2021 60th IEEE Conference on Decision and Control (CDC)
December 13-15, 2021. Austin, Texas

Provably Safe Model-Based Meta Reinforcement Learning: An Abstraction-Based Approach

Xiaowu Sun[1], Wael Fatnassi[1], Ulices Santa Cruz[1], and Yasser Shoukry[1]

2021 60th IEEE Conference on Decision and Control (CDC) | 978-1-6654-3659-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CDC45484.2021.9683009

***Abstract*— While conventional reinforcement learning focuses on designing agents that can perform one task, meta-learning aims, instead, to solve the problem of designing agents that can generalize to different tasks (e.g., environments, obstacles, and goals) that were not considered during the design or the training of these agents. In this paper, we consider the problem of training a provably safe Neural Network (NN) controller for uncertain nonlinear dynamical systems that can generalize to new tasks that were not present in the training data while preserving strong safety guarantees. Our approach is to learn a set of NNs during the training phase. When the task becomes available at runtime, our framework will carefully select a subset of NNs and compose them to form the final NN controller. Critical to our approach is the ability to compute a finite-state abstraction of the nonlinear dynamical system. This abstract model captures the behavior of the closed-loop system under all possible NN weights, and is used to train the NNs and compose them when the task becomes available. We provide theoretical guarantees on the correctness of the resulting NN controller and show the efficacy of our approach via simulations.**

I. INTRODUCTION

Meta Reinforcement Learning (meta-RL) refers to algorithms that can leverage experience from previous learning experience to learn how to adapt to new tasks quickly. In other words, while contemporary reinforcement learning focuses on designing agents that can perform one task, meta-RL aims 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. To achieve such aim, and without loss of generality, meta-training can be seen as a bi-level optimization problem where one optimization contains another optimization as a constraint [1], [2]. The inner optimization corresponds to the classical training of a policy to achieve a particular task while the outer optimization focuses instead on optimizing the meta-representation that generalizes to different tasks [3], [4], [5], [6], [7]. For a review on the current achievements in the field of Meta-RL, we refer the reader to this survey [3].

While the current successes of meta-RL are undeniable, significant drawbacks of meta-RL in its current form are (i) the *lack of formal guarantees on its ability to generalize to unforeseen tasks* and (ii) the *lack of formal guarantees with regards to its safety*.

In this paper, we confine our attention to reach-avoid tasks (i.e., a robot that needs to reach a goal without hitting obstacles) and propose a framework for meta-RL that can generalize to tasks (e.g., different environments, obstacles, and goals) that were not present in the training data. The proposed framework results into NN controllers that are *provably safe* with regards to *any* reach-avoid task, which could be unseen during the design of these neural networks.

Recently, the authors proposed a framework for *provably-correct* training of neural networks [8]. Given an error-free nonlinear dynamical system, the framework in [8] computes a finite-state abstract model that captures the closed-loop behavior under all possible NN weights. Using this finite-state abstract model, the framework identifies the subset of NN weights guaranteed to satisfy the safety requirements (i.e., avoiding obstacles). During training, the learning algorithm is augmented with a NN weight projection operator that enforces the resulting NN to be provably safe. To account for the liveness properties (i.e., reaching the goal), the framework uses the finite-state abstract model to identify candidate NN weights that may satisfy the liveness properties and biases the NN training to achieve the liveness specification.

While the previous results reported in [8] focused on the case when the task (environment, obstacles, and goal) is known during the training of the NN controller, we extend these results in this paper to account for the case when the task is *unknown* during training. In particular, instead of training one neural network, we train a set of neural networks. To fulfill a set of infinitely many tasks using a finite set of NN controllers, our approach is to restrict each neural network to some local behavior, yet the composition of these neural networks captures all possible behaviors. Moreover, and unlike the results reported in [8], we consider in this paper the case when the nonlinear dynamical system is only partially known. We evaluated our approach on the problem of steering a wheeled robot and we show that our framework is capable of generalizing to tasks that were not present in the training of the NN controller while guaranteeing the safety of the robot.

II. PROBLEM FORMULATION

A. Notation

Let $\|x\|$ be the Euclidean norm of the vector $x \in \mathbb{R}^n$, $\|A\|$ be the induced 2-norm of the matrix $A \in \mathbb{R}^{m\times n}$, and $\|A\|_{\max} = \max_{i,j}|A_{ij}|$ be the max norm of the matrix A. Given two vectors $x_1 \in \mathbb{R}^{n_1}$ and $x_2 \in \mathbb{R}^{n_2}$, we denote by $(x_1, x_2) \in \mathbb{R}^{n_1+n_2}$ the column vector $[x_1^\top, x_2^\top]^\top$. We use $\oplus$ to denote the Minkowski sum, and $\mathrm{Int}(\mathcal{S})$ to denote the interior of the set $\mathcal{S}$. Any Borel space $\mathcal{X}$ is assumed to be endowed with a Borel σ-algebra, which is denoted by $\mathcal{B}(\mathcal{X})$. We use $\mathbf{1}_{\mathcal{S}}$ to denote the indicator function of a set $\mathcal{S}$.

This work was partially sponsored by the NSF awards #CNS-2002405 and #CNS-2013824.

[1] Xiaowu Sun, Wael Fatnassi, Ulices Santa Cruz, and Yasser Shoukry are with Department of Electrical Engineering and Computer Science, University of California, Irvine {xiaowus,wfatnass,usantacr,yshoukry}@uci.edu.

