Compositional Learning and Verification of Neural Network Controllers

RADOSLAV IVANOV*, KISHOR JOTHIMURUGAN*, STEVE HSU, SHAAN VAIDYA, RAJEEV ALUR, and OSBERT BASTANI, University of Pennsylvania, USA

Recent advances in deep learning have enabled data-driven controller design for autonomous systems. However, verifying safety of such controllers, which are often hard-to-analyze neural networks, remains a challenge. Inspired by compositional strategies for program verification, we propose a framework for compositional learning and verification of neural network controllers. Our approach is to decompose the task (e.g., car navigation) into a sequence of subtasks (e.g., segments of the track), each corresponding to a different mode of the system (e.g., go straight or turn). Then, we learn a separate controller for each mode, and verify correctness by proving that (i) each controller is correct within its mode, and (ii) transitions between modes are correct. This compositional strategy not only improves scalability of both learning and verification, but also enables our approach to verify correctness for arbitrary compositions of the subtasks. To handle partial observability (e.g., LiDAR), we additionally learn and verify a mode predictor that predicts which controller to use. Finally, our framework also incorporates an algorithm that, given a set of controllers, automatically synthesizes the pre- and postconditions required by our verification procedure. We validate our approach in a case study on a simulation model of the F1/10 autonomous car, a system that poses challenges for existing verification tools due to both its reliance on LiDAR observations, as well as the need to prove safety for complex track geometries. We leverage our framework to learn and verify a controller that safely completes any track consisting of an arbitrary sequence of five kinds of track segments.

CCS Concepts: • **Computer systems organization** → **Embedded and cyber-physical systems**; Real-time systems; • **Computing methodologies** → Reinforcement learning.

Additional Key Words and Phrases: neural networks, verification, compositional reasoning

ACM Reference Format:
Radoslav Ivanov, Kishor Jothimurugan, Steve Hsu, Shaan Vaidya, Rajeev Alur, and Osbert Bastani. 2021. Compositional Learning and Verification of Neural Network Controllers. *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (January 2021), 25 pages. https://doi.org/10.1145/3477023

1 INTRODUCTION

Deep reinforcement learning is a promising approach to solving challenging control problems, such as control from perception [46], multi-agent planning problems [43], autonomous driving while interacting with humans [15], or planning through contact such as walking [17] or grasping [10]. The basic premise is to learn a neural network (NN) controller directly mapping observations to actions. However, ensuring safety in these settings is challenging due to the complexity in

*Both authors contributed equally to this research.

Authors' address: Radoslav Ivanov; Kishor Jothimurugan; Steve Hsu; Shaan Vaidya; Rajeev Alur; Osbert Bastani, University of Pennsylvania, USA.

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Embedded Software (EMSOFT), 2021.

1539-9087/2021/1-ART1 $15.00
https://doi.org/10.1145/3477023

formally reasoning about NN models. For instance, small perturbations in their inputs can lead to unexpected changes in their output [66] and this can adversely affect control performance [42]. Thus, it is critical to formally verify the safety of the NN controller to guarantee safety under a wide range of inputs and operating conditions. Even learning NN controllers for complex tasks remains challenging [41, 45, 51], since existing approaches do not scale beyond tasks with short planning horizons.

As a consequence, there has been a great deal of interest in safe reinforcement learning [4, 12, 18] and verifying that the learned NN controller satisfies a given safety property [38, 42]. We focus on closed-loop safety, where the goal is to ensure that the controller, composed with a model of the robot dynamics and its environment, is safe over the entire planning horizon—e.g., that an autonomous car does not run into an obstacle, or a walking robot does not fall over. We consider the setting where the NN controller is learned in simulation, and the goal is to verify the learned controller.

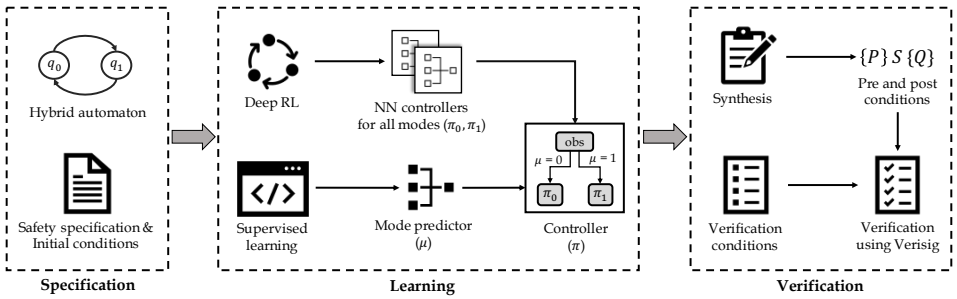


Fig. 1. An overview of our compositional learning and verification framework.

A key challenge in achieving this goal is proving safety for the full closed-loop system. One approach is to unroll the safety property over a finite horizon [38]. However, this approach becomes intractable as the planning horizon becomes large. In particular, existing verification algorithms rely on overapproximating the dynamics [16], and the approximation error accumulates over the horizon. Thus, very precise abstractions are required to verify safety for long horizons. An alternative approach is to establish the existence of an inductive invariant such as a Lyapunov function [18, 67] or a control barrier function [5, 57]. This strategy reduces the problem to a verification problem over a single step, since it suffices to prove that a candidate invariant is inductive and that it implies safety. However, establishing such an invariant can be intractable for high-dimensional state spaces, especially when using neural network controllers with many parameters.

These challenges are further exacerbated for real-world robotics systems, which are typically only partially observable (e.g., the inputs to the NN are LiDAR scans), and the geometry of the environment is a priori unknown (e.g., the robot is acting in a building with an unknown layout of hallways).

To address these challenges, we propose a framework for compositional learning and verification of NN controllers[1] (Figure 1). Our framework is inspired by classical techniques such as Hoare logic [34] for compositional program verification. The idea is to verify a program by decomposing it into modular components, devising verification conditions (VCs) for all components that suffice to prove safety, and then proving that each VC holds for its respective component.

[1]Although the proposed framework can be used with any verification tool, we use Verisig [38] for closed-loop verification. Since Verisig supports fully-connected NNs with sigmoid/tanh activations, we focus on this class of NNs as well.

In particular, our framework leverages this strategy to both learn an NN controller to solve a given control task and verify the learned controller. First, we decompose the task into a sequence of sub-tasks, where each sub-task is associated with a precondition (e.g., the region of the state space where the robot starts) and a postcondition (e.g., the region where the robot ends up). This decomposition is designed to satisfy two properties:

- **Mode safety and progress:** For any single sub-task, using the NN controller from any state satisfying the precondition should safely transition the system to a state satisfying the postcondition within some bounded number of steps.
- **Switching safety:** The postcondition of one sub-task should imply the precondition of the next.

As long as these two properties are satisfied, the NN controller is guaranteed to be safe for the entire planning horizon. Furthermore, these two properties are sufficient to guarantee a particular liveness property which states that any finite sequence of sub-tasks will be completed eventually. Intuitively, our strategy combines verification over a finite horizon (i.e., mode safety) with establishing inductive invariants (i.e., switching safety), except that the inductive invariants are established at the level of sub-tasks rather than individual steps in the system. Formally, we model the system as a hybrid automaton [5, 7, 53]—i.e., a model of the system is a set of modes of operation, with differential equations specifying the state dynamics of each mode; in our approach, the discrete transitions encode switching from one sub-task to the next. Many practical control tasks can be decomposed in such a way—e.g., navigation problems can be decomposed into sequences of sub-goals.

Given a hybrid automaton, our framework performs the following steps:

- **Compositional learning:** First, it learns a separate NN controller for each mode, using shaped rewards to encourage it to satisfy mode safety and progress. An added advantage of this approach is that we can use simpler NNs that are easier to both train and verify.
- **Pre/postcondition synthesis:** Next, it synthesizes *candidate pre/postconditions* (i.e., a candidate pair of pre- and postconditions for each mode) that satisfy switching safety and are consistent with a set of traces obtained by simulating the system with the learned controllers.
- **Compositional verification:** Finally, it uses hybrid systems verification tools [16, 38] to independently check mode safety and progress for each mode.

