlyab, M. Morari, G. J. Pappas, and V. M. Preciado, "Learning Lyapunov functions for hybrid systems," in *Proc. 24th ACM Int. Conf. Hybrid Syst.: Computation Control*, 2021, pp. 1–11.

[39] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 3387–3395.

[40] A. Robey et al., "Learning control barrier functions from expert demonstrations," in *Proc. IEEE 59th Conf. Decis. Control*, 2020, pp. 3717–3724.

[41] A. J. Taylor, A. Singletary, Y. Yue, and A. D. Ames, "A control barrier perspective on episodic learning via projection-to-state safety," *IEEE Control Syst. Lett.*, vol. 5, no. 3, pp. 1019–1024, Jul. 2021.

[42] L. Wang, E. A. Theodorou, and M. Egerstedt, "Safe learning of quadrotor dynamics using barrier certificates," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 2460–2465.

[43] W. Xiao, C. Belta, and C. G. Cassandras, "Adaptive control barrier functions," *IEEE Trans. Autom. Control*, vol. 67, no. 5, pp. 2267–2281, May 2022.

[44] O. Bastani, S. Li, and A. Xu, "Safe reinforcement learning via statistical model predictive shielding," in *Proc. Robot.: Sci. Syst.*, 2021, pp. 1–13.

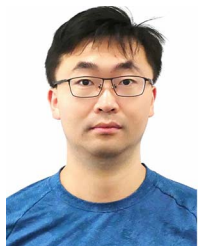
[45] K. P. Wabersich and M. N. Zeilinger, "Linear model predictive safety certification for learning-based control," in *Proc. IEEE 57th Conf. Decis. Control*, 2018, pp. 7130–7135.

[46] K. P. Wabersich and M. N. Zeilinger, "A predictive safety filter for learning-based control of constrained nonlinear dynamical systems," *Automatica*, vol. 129, 2021, Art. no. 109597.

[47] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee, "Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees," in *Proc. IEEE 58th Conf. Decis. Control*, 2019, pp. 5338–5343.

[48] A. Balakrishnan and J. V. Deshmukh, "Structured reward shaping using signal temporal logic specifications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 3481–3486.