B. Dynamical Model, Task, and Specification

We consider discrete-time nonlinear dynamical systems of the form:

$$x^{(k+1)} = f(x^{(k)}, u^{(k)}) + g(x^{(k)}, u^{(k)}), \tag{1}$$

where $x^{(k)} \in \mathcal{X} \subset \mathbb{R}^n$ is the state and $u^{(k)} \in \mathcal{U}$ is the control input at time step $k \in \mathbb{N}$. The dynamical model consists of two parts: the priori known nominal model f, and the unknown model-error g, which is deterministic and captures unmodeled dynamics. Though the model-error g is unknown, we assume it is bounded by a compact set $\mathcal{D} \subset \mathbb{R}^n$, i.e., $g(x, u) \in \mathcal{D}$ for all $x \in \mathcal{X}$ and $u \in \mathcal{U}$. We also assume both functions f and g are locally Lipschitz continuous. As a well-studied technique to learn unknown functions from data, we assume the model-error g can be learned using Gaussian Process (GP) regression [9]. We use $\mathcal{GP}(\mu_g, \sigma_g^2)$ to denote a GP regression model with the posterior mean and variance functions be μ_g and σ_g^2, respectively[1]. Given a feedback control law $\Psi : \mathcal{X} \to \mathcal{U}$, we use $\xi_{x_0,\Psi} : \mathbb{N} \to \mathcal{X}$ to denote the closed-loop trajectory of (1) that starts from the state $x_0 \in \mathcal{X}$ and evolves under the control law Ψ. In this paper, our primary focus is on controlling the nonlinear system (1) with a state-feedback NN controller $\mathcal{NN} : \mathcal{X} \to \mathcal{U}$.

We use $\mathcal{W} = \{\mathcal{X}_{\text{goal}}, \mathcal{O}_1, \ldots, \mathcal{O}_o\}$ to denote a task where $\mathcal{X}_{\text{goal}} \subset \mathcal{X}$ is the goal and $\{\mathcal{O}_1, \ldots, \mathcal{O}_o\}$ with $\mathcal{O}_i \subset \mathcal{X}$ is the set of obstacles. More formally, given a task $\mathcal{W}$, a safety specification ϕ_{safety} requires avoiding all the obstacles and a liveness specification ϕ_{liveness} requires reaching the goal in a bounded time horizon H. We use $\xi_{x_0,\Psi} \models \phi_{\text{safety}}$ and $\xi_{x_0,\Psi} \models \phi_{\text{liveness}}$ to denote a trajectory $\xi_{x_0,\Psi}$ satisfies the safety and liveness specifications, respectively, i.e.,

$$\xi_{x_0,\Psi} \models \phi_{\text{safety}} \iff \forall k \in \mathbb{N},\ \forall i \in \{\mathcal{O}_1, \ldots, \mathcal{O}_o\},\ \xi_{x_0,\Psi}(k) \notin \mathcal{O}_i,$$
$$\xi_{x_0,\Psi} \models \phi_{\text{liveness}} \iff \exists k \in \{1, \ldots H\},\ \xi_{x_0,\Psi}(k) \in \mathcal{X}_{\text{goal}}.$$

Given a set of initial states $\mathcal{X}_{\text{init}}$, a control law $\Psi : \mathcal{X} \to \mathcal{U}$ satisfies a specification ϕ (denoted by $\Psi, \mathcal{X}_{\text{init}} \models \phi$) if all trajectories starting from the set $\mathcal{X}_{\text{init}}$ satisfy the specification, i.e., $\xi_{x,\Psi} \models \phi$, $\forall x \in \mathcal{X}_{\text{init}}$. Since the specifications and the satisfying set of initial states depend on the task, we explicitly add $\mathcal{W}$ as a superscript whenever need emphasize the dependency, such as $\phi^{\mathcal{W}}_{\text{safety}}$, $\phi^{\mathcal{W}}_{\text{liveness}}$, and $\mathcal{X}^{\mathcal{W}}_{\text{init}}$.

While conventional reinforcement learning focuses on training a neural network that works for one specific task, meta-RL focuses, instead, on training controllers that can work for a multitude of tasks. To formally capture this, we use $\mathfrak{W}_{\mathcal{X}}$ to denote the set of all the tasks (corresponding to configurations of the goal and obstacles) with the goals and the obstacles be defined over the state space $\mathcal{X}$.

C. Main Problem

We consider the problem of designing provably correct NN controllers for unseen tasks. Specifically, the task $\mathcal{W} \in \mathfrak{W}_{\mathcal{X}}$ is unknown during the training of the NN controller, and will be known only at runtime. Therefore, our objective is to train a set (or a collection) of different ReLU NNs along with a selection algorithm that can select the correct NNs once the task $\mathcal{W}$ becomes available at runtime. Before presenting the problem under consideration, we introduce the following notion of NN composition.

Definition 2.1: Given a set of neural networks $\mathfrak{NN} = \{\mathcal{NN}_1, \mathcal{NN}_2, \ldots, \mathcal{NN}_m\}$ along with an activation map $\Gamma : \mathcal{X} \to \{1, \ldots, m\}$, the composed neural network $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ is defined as:

$$\mathcal{NN}_{[\mathfrak{NN},\Gamma]}(x) = \mathcal{NN}_{\Gamma(x)}(x)$$

In other words, the activation map Γ selects the index of the NN that need to be activated at a particular state $x \in \mathcal{X}$. Now, we can define the problem of interest as follows.

Problem 2.2: Given the nonlinear dynamical system (1). Design a NN controller $\mathcal{NN}$ consists of two parts: a set of ReLU NNs $\mathfrak{NN} = \{\mathcal{NN}_1, \mathcal{NN}_2, \ldots, \mathcal{NN}_m\}$ and a selection algorithm SEL, such that for any task $\mathcal{W} \in \mathfrak{W}_{\mathcal{X}}$, the selection algorithm $\texttt{SEL}(\mathcal{W}, \mathfrak{NN})$ returns a set of initial states $\mathcal{X}^{\mathcal{W}}_{\text{init}} \subseteq \mathcal{X}$ and an activation map $\Gamma^{\mathcal{W}}$ satisfying:

$$\mathcal{NN}_{[\mathfrak{NN},\Gamma^{\mathcal{W}}]}, \mathcal{X}^{\mathcal{W}}_{\text{init}} \models \phi^{\mathcal{W}}_{\text{safety}} \wedge \phi^{\mathcal{W}}_{\text{liveness}}.$$

III. FRAMEWORK

A. Overview

Before describing our approach to solve Problems 2.2, we start by recalling that every ReLU NN represents a Continuous Piece-Wise Affine (CPWA) function [10]. Let $\Psi_{\text{CPWA}} : \mathcal{X} \to \mathbb{R}^m$ denote a CPWA function of the form:

$$\Psi_{\text{CPWA}}(x) = K_i x + b_i \quad \text{if } x \in \mathcal{R}_i,\ i = 1, \ldots, L, \tag{2}$$

where the polytopic sets $\{\mathcal{R}_1, \ldots, \mathcal{R}_L\}$ is a partition of the set $\mathcal{X}$. In this paper, we confine our attention to CPWA controllers (and hence neural network 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 $b_i \in \mathcal{P}^b$.