The second step builds on recent work on invariant synthesis [31]. In particular, our synthesis algorithm uses testing to identify *implication examples* that connect the different (pre/postcondition) sets, and then tries to synthesize candidate pre/postconditions consistent with these examples.

One challenge is that in partially observed environments, the controller may not know when one sub-task has been completed and/or what the next sub-task is. To address this issue, we additionally train a *mode predictor*, which is a separate NN that predicts whether the postcondition for the current sub-task holds in the current state and if so, predicts the next sub-task. This mode predictor is incorporated into the overall controller. To ensure correctness, the safety and progress conditions are verified with respect to the full compositional controller (including the mode predictor). For instance, consider a robot navigating in a building with an unknown layout; then, it may not know if the next segment is to go straight, turn left, or turn right. Our approach naturally handles this setting since it proves safety for arbitrary compositions of the sub-tasks as long as the switching safety property is satisfied. Thus, the sequence of sub-tasks can be chosen dynamically based on observations of the environment—e.g., if a robot comes to a left turn at the end of a hallway, then the mode predictor would determine that the next sub-task is to make that left turn. Therefore, our framework enables us to learn and verify a controller that generalizes to multiple tasks composed of the same set of sub-tasks.

(a) Straight (b) Right (c) Left (d) Sharp Right (e) Sharp Left

Fig. 2. Different types of track segments.

(a) (b) (c)

Fig. 3. Example tracks decomposed into segments.

We evaluate our approach on a challenging benchmark—namely, a simulation model of the F1/10 autonomous racing system [1], where the goal is for an NN-controlled car to complete a track without crashing into the walls. Verifying safety for this system has received recent attention [38]; however, these approaches do not scale to verifying safety beyond short time horizons on a single, predefined track, due to two main reasons. First, the controller must rely on high-dimensional LiDAR observations of the environment, which poses challenges for scalability. Second, we ideally want to ensure safety for a wide variety of complex track geometries. As a consequence, this system is beyond the reach of existing state-of-the-art verification techniques.

We demonstrate that our framework can successfully learn and verify an NN controller for this system, by decomposing tracks into sequences of individual segments. In particular, we consider sub-tasks that include going straight or executing four different kinds of turns, and verify safety for any sequence of such sub-tasks. We also provide evidence that training a monolithic controller for an example track is significantly harder than our compositional learning approach.

In summary, our contributions are:

- A framework for compositional verification of NN controllers for hybrid systems (Section 3).
- An algorithm for automatically inferring pre/postconditions given a controller π, as well as a compositional learning algorithm for training π.
- An extensive evaluation[2] via a case study based on a model of the F1/10 autonomous car (Sections 5 & 6).

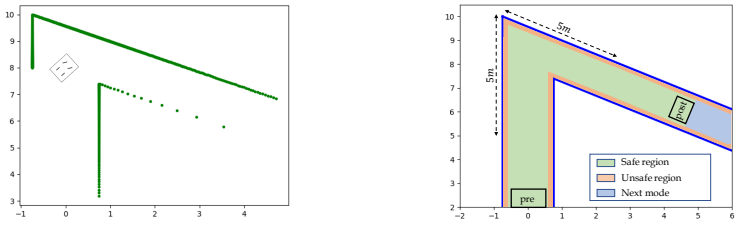
2 OVERVIEW

In this section, we give a brief overview of our approach using the F1/10 autonomous racing system as a motivating example.

F1/10 car. The objective is to safely navigate the autonomous F1/10 car along a racing track to complete a lap as quickly as possible. The safety property states that the car should not crash into the track walls. Ignoring modes for now, the state space is $\mathcal{X} \subseteq \mathbb{R}^4$ (a state $x \in \mathcal{X}$ denotes the 2D position, speed, and angle of the car), the action space is $\mathcal{U} \subseteq \mathbb{R}^2$ (an action $u \in \mathcal{U}$ consists of acceleration and steering angle), and the dynamics are the bicycle dynamics [58].

We assume the track is decomposed into a sequence of segments, where each segment is either a straight track, a left/right turn, or a sharp left/right turn as shown in Figure 2; these are the five

[2]Our implementation is available at https://github.com/keyshor/autonomous_car_verification.



(a) A LiDAR scan. (b) Different regions in the segment.

Fig. 4. A sharp right turn.

modes of the system. Our goal is not to learn and verify a controller for a given specific track, but rather to learn and verify a controller that works for *all* tracks constructed by composing these segments. Some example tracks are shown in Figure 3.

The F1/10 car observes its environment with a LiDAR sensor, which uses laser rays to determine the distance to the nearest obstacle along different directions. In particular, it produces an observation $o \in O \subseteq \mathbb{R}^m$, where each $o_i \in \mathbb{R}$ corresponds to an angle $\psi \in [-135, 135]$ and denotes the distance from the car position to the nearest wall in the direction $\vartheta + \psi$, where ϑ is the angle the car is currently facing. An example of a scan is shown in Figure 4a; each green point is the obstacle observed by one of the $m = 1081$ LiDAR rays.

Control problem. Our goal is to learn a controller $\pi : O \to \mathcal{U}$ that maps LiDAR observations to actions. Designing a safe controller for the F1/10 car is challenging due to the high-dimensional observation space. One approach is to train a neural network (NN) controller π using reinforcement learning, and then verify post-hoc that π is safe. This technique has been used to verify that the car can safely navigate a right turn [39]. However, existing verification approaches [22, 38, 68] do not scale to more complex tasks such as the tracks in Figure 3—even when the track is known ahead of time—due to the long planning horizon.

Compositional verification (fully observed). For now, let us assume that the controller π is given and that the state is fully observed, and describe how we verify that π is safe. We also assume that we are given a *pre-region* and a *post-region* for each mode, which are subsets of the state space such that the car always starts in the pre-region of the mode and ends in its post-region. Intuitively, membership in the pre-region (resp., post-region) corresponds to the precondition (resp., postcondition) for that mode. An example of the pre- and post-regions for the sharp right turn mode is shown in Figure 4b. These regions are chosen so that the system immediately and safely transitions from the post-region of any mode q to the pre-region of some subsequent mode q' (i.e., *switching safety*). If we know the sequence of track segments, then the choice of q' is unique. In our case, since we do not know the sequence of track segments a priori, we prove switching safety for *every* pair of modes q, q', which, together with mode safety and progress guarantees that the car safely completes *any* track consisting of an arbitrary sequence of these five kinds of segments. Finally, to prove mode safety and progress, it suffices to verify that π safely navigates the car from the pre-region of each mode to the corresponding post-region without crashing.

Compositional verification (partially observed). Verification is more challenging when the state is partially observed—e.g., π only has access to LiDAR observations. We assume π is decomposed into a *mode predictor* μ together with a controller π^q for each mode q. Then, π uses π^q, where q is the predicted mode at the current step.

Importantly, we do not assume that the mode predictor is correct; thus, π may use the incorrect controller. For example, in the case of the sharp right turn, if the LiDAR range is smaller than the

distance of the corner from the entry region, there will be regions where the mode predictor cannot distinguish the sharp turn segment from a straight segment using just LiDAR observations (see Figure 4b). Thus, we need to prove that the full controller π is correct, even if μ is wrong. This involves simultaneously reasoning about the controllers $\pi^{q''}$ for *all* modes q'', along with the mode predictor μ.

Compositional learning. We use deep reinforcement learning to train one neural network controller π^q for each mode q to drive the car from the pre-region to the post-region. Since the controller can only observe the LiDAR observations, we also train a mode predictor that predicts the current mode from the observations. We can do so using supervised learning from observations encountered while training π^q.

Importantly, we find that our compositional approach benefits not only verification but also learning. In particular, we can train simpler neural networks with fewer parameters, and training is less likely to get stuck at local maxima that are characteristic of long planning horizons.