[49] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U Topcu, "Safe reinforcement learning via shielding," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 2669–2678.
[50] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2017.
[51] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A Lyapunov-based approach to safe reinforcement learning," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2018, pp. 8092–8101.
[52] Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh, "Lyapunov-based safe policy optimization for continuous control," in *Proc. RL4RealLife Workshop 36th Int. Conf. Mach. Learn.*, 2019.
[53] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 2107–2116.
[54] R. Alur, S. Bansal, O. Bastani, and K. Jothimurugan, "A framework for transforming specifications in reinforcement learning," in *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*. Berlin, Germany: Springer, 2022, pp. 604–624.
[55] C. Neary, C. Verginis, M. Cubuktepe, and U. Topcu, "Verifiable and compositional reinforcement learning systems," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2022, pp. 615–623.
[56] J. Wang, S. Kalluraya, and Y. Kantaros, "Verified compositions of neural network controllers for temporal logic control objectives," in *Proc. IEEE 61st Conf. Decis. Control*, 2022, pp. 4004–4009.
[57] K. Jothimurugan, S. Bansal, O. Bastani, and R. Alur, "Compositional reinforcement learning from logical specifications," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2021, pp. 10026–10039.
[58] L. Illanes, X. Yan, R. T. Icarte, and S. A. McIlraith, "Symbolic plans as high-level instructions for reinforcement learning," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2020, pp. 540–550.
[59] Z. Xu and U. Topcu, "Transfer of temporal logic formulas in reinforcement learning," in *Proc. Int. Joint Conf. Artif. Intell.: Proc. Conf.*, 2019, Art. no. 4010.
[60] F. Zhuang et al., "A comprehensive survey on transfer learning," *Proc. IEEE*, vol. 109, no. 1, pp. 43–76, Jan. 2021.
[61] G. Anderson, A. Verma, I. Dillig, and S. Chaudhuri, "Neurosymbolic reinforcement learning with formally verified exploration," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2020, pp. 6172–6183.
[62] A. Verma, H. Le, Y. Yue, and S. Chaudhuri, "Imitation-projected programmatic reinforcement learning," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2019, pp. 15752–15763.
[63] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2018, pp. 2499–2509.
[64] G. Weiss, Y. Goldberg, and E. Yahav, "Extracting automata from recurrent neural networks using queries and counterexamples," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 5247–5256.
[65] S. Carr, N. Jansen, and U. Topcu, "Verifiable RNN-based policies for POMDPS under temporal logic constraints," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, 2020, pp. 4121–4127.
[66] C. E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.
[67] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 49–58.
[68] F. Rossi and N. Mattei, "Building ethically bounded AI," in *Proc. 33rd AAAI Conf. Artif. Intell.*, 2019, pp. 9785–9789.
[69] J. Ferlez and Y. Shoukry, "Bounding the complexity of formally verifying neural networks: A geometric approach," in *Proc. IEEE 60th Conf. Decis. Control*, 2021, pp. 5104–5109.
[70] S. Bak, H.-D. Tran, K. Hobbs, and T. T. Johnson, "Improved geometric path enumeration for verifying ReLU neural networks," in *Proc. 32nd Int. Conf. Comput. Aided Verification*, 2020, pp. 66–96.
[71] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[72] T. Latvala, "Efficient model checking of safety properties," in *Proc. Model Checking Softw. 10th Int. SPIN Workshop*, 2003, pp. 74–88.
[73] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *Protocol Specification, Testing and Verification XV*. 1996, pp. 3–18.
[74] S. E. Z. Soudjani and A. Abate, "Adaptive and sequential gridding procedures for the abstraction and verification of stochastic processes," *SIAM J. Appl. Dynamical Syst.*, vol. 12, no. 2, pp. 921–956, 2013.
[75] K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck, "Multi-layered abstraction-based controller synthesis for continuous-time systems," in *Proc. 21st Int. Conf. Hybrid Syst.: Computation Control*, 2018, pp. 120–129.
[76] F. Chollet et al., "Keras," 2015. Accessed: Jun. 2022. [Online]. Available: https://github.com/fchollet/keras
[77] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, *Wheeled Mobile Robotics*. Amsterdam, The Netherlands: Elsevier, 2017.
[78] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 5331–5340.
[79] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. 31st Int. Conf. Mach. Learn.*, 2014. [Online]. Available: https://dblp.uni-trier.de/rec/journals/corr/KingmaW13.html?view=bibtex
[80] X. Ding, S. L. Smith, C. Belta, and D. Rus, "Optimal control of Markov decision processes with linear temporal logic constraints," *IEEE Trans. Autom. Control*, vol. 59, no. 5, pp. 1244–1257, May 2014.
[81] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1487–1507, Oct. 2021.
[82] C. I. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 4817–4822.



Xiaowu Sun (Graduate Student Member, IEEE) received the B.Sc. degree in physics from Nanjing University, Nanjing, China in 2013, and the M.Sc. degree in electrical engineering from the University of Maryland, College Park, MD, USA, in 2018. He is currently working toward the Ph.D. degree with the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA, USA .

His research interests include formal methods for control, neural networks, reinforcement learning and robotics.

Mr. Sun was the Finalist in the ACM SIGBED SRC Student Competition at the Cyber-Physical Systems (CPS-IoT) Week 2021. His research on using formal verification to analyze neural network controlled systems was nominated for consideration in the Communications of the ACM Research Highlights.

Yasser Shoukry (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the University of California, Los Angeles, CA, USA, in 2015.

He is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA, where he leads the Resilient Cyber-Physical Systems Lab. Before joining UCI, he spent two years as an Assistant Professor with the University of Maryland, College Park, MD, USA. Between 2015 and 2017, he was a joint Postdoctoral Researcher with UC Berkeley, UCLA, and UPenn. His current research focuses on the design and implementation of resilient, AI-enabled, cyber-physical systems, and IoT.

Dr. Shoukry was the recipient of the Best Paper Award from the International Conference on Cyber-Physical Systems (ICCPS) in 2016, and the Distinguished Dissertation Award from UCLA EE department in 2016, the Best Demo Award from the International Conference on Information Processing in Sensor Networks (IPSN) in 2017, the NSF CAREER Award in 2019, George Corcoran Memorial Award from the University of Maryland for his contributions to teaching and educational leadership in the field of CPS and IoT in 2019, and the Early Career Award from the IEEE Technical Committee on Cyber-Physical Systems (TC-CPS) in 2021. In 2015, he led the UCLA/Caltech/CMU team to win the NSF Early Career Investigators (NSF-ECI) research challenge. His team represented the NSF- ECI in the NIST Global Cities Technology Challenge, an initiative designed to advance the deployment of Internet of Things (IoT) technologies within a smart city.