To fulfill a set of *infinitely* many tasks $\mathfrak{W}_{\mathcal{X}}$ using a *finite* set of ReLU NNs $\mathfrak{NN}$, our approach is to restrict each NN in the set $\mathfrak{NN}$ to some local behavior, yet the set $\mathfrak{NN}$ captures all possible behavior of the system. We use the mathematical model of the physical system (1) to guide training of the NNs, as well as selecting NNs from the set $\mathfrak{NN}$ at runtime.

During training, without knowing the tasks, we train a set of ReLU NNs $\mathfrak{NN}$ using the following two steps:

- Capture the closed-loop behavior of the system under *all* CPWA controllers using a finite-state Markov decision process (MDP). To define the action space of this MDP, we partition the space of all CPWA controllers into a finite number of partitions. Each partition corresponds to a family of CPWA controllers. Hence, each transition in the MDP is labeled by a symbol that corresponds to a particular family of CPWA functions. The transition probabilities can then be computed using the knowledge of the model (1) and the Gaussian Process $\mathcal{GP}(\mu_g, \sigma_g^2)$.We refer to this finite-state MDP as the abstract model of the system.
- Train one NN corresponds to each transition in the MDP. We refer to each of these NNs as a local NN. Let $\mathfrak{NN}$ be the set of all such local NNs. The training enforces each local NN to represent a CPWA function

[1]In the case of a multiple output function g, i.e., $m > 1$, we model each output dimension with an independent GP. We keep the notations unchanged for simplicity.

that belongs to the family of CPWA controllers associated with this transition. This is achieved by using the NN weight projection operator introduced in [8].

At runtime, given an arbitrary task $\mathcal{W} \in \mathfrak{W}_{\mathcal{X}}$, the algorithm $\texttt{SEL}(\mathcal{W}, \mathfrak{NN})$ selects NNs from the set $\mathfrak{NN}$ to satisfy $\phi^{\mathcal{W}}_{\text{safety}} \wedge \phi^{\mathcal{W}}_{\text{liveness}}$:

- To satisfy the safety specification $\phi^{\mathcal{W}}_{\text{safety}}$, the algorithm SEL identifies the partitions of CPWA controllers that are safe at each abstract state in the MDP. The selected NNs from the set $\mathfrak{NN}$ must correspond to the safe CPWA partitions.
- For the liveness specification $\phi^{\mathcal{W}}_{\text{liveness}}$, the algorithm SEL first searches for the optimal policy of the MDP using dynamic programming (DP), where the allowed transitions in the MDP are limited to those have been identified to be safe. Based on the optimal policy of the MDP, it decides which local NN in the set $\mathfrak{NN}$ should be used at each state.

We highlight that the proposed framework above *always* guarantees that the resulting NN controller satisfies the safety specification $\phi^{\mathcal{W}}_{\text{safety}}$ for any task $\mathcal{W} \in \mathfrak{W}_{\mathcal{X}}$, regardless the accuracy of the learned model-error using GP regression. For the liveness specification $\phi^{\mathcal{W}}_{\text{liveness}}$, due to the learned model-error is probabilistic, we relax Problem 2.2 to maximize the probability of satisfying the liveness specification $\phi^{\mathcal{W}}_{\text{liveness}}$. We also provide a quantified bound on the probability for the NN controller to satisfy $\phi^{\mathcal{W}}_{\text{liveness}}$.

Figure 1 conceptualizes our framework. In Figure 1 (left), we partition the state space $\mathcal{X}$ into a set of abstract states $\mathbb{X} = \{q_1, q_2, q_3, q_4\}$ and the controller space $\mathcal{P}^K \times \mathcal{P}^b$ into a set of controller partitions $\mathbb{P} = \{\mathcal{P}_1, \mathcal{P}_2\}$. Figure 1 (right) shows the resulting MDP, with transition probabilities labeled by the side of the transitions. Then, the set $\mathfrak{NN}$ contains 9 local NNs corresponding to the 9 transitions in the MDP.

Consider two different tasks given at runtime. Task $\mathcal{W}_1$ specifies that the goal $\mathcal{X}_{\text{goal}}$ is represented by the abstract state q_4 and the only obstacle is q_2. At state q_1, our selection algorithm decides to use the local network $\text{NN}_{(q_1,\mathcal{P}_2,q_4)}$, which corresponds to the transition from state q_1 to q_4 under partition $\mathcal{P}_2$. In task $\mathcal{W}_2$, state q_4 is still the goal, but there is no obstacle. For this task, our selection algorithm decides to use $\text{NN}_{(q_1,\mathcal{P}_1,q_2)}$ at state q_1 and use $\text{NN}_{(q_2,\mathcal{P}_1,q_4)}$ at state q_2. Notice that with this choice the probability of reaching the goal is 1, which is higher than the probability 0.1 by using $\text{NN}_{(q_1,\mathcal{P}_2,q_4)}$ at state q_1.

In the above procedure, the set $\mathfrak{NN}$ may contain a large number of local NNs. To accelerate the training process, we employ ideas from transfer learning and provide the same correctness guarantees. We refer readers to Section VII in the extended version [11] for the details on transfer learning.

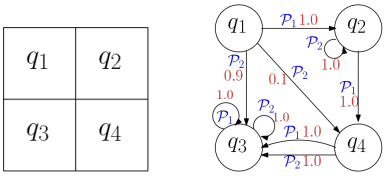

Fig. 1: (left) Partition the state space into four abstract states; (right) MDP labeled with transition probabilities (red).

IV. PROVABLY-CORRECT TRAINING OF THE SET OF NEURAL NETWORKS $\mathfrak{NN}$

A. Abstract Model

In this section, we extend the abstract model proposed in [8] by taking into account the unknown model-error g. Unlike the results reported in [8] where the system was assume to be error-free and deterministic (and hence can be abstracted by a finite-state machine), in this paper, the dynamical model (1) is stochastic due to the use of GP regression to capture the error in the model. This necessitates the use of finite-state MDP to abstract the dynamics in (1).