Candidate pre/post-region synthesis. Finally, manually specifying the pre- and post-regions for each mode can be challenging. We propose an algorithm for automatically inferring these regions. Our algorithm, based on invariant inference [26, 31, 60], alternates between synthesizing *candidate pre/post-regions* that are consistent with all the example traces generated so far, and generating new example traces using π.

In particular, the synthesis algorithm uses the example traces to identify both *unsafe examples* z from which π is known to be unsafe, and *implication examples* $z \to z'$, which say that z' is reachable from z using π. Then, it represents the pre- and post-regions as boxes in $\mathbb{R}^n$, and infers a set of boxes that are consistent with the identified examples. Finally, it uses the inferred pre/post-regions to try and verify that π is safe.

3 COMPOSITIONAL VERIFICATION

In this section, we describe our framework for compositional verification of controllers. Our model of the system is based on hybrid automata [7, 8, 53] tailored to our setting. We define safety and liveness in our context and show that we can reduce safety and liveness to a set of verification conditions (VCs) that are local to the modes of the hybrid automaton and can be checked using existing verification tools.

3.1 Problem Formulation

Dynamics. We consider a hybrid dynamical system with states $z \in \mathcal{Z}$ and actions $\mathcal{U} \subseteq \mathbb{R}^k$. We assume the state space has structure $\mathcal{Z} = Q \times \mathcal{X}$, where Q is a finite set of *modes* and $\mathcal{X} \subseteq \mathbb{R}^n$ is the continuous component of the state space. We denote the states in mode q by $\mathcal{Z}^q = \{q\} \times \mathcal{X}$. Within a mode $q \in Q$, the dynamics are given by a function $f : \mathcal{Z} \times \mathcal{U} \to \mathbb{R}^n$; in particular, the system evolves according to the differential equation $\dot{x}(t) = f(z(t), u(t))$ (with respect to time t). When there is no ambiguity, we simply write $\dot{x} = f(z, u)$. The mode transitions are given by a relation $\mathcal{T} \subseteq \mathcal{Z} \times \mathcal{Z}$, where an edge $z \to z' \in \mathcal{T}$ means the system can transition from state z to state z'. We let $\mathcal{Z}_F = \{z \in \mathcal{Z} \mid \exists z' \in \mathcal{Z} \text{ s.t. } z \to z' \in \mathcal{T}\}$ denote the set of states where mode transitions can occur. The mode transitions are assumed to be *urgent*—i.e., a mode transition occurs as soon as the system reaches some $z \in \mathcal{Z}_F$; we assume that $\mathcal{Z}_F$ is closed so this property is well-defined.

Intuitively, the corresponding discrete time dynamics are given by $z_+ = z'$ if $z \to z' \in \mathcal{T}$ and $z_+ = (q, x + f(z, u) \cdot \Delta t)$ otherwise. Note that the mode transitions are nondeterministic, since the condition $z \to z' \in \mathcal{T}$ may be satisfied by multiple $z' \in \mathcal{Z}$. This nondeterminism is needed to capture settings where the sequence of sub-tasks is a priori unknown. In our F1/10 example, at a state z about to exit the current mode, transitions $z \to (q', x')$ exist for all modes

$q' \in \{\text{straight}, \text{left turn}, \ldots\}$. Finally, our goal is to control the system based on observations $o \in \mathcal{O} \subseteq \mathbb{R}^m$; in particular, an observation function $h : \mathcal{Z} \to \mathcal{O}$ maps states to observations. If the system is fully observable, $\mathcal{O}$ can be taken to be $\mathcal{Z}$ with $h(z) = z$ for all $z \in \mathcal{Z}$.

We formally represent the dynamical system as a hybrid automaton which is defined as:

Definition 3.1. A *hybrid automaton* $\mathcal{A}$ is a tuple $\mathcal{A} = (\mathcal{Q}, \mathcal{X}, \mathcal{U}, \mathcal{T}, \mathcal{O}, f, h)$.

Control. A controller is a function $\pi : \mathcal{O} \to \mathcal{U}$, where $u = \pi(h(z))$ specifies the action to use in state z. We use $f(z, \pi, t) \in \mathcal{Z}$ to denote the state reached at time $t \in \mathbb{R}_{\geq 0}$ by evolving the system according to $\dot{x} = f(z, \pi(h(z)))$. Furthermore, let $F(z, \pi, t) \subseteq \mathcal{Z}$ denote the set of states visited until time t—i.e., $F(z, \pi, t) = \{f(z, \pi, t') \mid 0 \leq t' \leq t\}$.

We decompose π into controllers $\pi^q : \mathcal{O} \to \mathcal{U}$ designed to be used in mode $q \in \mathcal{Q}$, and a *mode predictor* $\mu : \mathcal{O} \to \mathcal{Q}$ that predicts the current mode. Then, we have $\pi(o) = \pi^q(o)$ where $q = \mu(o)$. We do not assume that the mode predictor is always correct—i.e., we may have $\mu(o) = q$ even though the current mode is $q' \neq q$, in which case π would use the wrong controller.

Trajectories. Next, we describe the space of trajectories that may be generated by a given controller π. Since the dynamics are continuous-time, the trajectory is a curve in the state space parameterized by time $t \in \mathbb{R}_{\geq 0}$. However, formally reasoning about this representation is difficult. Instead, we represent a trajectory as an infinite sequence $\rho = (z_0 \xrightarrow{t_0} z_1 \xrightarrow{t_1} \cdots)$, where $t_i \in \mathbb{R}_{\geq 0}$ for all $i \in \mathbb{N}$. In particular, an edge $z_i \xrightarrow{t_i} z_{i+1}$ in ρ says that the system transitions from z_i to z_{i+1} in time t_i. For clarity, we omit the t_i's from ρ when it is not needed. There are two kinds of transitions $z_i \to z_{i+1}$ that can occur:

- **Continuous transition:** This kind of transition occurs when $z_i \notin \mathcal{Z}_F$. Then, the system evolves according to the continuous dynamics f—i.e., $z_{i+1} = f(z_i, \pi, t_i)$, where $t_i > 0$. We assume that no mode transition is triggered—i.e., $f(z_i, \pi, t) \notin \mathcal{Z}_F$ for all $t \in [0, t_i)$. We denote such a transition by $z_i \to_f z_{i+1}$.
- **Mode transition:** This kind of transition occurs when $z_i \in \mathcal{Z}_F$. Then, the system instantaneously transitions to some z_{i+1} such that $z_i \to z_{i+1} \in \mathcal{T}$—i.e., $t_i = 0$. We denote such a transition by $z_i \to_{\mathcal{T}} z_{i+1}$.

We assume all trajectories are *non-Zeno*—i.e., $\sum_{i=0}^{\infty} t_i = \infty$. It is only necessary to consider Zeno trajectories if subsequent mode transitions can occur after arbitrarily small amounts of time, which cannot happen if the system requires a minimum amount of time before triggering the next mode transition. In our F1/10 example, the car must traverse an entire segment to trigger another mode transition, which cannot happen arbitrarily quickly since velocity is bounded from above.

Correctness properties. We consider a safety property specified as a region $\mathcal{Z}_{\text{safe}} \subseteq \mathcal{Z}$ in which we expect the system to stay. In addition, we assume given a set of initial states $\mathcal{Z}_0 \subseteq \mathcal{Z}_{\text{safe}}$ from which we want to ensure safety.

Definition 3.2. A controller π is *safe* for a hybrid automaton $\mathcal{A}$ if for any trajectory ρ starting from $z_0 \in \mathcal{Z}_0$, for all $i \in \mathbb{N}$, we have $f(z_i, \pi, t) \in \mathcal{Z}_{\text{safe}}$ for all $t \in [0, t_i]$.

That is, the system should be safe for the duration of any trajectory generated using π from an initial state. Next, liveness says the system should switch modes infinitely often.

Definition 3.3. A controller π is *live* for a hybrid automaton $\mathcal{A}$ if for any trajectory ρ starting from $z_0 \in \mathcal{Z}_0$, we have $z_i \to_{\mathcal{T}} z_{i+1}$ for infinitely many $i \in \mathbb{N}$.

3.2 Verification Conditions

Our verification algorithm reduces the problem of verifying safety and liveness to a set of verification conditions (VCs).

Pre- and post-regions. Following our compositional approach, our VCs decompose the problem into properties of individual modes or pairs of modes. For each mode q, we assume given a *pre-region* $\mathcal{X}^q_{\text{pre}} \subseteq \mathcal{X}$ and a *post-region* $\mathcal{X}^q_{\text{post}} \subseteq \mathcal{X}$. In addition, we define $\mathcal{Z}^q_{\text{pre}} = \{q\} \times \mathcal{X}^q_{\text{pre}}$ and $\mathcal{Z}^q_{\text{post}} = \{q\} \times \mathcal{X}^q_{\text{post}}$. Intuitively, the precondition (resp., postcondition) for q is membership in its pre-region (resp., post-region). We require that pre- and post-regions satisfy the following conditions, which we call *compatibility conditions (CCs)* since they are not checked by the verifier, but are directly enforced when we generate the pre/post regions.

Definition 3.4 (CC 1)**.** We have $\mathcal{Z}_0 \subseteq \bigcup_{q \in Q} \mathcal{Z}^q_{\text{pre}}$.

That is, every initial state is contained in a pre-region.

Definition 3.5 (CC 2)**.** We have $\bigcup_{q \in Q} \mathcal{Z}^q_{\text{post}} \subseteq \mathcal{Z}_F$.

That is, every state in the post-region triggers a mode transition; intuitively, the post-region should only include states that "exit" the mode. Now, we have two kinds of VCs:

- **Mode safety and progress:** For each mode $q \in Q$, the system safely transitions from $\mathcal{Z}^q_{\text{pre}}$ to $\mathcal{Z}^q_{\text{post}}$.
- **Switching safety:** For each pair of modes $q, q' \in Q$ with a mode transition $(q, x) \to (q', x') \in \mathcal{T}$, the system safely transitions from $\mathcal{Z}^q_{\text{post}}$ to $\mathcal{Z}^{q'}_{\text{pre}}$.

First, our VC for mode safety and progress is:

Definition 3.6 (VC 1)**.** For any $z \in \mathcal{Z}^q_{\text{pre}}$, there exists $t \in \mathbb{R}_{>0}$ such that $f(z, \pi, t) \in \mathcal{Z}^q_{\text{post}}$, $F(z, \pi, t) \subseteq \mathcal{Z}_{\text{safe}}$, and $f(z, \pi, t') \notin \mathcal{Z}_F$ for all $t' \in [0, t)$.

That is, π safely transitions the system from any state in the pre-region of mode q to the post-region of q. The last condition is needed to ensure that the system does not trigger a mode transition $z \to z' \in \mathcal{T}$ at some state $z \notin \mathcal{Z}^q_{\text{post}}$. That is, $f(z, \pi, t)$ is the first state reached that triggers a mode transition (such a state exists since we have assumed $\mathcal{Z}_F$ is closed).

Remark 3.7. *Although VC 1 is local to a mode $q \in Q$, it is a property of the full controller π which includes the mode predictor μ and controllers $\pi^{q'}$ for all $q' \in Q$.*

Next, our VC for switching safety is:

Definition 3.8 (VC 2)**.** For all $z \in \mathcal{Z}^q_{\text{post}}$ and all $z \to z' \in \mathcal{T}$, we have $z' \in \mathcal{Z}^{q'}_{\text{pre}}$ for some $q' \in Q$.

That is, for every state z in a post-region and every mode transition $z \to z'$, the target state z' is contained in the pre-region of another mode q'.

Together, CCs 1 & 2 and VCs 1 & 2 imply that π is safe and live for $\mathcal{A}$. First, CC1 ensures that the initial states satisfy the precondition of some mode q. Then, VC 1 says that the precondition of mode q implies the postcondition of mode q. Next, VC 2 and CC 2 together say that the postcondition of mode q implies the precondition of another mode q'.

Theorem 3.9. *Given controller π for hybrid automaton $\mathcal{A}$, if CCs 1 & 2 and VCs 1 & 2 hold, then π is safe and live for $\mathcal{A}$.*

We give a proof in Appendix A, and describe how we use verification tools [16, 38] to verify the VCs in Appendix B.

Algorithm 1 Compositional learning and synthesis. *Inputs:* Hybrid automaton $\mathcal{A}$ and initial candidate pre/post-regions B_0. *Output:* A verified controller π or FAIL. *Hyperparameters:* Number of synthesis iterations $K \in \mathbb{N}$.

procedure LearnController($\mathcal{A}, B_0$)
  $\pi \leftarrow \mathsf{Learn}(\mathcal{A}, B_0)$
  $B_\pi \leftarrow \mathsf{Synthesize}(\mathcal{A}, \pi, B_0)$
  if $B_\pi = \emptyset$ **then return** FAIL
  if $\mathsf{Verify}(\mathcal{A}, \pi, B_\pi)$ **then return** π
  return FAIL
procedure Synthesize($\mathcal{A}, \pi, B$)
  $E \leftarrow \emptyset$
  for $i \in \{1, \ldots, K\}$ **do**
    $E \leftarrow E \cup \mathsf{Test}(\mathcal{A}, \pi, B)$
    $B \leftarrow \mathsf{Infer}(E)$
    if $B = \emptyset$ **then return** $\emptyset$
  return B

4 COMPOSITIONAL LEARNING AND SYNTHESIS

Our overall framework is summarized in Algorithm 1. Suppose we are given initial *pre/post-regions* B_0—i.e., a pre- and a post-region for every mode $q \in Q$. Then, the method consists of the following steps:

- **Learning:** Train a controller π that tries to drive the system from every state in the pre-region of each mode q to the post-region of q, where we use the pre/post regions in B_0.
- **Pre/post-region synthesis:** Synthesize new candidate pre/post-regions B_π for π.
- **Verification:** Use the algorithm in Section 3 with B_π to try and prove that π is safe and live.

A natural choice for the initial pre/post-regions is to take $\mathcal{Z}^q_{\text{pre}} = \mathcal{Z}_0 \cap \mathcal{Z}^q$ and $\mathcal{Z}^q_{\text{post}} = \mathcal{Z}_F \cap \mathcal{Z}^q$ for all $q \in Q$. The above procedure can fail because of two reasons: either synthesis fails (i.e., no set of pre/post-regions consistent with the generated examples exists) or verification fails. In either case, we retry the above steps with modified rewards for learning and/or a different choice of initial pre/post-regions. In our experiments, we retried our procedure (Algorithm 1) a few (3-4) times with different reward functions until we were able to verify the learned controller.

The subroutine for synthesizing a candidate set of pre/post-regions alternates between the following two steps:

- **Testing:** Generate new examples using testing.
- **Inference:** Infer a candidate set of pre/post-regions B based on examples E generated so far.

The examples E include both *implication examples* $z \to z' \in \mathcal{Z}^2$ such that z' is reachable from z using π, and *unsafe examples* $z \in \mathcal{Z}$ that reach an unsafe state using π.

Below, we describe our pre/post-region inference algorithm (Section 4.1) and our testing algorithm (Section 4.2), as well as our compositional learning algorithm (Section 4.3).

4.1 Pre/Post-Region Inference

Problem formulation. We describe our algorithm for inferring pre- and post-regions given a set of examples. First, we represent the regions using boxes–i.e., products of intervals.

Definition 4.1. A *box* $b \in \mathcal{B}$ in $\mathbb{R}^n$ is defined by $b = \prod_{i=1}^n [x_i, y_i] \subseteq \mathbb{R}^n$, where $x_i \le y_i$ for all $i \in \{1, ..., n\}$.