State and Controller Space Partitioning: We partition the state space $\mathcal{X} \subset \mathbb{R}^n$ into a set of abstract states, denoted by $\mathbb{X} = \{q_1, \ldots, q_N\}$. Each $q_i \in \mathbb{X}$ is an infinity-norm ball in $\mathbb{R}^n$ centered around some state $x_i \in \mathcal{X}$. The partitioning satisfies $\mathcal{X} = \bigcup_{q \in \mathbb{X}} q$, and $\text{Int}(q_i) \cap \text{Int}(q_j) = \emptyset$ if $i \neq j$. With an abuse of notation, q denotes both an abstract state, i.e., $q \in \mathbb{X}$, and a subset of states, i.e., $q \subset \mathcal{X}$. Since we construct the abstract model before knowing the tasks, the state space $\mathcal{X}$ does not contain any obstacle or goal.

Similarly, we partition the controller space into polytopic subsets. For simplicity of notation, we define the set of parameters $\mathcal{P}^{K \times b} \subset \mathbb{R}^{m \times (n+1)}$ be a polytope that combines $\mathcal{P}^K \subset \mathbb{R}^{m \times n}$ and $\mathcal{P}^b \subset \mathbb{R}^m$. With some abuse of notation, we 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 $(K_i', b_i') = K_i$. The controller space $\mathcal{P}^{K \times b}$ is discretized into a collection of polytopic subsets in $\mathbb{R}^{m \times (n+1)}$, denoted by $\mathbb{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$. Each $\mathcal{P}_i$ 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 $i \neq j$. We call each of the subsets $\mathcal{P}_i \in \mathbb{P}$ a controller partition. Each controller partition $\mathcal{P} \in \mathbb{P}$ represents a subset of CPWA functions, by restricting parameters K_i in a CPWA function to take values from $\mathcal{P}$.

MDP Transitions: Next, we compute the set of all allowable transitions in the MDP. To that end, we define the posterior of an abstract state q under a controller partition $\mathcal{P}$ be the set of states that can be reached in one step from states $x \in q$ by using affine state feedback controllers with parameters $K \in \mathcal{P}$ under the dynamical model (1) as follows:

$$\text{Post}(q, \mathcal{P}) \triangleq \{h(x, K(x)) \in \mathbb{R}^n \mid x \in q, K \in \mathcal{P}\} \oplus \mathcal{D}, \quad (3)$$

where $\mathcal{D} \subset \mathbb{R}^n$ is defined in Section II-B as the bound of the model-error g. Since calculating the exact posterior of a nonlinear system is computationally daunting, we rely on over-approximation $\widehat{\text{Post}}(q, \mathcal{P})$. Furthermore, we define the Next operator as follows:

$$\text{Next}(q, \mathcal{P}) \triangleq \{q' \in \mathbb{X} \mid q' \cap \widehat{\text{Post}}(q, \mathcal{P}) \neq \emptyset\}. \quad (4)$$

Then, a transition from state q to state q' with label $\mathcal{P}$ is allowed in the MDP if and only if $q' \in \text{Next}(q, \mathcal{P})$.

Transition Probability: The final step is to compute the transition probabilities associated with each of the transitions constructed in the previous step. We define transition probabilities based on representative points in abstract states and controller partitions. Specifically, we choose the representative points to be the centers (recall that both q and $\mathcal{P}$ are infinity-norm balls and hence their centers are well defined).

Let $\mathrm{ct}_{\mathbb{X}} : \mathbb{X} \to \mathcal{X}$ map an abstract state $q \in \mathbb{X}$ to its center and $\mathrm{ct}_{\mathbb{P}} : \mathbb{P} \to \mathcal{P}^{K\times b}$ map a controller partition $\mathcal{P} \in \mathbb{P}$ to the matrix $\mathrm{ct}_{\mathbb{P}}(\mathcal{P}) \in \mathcal{P}^{K\times b}$, which is the center of $\mathcal{P}$. Furthermore, we use $\mathrm{abs}_{\mathbb{X}} : \mathcal{X} \to \mathbb{X}$ to denote the map from a state $x \in \mathcal{X}$ to the abstract state that contains x, i.e., $x \in \mathrm{abs}_{\mathbb{X}}(x) \in \mathbb{X}$, and similarly, the map $\mathrm{abs}_{\mathbb{P}} : \mathcal{P}^{K\times b} \to \mathbb{P}$ satisfies $K \in \mathrm{abs}_{\mathbb{P}}(K) \in \mathbb{P}$ for any $K \in \mathcal{P}^{K\times b}$.

Given the dynamical system (1) with the model-error g learned by a GP regression model $\mathcal{GP}(\mu_g, \sigma_g^2)$, let $t : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to [0, 1]$ be the corresponding conditional stochastic kernel. Specifically, given the current state $x \in \mathcal{X}$ and input $u \in \mathcal{U}$, the distribution $t(\cdot|x, u)$ is given by the Gaussian distribution $\mathcal{N}(f(x, u)+\mu_g(x, u), \sigma_g^2(x, u))$. For any set $\mathcal{A} \in \mathcal{B}(\mathcal{X})$ and any $k \in \mathbb{N}$, the probability of reaching the set $\mathcal{A}$ in one step from state $x^{(k)}$ with input $u^{(k)}$ is given by:

$$\Pr(x^{(k+1)} \in \mathcal{A}|x^{(k)}, u^{(k)}) = \int_{\mathcal{A}} T(dx^{(k+1)}|x^{(k)}, u^{(k)}) \quad (5)$$

where we use the notation $T(dx'|x, u) \triangleq t(x'|x, u)\mu(dx')$. This integral can be easily computed since $t(\cdot|x, u)$ is a Gaussian distribution. With above notations, we define our abstract model as follows:

Definition 4.1: The abstract model of (1) is a finite MDP $\hat{\Sigma}$ defined as a tuple $\hat{\Sigma} = (\mathbb{X}, \mathbb{P}, \hat{T})$, where:

- The state space is the set of abstract states $\mathbb{X}$;
- The set of controls at each state is given by the set of controller partitions $\mathbb{P}$;
- The transition probability from state $q \in \mathbb{X}$ to $q' \in \mathbb{X}$ with label $\mathcal{P} \in \mathbb{P}$ is given by:

$$\hat{T}(q'|q, \mathcal{P}) = \begin{cases} \int_{q'} t(dx'|z, \kappa(z)) & \text{if } q' \in \mathrm{Next}(q, \mathcal{P}) \\ 0 & \text{else} \end{cases}$$

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

B. Train Local NNs with Weight Projection

Once the abstract model is computed, the next step is to train the set of local networks $\mathfrak{NN}$ without the knowledge of the tasks. In order to capture the closed-loop behavior of the system under *all* possible CPWA controllers, we train one local network corresponding to each transition (with non-zero transition probability) in the MDP $\hat{\Sigma}$. Algorithm 1 outlines training of all the local NNs. We use $\mathcal{NN}_{(q,\mathcal{P},q')}$ to denote the local NN corresponding to the transition in the MDP $\hat{\Sigma}$ from state $q \in \mathbb{X}$ to $q' \in \mathbb{X}$ under partition $\mathcal{P} \in \mathbb{P}$.