We synthesize a set of boxes $B = \{b_\alpha \mid \alpha \in \{\text{pre}, \text{post}\} \times Q\}$ denoting the pre- and post-regions of all the modes. For now, we assume given *lower and upper bounds* $b_\alpha^\bot, b_\alpha^\top$ for all $\alpha \in A = \{\text{pre}, \text{post}\} \times Q$. As discussed below, these bounds are used to enforce CCs 1 & 2. Then, our goal is to find boxes b_α for all $\alpha \in A$ satisfying $b_\alpha^\bot \subseteq b_\alpha \subseteq b_\alpha^\top$, such that taking $\mathcal{X}_{\text{pre}}^q = b_{(\text{pre},q)}$ and $\mathcal{X}_{\text{post}}^q = b_{(\text{post},q)}$, VCs 1 & 2 are satisfied. We denote the set of lower and upper bounds by $B^\bot$ and $B^\top$ respectively.

First, we describe the kinds of examples that are available. Examples are states (or pairs of states) that encode a *necessary* condition for the VCs to hold—i.e., if the invariant does not satisfy an example, then it cannot possibly satisfy the VCs, but the converse is not true. First, we have states from which using π is unsafe.

Definition 4.2. An *unsafe example* is a pair (α, x) where $\alpha = (\text{pre}, q) \in A$ and $x \in \mathcal{X}$ such that there exists $t \in \mathbb{R}_{\geq 0}$ with $f((q,x),\pi,t) \notin \mathcal{Z}_{\text{safe}}$ and $f((q,x),\pi,t') \notin \mathcal{Z}_F$ for all $t' \in [0,t)$.

Next, we have examples that correspond to pairs of states z and z' where z' is reachable from z.

Definition 4.3. An *implication example* is a pair $(\alpha, x) \to (\alpha, x')$ with $\alpha, \alpha' \in A$ and $x, x' \in \mathcal{X}$ such that either (i) $\alpha = (\text{post}, q)$ and $\alpha' = (\text{pre}, q')$, with $(q,x) \to (q',x') \in \mathcal{T}$, or (ii) $\alpha = (\text{pre}, q)$ and $\alpha' = (\text{post}, q)$, and there exists $t \in \mathbb{R}_{\geq 0}$ with $(q,x') = f((q,x),\pi,t) \in \mathcal{Z}_F$, $F((q,x),\pi,t) \subseteq \mathcal{Z}_{\text{safe}}$ and $f((q,x),\pi,t') \notin \mathcal{Z}_F$ for all $t' \in [0,t)$.

Given these two kinds of examples, our goal is to synthesize a candidate set of boxes that is consistent with them—i.e., it excludes examples that are inconsistent with our VCs.

Definition 4.4. Given lower and upper bounds $B^\bot, B^\top$, unsafe examples C and implication examples I, a candidate set of boxes B is *consistent* if the following hold:

- For all $(\alpha, x) \in C$, we have $x \notin b_\alpha$.
- For all $(\alpha, x) \to (\alpha', x') \in I$, $x \in b_\alpha \Rightarrow x' \in b_{\alpha'}$.
- For all $\alpha \in A$, we have $b_\alpha^\bot \subseteq b_\alpha \subseteq b_\alpha^\top$.

Furthermore, B is *minimal* if for any candidate set of boxes $\tilde{B}$ satisfying these conditions, $b_\alpha \subseteq \tilde{b}_\alpha$ for all $\alpha \in A$.

Given bounds $B^\bot, B^\top$, unsafe examples C, and implication examples I, the Infer subroutine used in Algorithm 1 returns a minimal consistent candidate set of boxes (if one exists, returning $\varnothing$ otherwise).

Algorithm. Next, we describe our algorithm for synthesizing minimal set of boxes given a set of examples. This algorithm is outlined in Algorithm 2. Our approach is to reduce the synthesis problem to the following:

Definition 4.5 (Consistent Box)**.** Given positive examples $X^+ \subseteq \mathbb{R}^n$, negative examples $X^- \subseteq \mathbb{R}^n$ and boxes $b^\bot, b^\top$, the *(minimal) consistent box* is

$$b^* = \operatorname*{arg\,min}_{b \in \mathcal{B}} \prod_{i=1}^{n} (y_i - x_i) \quad \text{subj. to} \quad X^+ \subseteq b, \quad b \cap X^- = \varnothing, \quad b^\bot \subseteq b \subseteq b^\top.$$

That is, the goal is to find the smallest box that includes X^+ and excludes X^-. This problem can be solved efficiently—in particular, let $b = \prod_{i=1}^n [x_i, y_i]$, where $x_i = \min\{x_i' \mid x' \in X^+\} \cup \{x_i^\bot\}$ and $y_i = \max\{x_i' \mid x' \in X^+\} \cup \{y_i^\bot\}$ where $b^\bot = \prod_{i=1}^n [x_i^\bot, y_i^\bot]$. Then, return b if $b \cap X^- = \varnothing$ and $b \subseteq b^\top$; otherwise, we return $\varnothing$ (i.e., no such box exists).

Our synthesis algorithm initializes positive examples $X_\alpha^+ = \varnothing$, and negative examples X_α^- to be the unsafe examples, for each $\alpha \in A$. Then, at each iteration, it independently synthesizes a

Algorithm 2 Pre/post-region inference. *Inputs:* Implication & unsafe examples E. *Output:* Candidate pre/post-regions B. *Hyperparamters:* $B^{\bot}, B^{\top}$

procedure INFER(E)
  $I, C \leftarrow E$
  for $\alpha \in A$ **do**
    $X^{+}_{\alpha} \leftarrow \emptyset$
    $X^{-}_{\alpha} \leftarrow \{x \mid (\alpha, x) \in C\}$
  while true **do**
    for $\alpha \in A$ **do**
      $b_{\alpha} \leftarrow \mathsf{ConsistentBox}\left(X^{+}_{\alpha}, X^{-}_{\alpha}, b^{\bot}_{\alpha}, b^{\top}_{\alpha}\right)$
      if $b_{\alpha} = \emptyset$ **then return** $\emptyset$
    $\psi \leftarrow$ true
    for $(\alpha, x) \rightarrow (\alpha', x') \in I$ **do**
      if $x \in b_{\alpha}$ **and** $x' \notin b_{\alpha'}$ **then**
        $X^{+}_{\alpha'} \leftarrow X^{+}_{\alpha'} \cup \{x'\}$
        $\psi \leftarrow$ false
    if ψ **then return** $\{b_{\alpha} \mid \alpha \in A\}$

consistent box b_{α} to be the minimal consistent box[3] for positive examples X^{+}_{α}, negative examples X^{-}_{α}, and boxes $b^{\bot}_{\alpha}, b^{\top}_{\alpha}$. Next, it handles implication examples in I by expanding the sets X^{+}_{α} for $\alpha \in A$. In particular, it checks if any of the implication examples $(\alpha, x) \rightarrow (\alpha', x') \in I$ violate the current candidate invariant—i.e., $x \in b_{\alpha}$ but $x' \notin b_{\alpha'}$. If so, it requires that $x' \in b_{\alpha'}$ by adding x' to $X^{+}_{\alpha'}$. It continues the iterative process until either all examples in I are satisfied, in which case it returns the current candidate boxes B, or the consistent box subroutine fails, in which case it returns $\emptyset$.

Suppose there exists a set of minimal consistent boxes $\{b^{*}_{\alpha} \mid \alpha \in A\}$. Then, our algorithm maintains the invariant that the current candidate boxes $\{b_{\alpha} \mid \alpha \in A\}$ are contained in the minimal consistent boxes—i.e., $b_{\alpha} \subseteq b^{*}_{\alpha}$ for all $\alpha \in A$. Therefore, when dealing with an inconsistent implication example $(\alpha, x) \rightarrow (\alpha', x') \in I$ with $x \in b_{\alpha}$, we can infer that $x' \in b^{*}_{\alpha'}$ and hence it correctly adds x' to $X^{+}_{\alpha'}$, forcing $b_{\alpha'}$ in the next iteration to include x'. Since we deal with any implication example at most once and we deal with at least one implication example in every iteration (except the last iteration), we have:

THEOREM 4.6. *Algorithm 2 terminates after at most $|I|$ iterations and computes a set of minimal consistent boxes if one exists and returns $\emptyset$ otherwise.*

Choosing upper and lower bounds. Finally, we use the upper and lower bounds to handle CCs 1 & 2. First, CC 1 says that for every state $(q, x) \in \mathcal{Z}_0$, we have $x \in b_{\alpha}$ where $\alpha = (\text{pre}, q)$. Thus, to ensure this condition holds, it suffices to choose $b^{\bot}_{\alpha}$ such that $\mathcal{X}^{q}_{0} \subseteq b^{\bot}_{\alpha}$ for all $q \in \mathcal{Q}$. Similarly, CC 2 says that for every $x \in b_{\alpha}$ with $\alpha = (\text{post}, q)$, we have $(q, x) \in \mathcal{Z}_F$; thus, it suffices to choose $b^{\top}_{\alpha} \subseteq \mathcal{X}^{q}_{F}$ for all $q \in \mathcal{Q}$.

4.2 Testing

Our testing subroutine takes as input candidate pre/post-regions B and uses simulated trajectories from random start states to try and discover examples that are inconsistent with our VCs. Our testing algorithm is summarized in Algorithm 3.

[3] Although X^{+}_{α} is initialized to $\emptyset$, b_{α} is not empty since it has to contain $b^{\bot}_{\alpha}$.

Algorithm 3 Testing to check verification conditions. *Inputs:* Hybrid automaton $\mathcal{A}$, NN controller π, and candidate pre/post-regions B. *Output:* Implication & unsafe examples E. *Hyperparameters:* Horizon $T \in \mathbb{R}_{>0}$, iterations $K \in \mathbb{N}$.

procedure TEST($\mathcal{A}, \pi, B$)
  $E \leftarrow \emptyset$
  for $i \in \{1, ..., K\}$ **do**
    $\alpha \leftarrow (\beta, q) \sim \text{Uniform}(A)$
    $z \leftarrow (q, x)$ where $x \sim \mathcal{P}(b_\alpha)$
    if $\beta = \text{pre}$ **then**
      $\zeta \leftarrow F(z, \pi, T)$ (stop as soon as ζ enters $\mathcal{Z}_F$)
      $z' \leftarrow (q, x') = f(z, \pi, T)$
      if $\zeta \not\subseteq \mathcal{Z}_{\text{safe}}$ **then** $E.C.\text{Add}((\alpha, x))$
      if $z' \notin \mathcal{Z}^q_{\text{post}}$ **then** $E.I.\text{Add}\big((\alpha, x) \rightarrow ((\text{post}, q), x')\big)$
    else
      $z' \leftarrow (q', x') \sim \mathcal{P}(\{z' \mid z \rightarrow z' \in \mathcal{T}\})$
      if $z' \notin \mathcal{Z}^{q'}_{\text{pre}}$ **then** $E.I.\text{Add}\big((\alpha, x) \rightarrow ((\text{pre}, q'), x')\big)$
  return E

Algorithm 4 Compositional learning to try and satisfy verification conditions. *Inputs:* Hybrid automaton $\mathcal{A}$, candidate pre/post-regions B. *Output:* Compositional controller π.

procedure LEARN($\mathcal{A}, B$)
  for $q \in Q$ **do**
    $p(x) = p^q(x)$ where $x \sim \mathcal{P}(b_{(\text{pre},q)})$
    $r(x) = r^q(x, b_{(\text{post},q)})$
    $\pi^q \leftarrow \text{ReinforcementLearning}(f^q, p(x), r(x))$
  $p(q, x) = p(q)p(x)$ where $q \sim \text{Uniform}(Q)$, $x \sim \mathcal{P}_{\pi^q}$
  $\mu \leftarrow \text{SupervisedLearning}(p(z), h(z))$
  return π

At a high level, it samples trajectories $\zeta \subseteq \mathcal{Z}$ starting from random states $z = (q, x)$, where $\alpha = (\beta, q) \sim \text{Uniform}(A)$, and $x \sim \mathcal{P}(b_\alpha)$—e.g., we can take $\mathcal{P}(b_\alpha)$ to be the uniform distribution over b_α. Then, it checks whether ζ is an unsafe or an implication example that is inconsistent with B; if so, it adds z to $E.C$ and/or $z \rightarrow z'$ (z' is the last state in ζ) to $E.I$, respectively. Finally, it returns the set of examples E which is then used by our pre/post-region inference algorithm.

4.3 Controller & Mode Predictor Learning

We describe our approach for learning the compositional controller π, which involves learning the controller π^q for each mode $q \in Q$ as well as learning the mode predictor μ. Our approach is summarized in Algorithm 4.

Controllers. First, we use reinforcement learning to learn the controllers π^q for each mode q. We parameterize $\pi^q = \pi^q_\theta$ as a neural network $\pi^q_\theta : O \rightarrow U$ mapping observations to actions. The inputs to the reinforcement learning algorithm are the dynamics f^q for mode q, a distribution $p(x)$ over initial states x, and a reward function $r : X \rightarrow \mathbb{R}$. For the initial state distribution, we assume given a distribution $\mathcal{P}(b_{(\text{pre},q)})$ over the pre-region of q—e.g., the uniform distribution. The reward function should encourage the system to reach the next region. We can use any reinforcement

learning algorithm in conjunction with these inputs to learn π^q. We use the twin delayed deep deterministic policy gradient (TD3) algorithm [28], which is a more stable variant of the popular deep deterministic policy gradient (DDPG) algorithm [48].

Mode predictor. Next, we learn the mode predictor using supervised learning. To do so, we need to construct a training set consisting of input-output examples $(o, q) \in \mathcal{O} \times \mathcal{Q}$, where observation o is the input and mode q is the ground truth mode. To do so, we sample states $z = (q, x)$, compute the observations $o = h(z)$, and then construct the training examples (o, q). For the distribution $p(z) = p(q, x)$ over states z, we use the uniform distribution over q and the distribution $\mathcal{P}_{\pi^q}$ over x visited by the controller π^q. The reason we use this distribution over x is that it is the distribution of x values that the mode predictor will encounter when running π. Finally, we parameterize μ using a neural network $\mu_\theta : \mathcal{O} \times \mathcal{Q} \to [0, 1]$ (i.e., predict the probability $\mu_\theta(q \mid o)$ of mode $q \in \mathcal{Q}$ given observation $o \in \mathcal{O}$).

5 SYSTEM MODELING

We briefly describe the F1/10 car model used in our evaluation, and how we train the controllers π^q and the mode predictor μ.

Dynamics model. We use the model in [39]. We use vector notations $\vec{x} \in \mathcal{X}$ and $\vec{u} \in \mathcal{U}$ for clarity. The car dynamics are given by a kinematic bicycle model with 4D state space $\vec{x} = (x, y, \vartheta, v) \in \mathcal{X} \subseteq \mathbb{R}^4$, including 2D position (x, y), orientation ϑ, and velocity v. The actions are $\vec{u} = (a, \phi) \in \mathcal{U} \subseteq \mathbb{R}^2$, where a denotes throttle and ϕ is the orientation of the front wheels. We assume throttle is constant at $a = 16$ (resulting in a top speed of 2.4m/s), whereas ϕ is set by the controller at a sampling rate of 10Hz. The dynamics are governed by the following differential equations (with respect to time):

$$\begin{aligned} \dot{x} &= v \cdot \cos(\vartheta) \qquad & \dot{v} &= -c_a \cdot v + c_a \cdot c_m \cdot (a - c_h) \\ \dot{y} &= v \cdot \sin(\vartheta) \qquad & \dot{\vartheta} &= \frac{v}{\ell} \cdot \tan(\phi) \end{aligned} \tag{1}$$

where $c_a = 1.633$ is the car's acceleration constant, $c_m = 0.2$ is its motor constant, $c_h = 4$ is its hysteresis constant, and $\ell = 0.45$ is the its length. We consider two different observation models.