We train each local network $\mathcal{NN}_{(q,\mathcal{P},q')}$ using Proximal Policy Optimization (PPO) [12] (line 5 in Algorithm 1). While choosing the reward function in reinforcement learning is often challenging, our algorithm enjoys a straightforward yet efficient formulation of reward functions. To be specific, for a local network $\mathcal{NN}_{(q,\mathcal{P},q')}$, let $\kappa = \mathrm{ct}_{\mathbb{P}}(\mathcal{P})$ and $w_1, w_2 \in \mathbb{R}$ be pre-specified weights, our reward function encourages moving towards the state q' with controllers chosen from the partition $\mathcal{P}$:

$$r(x, u) = \begin{cases} -w_2\|u - \kappa(x)\|, & \text{if } h(x, u) + \mu_g(x, u) \in q' \\ -w_1\|f(x, u) + \mu_g(x, u) - \mathrm{ct}_{\mathbb{X}}(q')\| - w_2\|u - \kappa(x)\| & \text{otherwise} \end{cases}$$

where μ_g is the posterior mean function from the GP regression. With this dynamical model, PPO can efficiently explore the workspace without running the real agent.

The training of local networks $\mathcal{NN}_{(q,\mathcal{P},q')}$ is followed by applying a NN weight projection operator `Project` introduced in [8]. Given a neural network $\mathcal{NN}$ and a controller partition $\mathcal{P}$, this projection operator ensures that:

$$\texttt{Project}(\mathcal{NN}, \mathcal{P}) \in \mathcal{P}.$$

In other words, this projection operator forces that $\mathcal{NN}$ can only give rise to one of the CPWA functions that belong to the controller partition $\mathcal{P}$. We refer readers to [8] for more details on the NN weight projection. Algorithm 1 summarizes the discussion in this subsection.

Algorithm 1 TRAIN-LOCAL-NNS ($\hat{\Sigma}$)

1: $\mathfrak{NN} = \{\}$
2: **for** $q \in \mathbb{X}$ **do**
3: **for** $\mathcal{P} \in \mathbb{P}$ **do**
4: **for** $q' \in \mathrm{Next}(q, \mathcal{P})$ **do**
5: $\mathcal{NN}_{(q,\mathcal{P},q')} = \texttt{PPO}(q, \mathcal{P}, q', h, \mu_g)$
6: $\mathcal{NN}_{(q,\mathcal{P},q')} = \texttt{Project}(\mathcal{NN}_{(q,\mathcal{P},q')}, \mathcal{P})$
7: $\mathfrak{NN} = \mathfrak{NN} \cup \{\mathcal{NN}_{(q,\mathcal{P},q')}\}$
8: **Return** $\mathfrak{NN}$

V. THE SELECTION ALGORITHM $\texttt{SEL}(\mathcal{W}, \mathfrak{NN})$

In this section, we present our selection algorithm $\texttt{SEL}(\mathcal{W}, \mathfrak{NN})$ which is used at runtime when an arbitrary task $\mathcal{W} \in \mathfrak{W}_{\mathcal{X}}$ is given. The $\texttt{SEL}(\mathcal{W}, \mathfrak{NN})$ algorithm assigns one local NN in the set $\mathfrak{NN}$ to each abstract state in order to satisfy the reach-avoid specification $\phi_{\text{safety}} \wedge \phi_{\text{liveness}}$. Our approach is to first exclude all transitions in the MDP that can lead to violation of ϕ_{safety}, followed by selecting the optimal solution from the remaining transitions in the MDP.

A. Exclude Unsafe Transitions using Backtracking

Given a task $\mathcal{W} \in \mathfrak{W}_{\mathcal{X}}$ that specifies a set of obstacles $\{\mathcal{O}_1, \ldots, \mathcal{O}_o\}$ and a goal $\mathcal{X}_{\text{goal}}$, we use $\mathbb{X}_{\text{obst}}$ to denote the subset of abstract states that intersect the obstacles, i.e., $\mathbb{X}_{\text{obst}} = \{q \in \mathbb{X} | \exists i \in \{1, \ldots, o\}, q \bigcap \mathcal{O}_i \neq \emptyset\}$, and use $\mathbb{X}_{\text{goal}}$ to denote the subset of abstract states contained in the goal, i.e., $\mathbb{X}_{\text{goal}} = \{q \in \mathbb{X} | q \subseteq \mathcal{X}_{\text{goal}}\}$.

Algorithm 2 computes the set of safe states and safe controller partitions using an iterative backward procedure introduced in [8]. With the set of unsafe states initialized to be the obstacles (line 1 in Algorithm 2), the algorithm backtracks unsafe states until a fixed point is reached, i.e., it can not find new unsafe states (line 2-4 in Algorithm 2). The set of safe initial states $\mathcal{X}_{\text{init}}^{\mathcal{W}}$ is the union of all the abstract states that are identified to be safe (line 6 in Algorithm 2). Furthermore, it computes the function $P_{\text{safe}}^{\mathcal{W}} : \mathbb{X}_{\text{safe}}^{\mathcal{W}} \to 2^{\mathbb{P}}$, which assigns a subset of safe controller partitions $P_{\text{safe}}^{\mathcal{W}}(q) \subseteq \mathbb{P}$ at each abstract state $q \in \mathbb{X}_{\text{safe}}^{\mathcal{W}}$. Again, we use the superscript $\mathcal{W}$ to emphasize the dependency of $\mathcal{X}_{\text{init}}^{\mathcal{W}}$, $\mathbb{X}_{\text{safe}}^{\mathcal{W}}$ and $P_{\text{safe}}^{\mathcal{W}}$ on the task $\mathcal{W}$.