State-feedback system. First, we consider a variant of the F1/10 car with state-feedback—i.e., $\mathcal{O} = \mathcal{Z}$ and the controller $\pi^q : \mathcal{Z} \to \mathcal{U}$ has access to the true state of the car; similarly, the mode predictor $\mu : \mathcal{Z} \to \mathcal{Q}$ outputs the true mode $\mu(q, x) = q$. This setting allows us to evaluate the controllers in isolation of the mode detector.

LiDAR observation model. Next, we consider a LiDAR based observation model. A LiDAR scan consists of a number of laser rays emanating at a range of degrees with respect to the car's orientation. For each ray, the car receives the distance to the nearest object reached by the ray, or the maximum LiDAR range of 5m if no obstacle is in that range. The controller has access to the LiDAR measurements only and cannot observe the position, orientation or the velocity of the car. Similar to prior work [39], we focus on a LiDAR scan with 21 rays since the complexity of the verification task increases exponentially with the number of rays. More details can be found in Appendix C.

Tracks. We consider tracks consisting of a sequence of segments, each corresponding to one of five modes: right and left 90-degree turns, right and left 120-degree turns, and straight segments. Each segment is 1.5m wide and is of a fixed length. Straight segments can be of arbitrary lengths but must be sufficiently long to allow for an inductive proof of our VCs; see Section 6. The segments are lined up with the end of one segment meeting the start of the next one. We represent each segment as having coordinates where the top-most corner is at the origin. Then, a mode transition

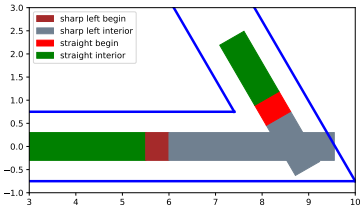


Fig. 5. Regions for training the mode predictor

$z \to z' \in \mathcal{T}$ is an (instantaneous) affine change of coordinates[4] to bring the car into this coordinate system. Furthermore, there is a mode transition from any state at the end of any segment to a state at the start of every segment, thereby modeling all possible tracks in a single hybrid automaton.

Safety. The safety property is that the car should not run into any of the walls. We model the car as a square of size $\gamma = 0.15m$ and the walls as line segments. Then, the car should not intersect the wall—i.e., $\mathcal{X}^q_{\text{safe}} = \{\vec{x} \in \mathcal{X} \mid \forall w \in \text{walls}[q] \,.\, \|(x, y) - (w_x, w_y)\|_\infty \geq \gamma\}$.

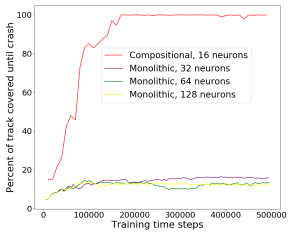
Controller. For the state-feedback system, each controller has 5 inputs: the x and y distances to each of the two corners in the turn and the car's orientation relative to the segment. For LiDAR-feedback, each controller has 21 inputs corresponding to the LiDAR rays. We use reinforcement learning to train the controllers π^q. We represent the policy as an NN $\pi^q = \pi^q_\theta$ with two fully connected layers with tanh activations and 16 neurons per layer for the state-feedback system and 64 neurons per layer for the LiDAR system. We use a uniform distribution on the pre-region as the initial state distribution. We use a reward function that aims to achieve two goals: (i) stay in the safe region, (ii) stay in regions where we can compose the different verification results. The second goal is necessary for our compositional approach to work, since we need the car to visit the post-region when started in the pre-region. To achieve this goal, we train controllers that stay in the middle of each segment after turns, with the exception of the sharp turns, where it seems challenging to train controllers to stay in the middle. More details can be found in Appendix D.

Mode predictor. We decompose the mode predictor into two parts: (i) a *new mode predictor* $\mu^p : \mathcal{O} \to \mathcal{Q}$ and (ii) an *exit detector* $\mu^q : \mathcal{O} \to \{0, 1\}$, one for each mode q. Intuitively, μ^p is used to determine the mode q the system is about to enter; once q is determined, the corresponding μ^q is run until it predicts that system has exited mode q (at which point μ^p is run again). Since standard control systems are sampled periodically, let o_k denote the observation at sampling step k. Then, the output of the overall mode predictor at step k, q_k, is defined as follows:

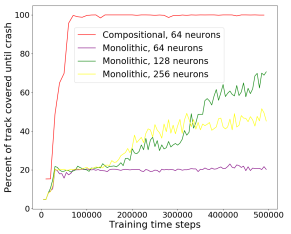
$$\begin{aligned} q_k &= q_{k-1} && \text{if } \mu^{q_{k-1}}(o_k) = 0 \\ q_k &= \mu^p(o_k) && \text{if } \mu^{q_{k-1}}(o_k) = 1, \end{aligned} \tag{2}$$

where $q_0 = \mu^p(o_0)$. This decomposition simplifies mode predictor training since each individual NN is trained either only on data from one mode (in the case of μ^q) or on data from the pre-regions of all the modes (in the case of μ^p). Specifically, we divide each track segment into two regions: one consisting of the 50cm at the beginning of the segment, and the other of the rest of the segment; examples are shown in Figure 5. Each exit detector μ^q is trained to predict 0 (i.e., "not exited") on LiDAR scans taken in its own mode q (both in the beginning region and the remainder region) and 1 (i.e., "exited") on scans from the beginning region of other modes $q' \neq q$. The new mode detector μ^p

[4]The post-region of one segment is contained within the pre-region of the next segment (after change of coordinates) since mode transitions are instantaneous and do not involve movement of the car.



(a) State-feedback system.



(b) LiDAR-feedback system.

Fig. 6. Training evolution for state- and LiDAR-feedback controllers. The "Compositional" controller curve shows the combined number of training steps for controllers trained on each individual turn, whereas the "Monolithic" controllers are trained on the track from Figure 3c. All NNs have two fully connected layers, with the number of neurons per layer indicated in the legend. Results are averaged over five runs per setup.

is trained to predict the mode in which the LiDAR scan was taken, with half the training examples from beginning regions of each mode and half from remaining regions. This strategy allows the mode predictor to recover from incorrect predictions by μ^p—i.e., if $q_k = \mu^p(o_k)$ is an error, then μ^{q_k} should predict that q_k is wrong at the next step (i.e., $\mu^{q_k}(o_{k+1})$ should be 1) and ask μ^p to update its prediction. All NNs have two fully connected layers with tanh activations and 32 neurons per layer. More details can be found in Appendix E.

Verification. We use the Verisig tool [38] for verification. Verisig verifies neural networks with smooth activation functions (e.g., sigmoid, tanh) by transforming the networks into hybrid systems. The neural network hybrid system is then composed with the dynamics model, thereby converting the closed-loop problem into a hybrid system verification problem that is solved by Flow* [16].

6 EXPERIMENTAL RESULTS

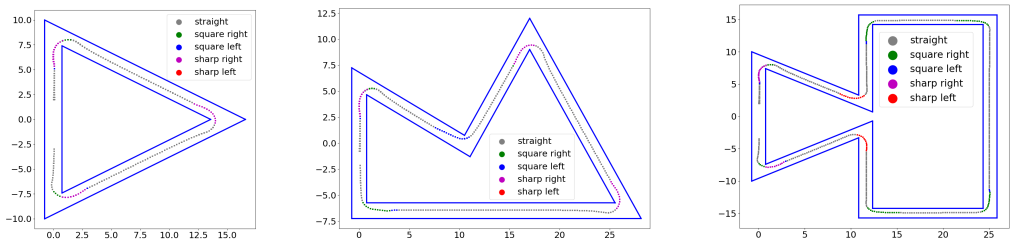
We evaluate our framework on the F1/10 car, aiming to address the following research questions:

- Can our compositional learning strategy improve the scalability of reinforcement learning?
- Can our compositional verification algorithm be used to prove that the learned controller safe and live for arbitrary sequences of track segments?