Algorithm 2 BACKTRACK-SAFETY ($\hat{\Sigma}$, $\mathcal{W}$)

1: $\mathbb{X}^0_{\text{unsafe}} = \emptyset$, $\mathbb{X}^1_{\text{unsafe}} = \mathbb{X}_{\text{obst}}$, $k = 1$
2: **while** $\mathbb{X}^k_{\text{unsafe}} \neq \mathbb{X}^{k-1}_{\text{unsafe}}$ **do**
3: $\quad \mathbb{X}^{k+1}_{\text{unsafe}} = \{q \in \mathbb{X} | \forall \mathcal{P} \in \mathbb{P} : \text{Next}(q, \mathcal{P}) \cap \mathbb{X}^k_{\text{unsafe}} \neq \emptyset\} \cup \mathbb{X}^k_{\text{unsafe}}$
4: $\quad k = k + 1$
5: $\mathbb{X}^{\mathcal{W}}_{\text{safe}} = \mathbb{X}^{\mathcal{W}} \setminus \mathbb{X}^k_{\text{unsafe}}$
6: $\mathcal{X}^{\mathcal{W}}_{\text{init}} = \bigcup_{q \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}} q$
7: **for** $q \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}$ **do**
8: $\quad P^{\mathcal{W}}_{\text{safe}}(q) = \{\mathcal{P} \in \mathbb{P} \mid \text{Next}(q, \mathcal{P}) \cap \mathbb{X}^k_{\text{unsafe}} = \emptyset\}$
9: **Return** $\mathcal{X}^{\mathcal{W}}_{\text{init}}$, $\mathbb{X}^{\mathcal{W}}_{\text{safe}}$, $\{P^{\mathcal{W}}_{\text{safe}}(q)\}_{q \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}}$

B. Assign Controller Partition by Solving MDP

Once the set of safe controller partitions $P^{\mathcal{W}}_{\text{safe}}(q)$ is computed, the next step is to assign one controller partition in $P^{\mathcal{W}}_{\text{safe}}(q)$ to each abstract state q. In particular, we consider the problem of solving the optimal policy for the MDP $\hat{\Sigma}$ with states and controls limited to the set of safe abstract states $\mathbb{X}^{\mathcal{W}}_{\text{safe}}$ and the set of safe controller partitions $P^{\mathcal{W}}_{\text{safe}}(q)$ at $q \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}$, respectively. Since we are interested in maximizing the probability of satisfying the liveness specification ϕ_{liveness}, let the optimal value function $\hat{V}^*_k : \mathbb{X}^{\mathcal{W}}_{\text{safe}} \to [0, 1]$ map an abstract state $q \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}$ to the maximum probability of reaching the goal in $H - k$ steps from q. Using this notation, $\hat{V}^*_0(q)$ is then the maximum probability of satisfying the liveness specification ϕ_{liveness}. The optimal value functions can be solved by the Dynamic Programming (DP) recursion [13]:

$$\hat{V}_k(q, \mathcal{P}) = \mathbf{1}_{\mathbb{X}_{\text{goal}}}(q) + \mathbf{1}_{\mathbb{X}_{\text{safe}} \setminus \mathbb{X}_{\text{goal}}}(q) \sum_{q' \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}} \hat{V}^*_{k+1}(q') \hat{T}(q' | q, \mathcal{P}) \tag{6}$$

$$\hat{V}^*_k(q) = \max_{\mathcal{P} \in P^{\mathcal{W}}_{\text{safe}}(q)} \hat{V}_k(q, \mathcal{P}) \tag{7}$$

with the initial condition $\hat{V}^*_H = \mathbf{1}_{\mathbb{X}_{\text{goal}}}$, where $k = H, \ldots, 0$.

Algorithm 3 solves the optimal policy for the MDP $\hat{\Sigma}$ using the Dynamic Programming (DP) recursion (6)-(7). At time step k, the optimal controller partition $\mathcal{P}^*$ at state q is given by the maximizer of $\hat{V}_k(q, \mathcal{P})$ (line 8 in Algorithm 3). The last step is to assign a corresponding neural network to be used at all the states $x \in q$ for each $q \in \mathbb{X}_{\text{safe}}$. To that end, the activation map $\Gamma^{\mathcal{W}}_{k,\text{abs}}$ assigns the neural network indexed by $(q, \mathcal{P}^*, q'^*)$ to the abstract state q, where q'^* maximizes the transition probability $\hat{T}(q' | q, \mathcal{P}^*)$ (line 9-10 in Algorithm 3). While the activation map $\Gamma^{\mathcal{W}}_{k,\text{abs}}$ assigns a neural network index to the abstract state q, we can directly get the activation map to the actual state $x \in \mathcal{X}$ as:

$$\Gamma^{\mathcal{W}}_k(x) = \Gamma^{\mathcal{W}}_{k,\text{abs}}(\text{abs}_{\mathbb{X}}(x)).$$

In other words, given the state of the system x, we first compute the corresponding abstract state $\text{abs}_{\mathbb{X}}(x)$, and use the corresponding neural network assigned to this abstract state to control the system. Note that, unlike the definition of the activation map $\Gamma^{\mathcal{W}}$ in Problem 2.2, the activation map obtained here is time-varying as captured by the subscript $k, k = 0, \ldots, H$. This reflects the nature of the optimal solution computed by the DP regression (6)-(7).

The $\mathcal{X}^{\mathcal{W}}_{\text{init}}$ computed by Algorithm 2 along with the selection map $\Gamma^{\mathcal{W}}_{k,\text{abs}}$ returned by Algorithm 3 constitutes the $\texttt{SEL}(\mathcal{W}, \mathfrak{NN})$ algorithm.

Algorithm 3 DP-LIVENESS ($\hat{\Sigma}$, $\mathbb{X}^{\mathcal{W}}_{\text{safe}}$, $\{P^{\mathcal{W}}_{\text{safe}}(q)\}_{q \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}}$)

1: **for** $q \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}$ **do**
2: $\quad \hat{V}^*_H(q) = \mathbf{1}_{\mathbb{X}_{\text{goal}}}(q)$
3: $k = H - 1$
4: **while** $k \geq 0$ **do**
5: $\quad$ **for** $q \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}$ **do**
6: $\quad\quad \hat{V}_k(q, \mathcal{P}) = \mathbf{1}_{\mathbb{X}_{\text{goal}}}(q) + \mathbf{1}_{\mathbb{X}_{\text{safe}} \setminus \mathbb{X}_{\text{goal}}}(q) \sum_{q' \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}} \hat{V}^*_{k+1}(q') \hat{T}(q' | q, \mathcal{P})$
7: $\quad\quad \hat{V}^*_k(q) = \max_{\mathcal{P} \in P^{\mathcal{W}}_{\text{safe}}(q)} \hat{V}_k(q, \mathcal{P})$
8: $\quad\quad \mathcal{P}^* = \text{argmax}_{\mathcal{P} \in P^{\mathcal{W}}_{\text{safe}}(q)} \hat{V}_k(q, \mathcal{P})$
9: $\quad\quad q'^* = \text{argmax}_{q' \in \mathbb{X}^{\mathcal{W}}_{\text{safe}}} \hat{T}(q' | q, \mathcal{P}^*)$
10: $\quad\quad \Gamma^{\mathcal{W}}_{k,\text{abs}}(q) = (q, \mathcal{P}^*, q'^*)$
11: $\quad k = k - 1$
12: **Return** $\Gamma^{\mathcal{W}}_{k,\text{abs}}$

VI. THEORETICAL GUARANTEES

In this section, we study the theoretical guarantees of the proposed solution. We analyze the guarantees of satisfying ϕ_{safety} and ϕ_{liveness} separately.

A. Safety Guarantee

The following theorem summarizes the safety guarantees for our solution.

Theorem 6.1: Consider the dynamical model (1). Let the NN controller $\mathcal{NN}$ consists of two parts: the set of local neural networks $\mathfrak{NN}$ trained by Algorithm 1 and the selection algorithm $\texttt{SEL}$ defined by Algorithm 2 and Algorithm 3. For any task $\mathcal{W} \in \mathfrak{W}_{\mathcal{X}}$, consider the set of initial conditions $\mathcal{X}^{\mathcal{W}}_{\text{init}}$ and the activation map $\Gamma^{\mathcal{W}}_k$ computed by $\texttt{SEL}(\mathcal{W}, \mathfrak{NN})$, the following holds: $\mathcal{NN}_{[\mathfrak{NN}, \Gamma^{\mathcal{W}}_k]}, \mathcal{X}^{\mathcal{W}}_{\text{init}} \models \phi^{\mathcal{W}}_{\text{safe}}$.

The proof of Theorem 6.1 follows the same argument of the error-free case presented in [8] and hence is omitted for brevity. To take into account the model-error g, the posterior in (3) is inflated with the error bound $\mathcal{D}$. With the NN weight projection in the training of local NNs (line 6 in Algorithm 1), the resulting NN controller is guaranteed to be safe for any task $\mathcal{W} \in \mathfrak{W}_{\mathcal{X}}$ regardless the accuracy of the learned model-error by GP regression.

B. Probabilistic Optimality Guarantee

Due to the unknown model-error g, which is learned by GP regression, the liveness specification ϕ_{liveness} may not be always satisfied. However, in this subsection, we provide a bound on the probability for the trained NN controller to satisfy ϕ_{liveness}. Intuitively, this bound tells how close is the NN controller to the optimal controller, which maximizes the probability of satisfying ϕ_{liveness}.

Let value functions $V^{\mathcal{NN}}_k : \mathcal{X}_{\text{safe}} \to [0, 1]$ and $V^*_k : \mathcal{X}_{\text{safe}} \to [0, 1]$ map a state $x \in \mathcal{X}_{\text{safe}}$ to the maximum probability of reaching the goal in $H - k$ steps from x under the NN controller $\mathcal{NN}$ and some unknown optimal controller, respectively. These value functions $V^{\mathcal{NN}}_k$ and V^*_k can be solved using Dynamic Programming (DP) recursion similar to (6)-(7). The difference between $V^{\mathcal{NN}}_k$ and V^*_k measures the optimality of the NN controller $\mathcal{NN}$ by comparing it

with the optimal controller. The following theorem upper bounds this difference.

***Theorem** 6.2:* For any $x \in \mathcal{X}_{\text{safe}}$, the difference between $V_k^{\mathcal{NN}}$ and V_k^* is upper bounded as follows:

$$|V_k^{\mathcal{NN}}(x) - V_k^*(x)| \leq (H-k)(\Delta^{\mathcal{NN}} + \Delta^*) \tag{8}$$

where

$$\Delta^{\mathcal{NN}} = \max_{1\leq i\leq N'}(\Lambda_i\delta_q + \Gamma_i L_i \delta_q + \sqrt{m(n+1)}\mathcal{L}_{\mathcal{X}}\Gamma_i\delta_{\mathcal{P}})$$

$$\Delta^* = \max_{1\leq i\leq N'}(\Lambda_i\delta_q + \Gamma_i \mathcal{L}_{\mathcal{P}}\delta_q + 2\sqrt{m(n+1)}\mathcal{L}_{\mathcal{X}}\Gamma_i\delta_{\mathcal{P}}).$$

In the above theorem, $N' = |\mathbb{X}_{\text{safe}}|$ is the number of safe abstract states, δ_q and $\delta_{\mathcal{P}}$ are the discretization parameters of the state and controller space, respectively. Let $\lambda_i(y)$ and $\gamma_i(y)$ be the Lipschitz constants of the stochastic kernel $t : \mathcal{X}\times\mathcal{X}\times\mathcal{U} \to [0,1]$ at abstract state $q_i \in \mathbb{X}$, i.e., $\forall y \in \mathcal{X}$:

$$|t(y|x',u) - t(y|x,u)| \leq \lambda_i(y)\|x'-x\|, \forall x,x' \in q_i, u \in \mathcal{U}$$
$$|t(y|x,u') - t(y|x,u)| \leq \gamma_i(y)\|u'-u\|, \forall u,u' \in \mathcal{U}, x \in q_i.$$

Let L_i be the Lipschitz constant of an arbitrary local NN corresponding to a transition leaving $q_i \in \mathbb{X}_{\text{safe}}$, i,e.:

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

for any $\mathcal{P} \in P_{\text{safe}}(q_i)$ and $q' \in \text{Next}(q_i,\mathcal{P})$. Furthermore, let $\Lambda_i = \int_{\mathcal{X}_{\text{safe}}}\lambda_i(y)\mu(dy)$, $\Gamma_i = \int_{\mathcal{X}_{\text{safe}}}\gamma_i(y)\mu(dy)$, $\sup_{x\in\mathcal{X}_{\text{safe}}}\|x\| \leq \mathcal{L}_{\mathcal{X}}$, and $\sup_{K\in\mathcal{P}^{K\times b}}\|K\| \leq \mathcal{L}_{\mathcal{P}}$.

Notice that the upper bound in Theorem 6.2 can be arbitrarily small when the grid size $\delta_{\mathcal{X}}$ and $\delta_{\mathcal{P}}$ approach zero. Due to the space limit, we present the proof of Theorem 6.2 in the extended version [11].

VII. RESULTS

Consider a wheeled robot with the state vector $x = [\zeta_x, \zeta_y, \theta]^\top \in \mathcal{X} \subset \mathbb{R}^3$, where ζ_x, ζ_y denote the coordinates of the robot, and θ is the heading direction. In the form of (1), the priori known nominal model f is given by:

$$\begin{aligned}\zeta_x^{(t+\Delta t)} &= \zeta_x^{(t)} + \Delta t\ v\ \cos(\theta^{(t)})\\ \zeta_y^{(t+\Delta t)} &= \zeta_y^{(t)} + \Delta t\ v\ \sin(\theta^{(t)})\\ \theta^{(t+\Delta t)} &= \theta^{(t)} + \Delta t\ u^{(t)}\end{aligned} \tag{9}$$

where the velocity $v = 3$ and discrete time step size $\Delta t = 0.1$. We also consider an unknown model-error g, which is bounded by $[0, 0.1]$ in x and y dimensions. The system is controlled by NN controllers trained by our algorithms, 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 $\mathcal{P}^{K\times b}$ considered to be a hyperrectangle.

As the first step of our algorithm, we discretized the state space $\mathcal{X} \subset \mathbb{R}^3$ 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 Figure 2. We uniformly partitioned the controller space $\mathcal{P}^{K\times b}$ into 240 hyperrectangles, and computed the reachable sets using TIRA [14].

During the offline training, we trained a subset of local networks $\mathfrak{NN}_{\text{part}} \subset \mathfrak{NN}$ (Algorithm 4 in [11]). Specifically, the local NNs are trained for Task $\mathcal{W}_1$ (the first subfigure in the upper row of Figure 2), and the set $\mathfrak{NN}_{\text{part}}$ consists of 658 local NNs. The total time for training and projecting weights of the 658 local networks in $\mathfrak{NN}_{\text{part}}$ is 2368 seconds. At runtime, we tested the trained NN controller in five unseen tasks $\mathcal{W}_i$, $i = 2,\ldots,6$, and the corresponding trajectories are shown in Figure 2. In case a local NN has not been trained offline, we employ transfer learning to fast learn the missing NNs at runtime (Algorithm 5 in [11]).

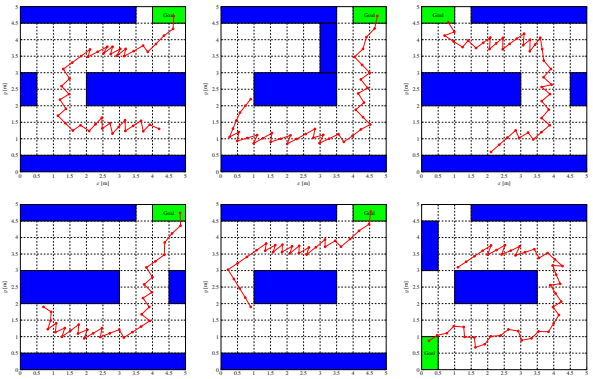

Fig. 2: The upper row shows trajectories for Task $\mathcal{W}_1$, $\mathcal{W}_3$, $\mathcal{W}_5$, and the lower row corresponds to Task $\mathcal{W}_2$, $\mathcal{W}_4$, $\mathcal{W}_6$.

REFERENCES

[1] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: from classical to evolutionary approaches and applications," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 276–295, 2017.

[2] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1568–1577.

[3] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *arXiv preprint arXiv:2004.05439*, 2020.

[4] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1126–1135.

[5] C. Fernando, J. Sygnowski, S. Osindero, J. Wang, T. Schaul, D. Teplyashin, P. Sprechmann, A. Pritzel, and A. Rusu, "Meta-learning by the baldwin effect," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2018, pp. 1313–1320.

[6] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Provably convergent policy gradient methods for model-agnostic meta-reinforcement learning," *arXiv preprint arXiv:2002.05135*, 2020.

[7] H. Liu, R. Socher, and C. Xiong, "Taming maml: Efficient unbiased meta-reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4061–4071.

[8] X. Sun and Y. Shoukry, "Provably correct training of neural network controllers using reachability analysis," *arXiv:2102.10806*, 2021.

[9] C. E. Rasmussen and C. Williams, "Gaussian processes for machine learning," *the MIT Press*, 2006.

[10] R. Pascanu, G. Montufar, and Y. Bengio, "On the number of response regions of deep feed forward networks with piece-wise linear activations," *arXiv preprint arXiv:1312.6098*, 2013.

[11] X. Sun, W. Fatnassi, U. Santa Cruz, and Y. Shoukry, "Provably safe model-based meta reinforcement learning: An abstraction-based approach," *arXiv preprint arXiv:2109.01255*, 2021.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[13] I. Tkachev, A. Mereacre, J.-P. Katoen, and A. Abate, "Quantitative automata-based controller synthesis for non-autonomous stochastic hybrid systems," in *Proceedings of the 16th ACM International Conference on Hybrid Systems: Computation and Control*, 2013, pp. 293–302.

[14] P.-J. Meyer, A. Devonport, and M. Arcak, "Tira: toolbox for interval reachability analysis," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, 2019, pp. 224–229.