6.1 Benefits of Compositional Learning

For both state-feedback and LiDAR systems, we trained two controllers: one for the 90-degree right turn and one for the 120-degree right turn. Since left and right turns are symmetric, we use the right-turn controller for a left turn by reflecting the observations and negating the control input. We also use the 90-degree controller in straight segments, since it is able to steer the car close to the middle.

To illustrate the benefit of compositional learning, we trained a single NN controller for the full track in Figure 3c. We used increasingly larger NNs (with 32, 64, 128 neurons per layer for state-feedback and 64, 128 and 256 neurons per layer for observation-feedback); however, none safely completed a lap in the entire track. Figures 6a & 6b show the performance of these controllers along with the performance of the compositional controller (the individual controllers combined with a pre-trained mode predictor) on the full track, as a function of the number of training steps. As expected, training is fast and stable for our compositional controller, whereas the monolithic ones are unable to converge to a stable policy. While it may be possible to train a monolithic controller using a larger NN or a different reward function, our results provide evidence that the compositional



(a) (b) (c)

Fig. 7. Example trajectories with LiDAR-feedback using the compositional controller. The color of each position indicates the mode predictor output.

approach is simpler and requires less expert domain knowledge, both in reinforcement learning and in the specific system.

Our compositional controller performs well (and can be verified, as shown in the verification experiments below) on *all* tracks constructed using the five kinds of segments. Figure 7 shows the simulated trajectories of the compositional controller on the tracks in Figure 3. While the mode predictor sometimes predicts the wrong mode when far from the turn, it eventually switches to the correct one selecting the appropriate controller for the remainder of the turn.

6.2 Pre/Post-Region Synthesis

Our synthesis algorithm is used to compute pre/post-regions for all the modes. We abuse notation and use y to denote the y-distance (in meters) from the start of the segment and x to denote the distance from the left wall. The synthesized pre-region is the same for all the modes because we have implication examples from the post-region of every mode to the pre-region of each mode. The pre-region computed for the LiDAR-feedback system is given by $x \in [0.75, 0.83]$, $y \in [0, 0.24]$, $\vartheta \in [\frac{\pi}{2} - 0.0042, \frac{\pi}{2} + 0.002]$, and $v \in [2.4, 2.4]$. The post-regions are the corresponding boxes at the end of each segment. For example, the post-region computed for the 90-degree right turn is given by $x \in [8, 8.24]$, $y \in [5.67, 5.75]$, $\vartheta \in [-0.0042, 0.002]$ and $v \in [2.4, 2.4]$.

6.3 Verification Results

We focus on verification results for the LiDAR-feedback system; state-feedback is similar (see Appendix G). Note that verifying safety for the LiDAR-feedback system is challenging due to multiple discrete computations. First, the controller π has a discrete internal state due to use of the mode predictor, which creates additional modes in the hybrid automaton given to Flow*. In addition, if a given LiDAR ray can reach multiple walls in a given reachable set of states, then each case needs to be encoded as a different mode of the hybrid automaton. During verification, a reachable set can get split into multiple reachable sets due to case analysis, generating multiple *branches* each of which is a verification instance of its own. The number of such branches can be exponential in the number of modes since branching occurs dynamically as time progresses. Thus, it is essential to keep the uncertainty as small as possible as reachable sets are propagated through time. However, closed-loop verification tools such as Verisig rely on overapproximating the system's reachable set, and this approximation error can grow quickly over time. A standard strategy is to partition the initial set and verify each subset separately. This process can also suffer from exponential blowup, but it alleviates the compounding uncertainty issue. Another benefit of this partitioning is that we can parallelize verification.

Mode	# instances	# branches	Verification time (hours)	
			Composed system	NN only
90-degree right	1000	8.22	5.80	0.31
90-degree left	1000	6.77	7.08	0.35
120-degree right	1000	12.17	12.22	0.580
120-degree left	1000	14.62	11.41	0.531
90-degree right*	20	2.30	2.18	0.110
90-degree left*	20	1.95	2.38	0.110
120-degree right*	20	3.45	2.97	0.150
120-degree left*	20	2.15	2.87	0.130
straight initial	20	1.35	0.93	0.043
straight inductive	1	1.00	0.04	0.002

Table 1. Verification results for LiDAR observations. Verification times and number of branches are averaged across all the instances for that mode. The cases labeled $*$ do not handle discrete sampling of the controller. "Composed system" is the (average) time for fully verifying a single instance, and "NN only" is the time spent propagating reachable sets through the NNs during closed loop verification.

An additional verification challenge is that for a real system, we need to sample the controller at discrete points in time. Thus, we cannot switch modes at the exact point in time after the mode transition happens. We can account for this error by enlarging the pre-region—e.g., for a controller sampled at 0.1s intervals, we need to enlarge the pre-region by 0.25m in the y-direction. We report results both with and without this modification, in order to illustrate the challenge introduced by an extra dimension of uncertainty.

Verification of turns. The results are summarized in Table 1. We use slightly larger pre/post-regions than those computed by the synthesis algorithm to account for overapproximation errors introduced in verification. We split the initial set by increments of 0.005 along the x-dimension and 0.005 along the y-dimension, resulting in 1000 verification instances per turn. We verify them in parallel on an 80-core machine running at 1.2GHz. Although the left and right turns are symmetric, we need to verify them separately since the full compositional controller may not be symmetric.

As shown in Table 1, most instances took a few hours to verify on average, depending mostly on the number of branches (of reachable sets) through the hybrid automaton. Note that verification requires significantly less computation for the case when no y uncertainty due to the discrete control sampling is considered. Note also that the 120-degree turn verification is much more challenging because of the larger open space in the turn, resulting in more branching due to LiDAR rays reaching different walls. Furthermore, the controller needs to take a more drastic action to make the turn, which makes the reachable set computation harder since the NN is sensitive to small changes to its input, which amplifies approximation errors. In particular, there were some instances with more that 70 branches, taking more than 60 hours to finish.

Verification of straight segments. The straight segment verification is different since straights can be of arbitrary length (above some minimum). Thus, we need an inductive argument to perform verification. Ideally, we would establish an inductive invariant $\mathcal{X}_{\text{inv}}$ such that if the car starts in $\mathcal{X}_{\text{inv}}$ at step k, then it remains in $\mathcal{X}_{\text{inv}}$ until step $k+1$ while making progress along the track (i.e., in the y-direction).

For a typical choice of such a region, the car might leave but then return after multiple steps. For example, if $\vartheta = \frac{\pi}{2} - 0.005$ and $x = 0.85$, then the car is facing to the right and will reach a value of x greater than 0.85 as soon as it moves; however, our NN controller eventually steers the car back

to a smaller x value. We find it is significantly easier to identify a *recurrent* set such that the system returns to this set periodically. Let[5] $\tilde{\mathcal{X}}_{\text{post}} = \{(x, \vartheta, v) \mid \exists y \,.\, (x, y, \vartheta, v) \in \mathcal{X}_{\text{post}}\}$. Then, we compute a subset $\tilde{\mathcal{X}}_{\text{rec}} \subseteq \tilde{\mathcal{X}}_{\text{post}}$, and prove (i) the car reaches $\tilde{\mathcal{X}}_{\text{rec}}$ from $\mathcal{X}_{\text{pre}}$ in i steps, and (ii) if the car starts in $\tilde{\mathcal{X}}_{\text{rec}}$, then it returns to $\tilde{\mathcal{X}}_{\text{rec}}$ in j steps for some $j \in \mathbb{N}$; during this time, it always stays in $\tilde{\mathcal{X}}_{\text{post}}$ and makes progress in the y-direction. Intuitively, (i) is the base case and (ii) is the inductive case of a safety proof by induction. We do not need to consider y-uncertainty for this case; thus, the number of instances is just 20. We give more details in Appendix F.

7 RELATED WORK

Verified machine learning. Multiple approaches have recently been proposed for analyzing machine learning systems. A key focus has been verifying robustness of NNs [11, 33, 66]—e.g., by casting the problem into a satisfiability modulo theory (SMT) program [23, 36, 42], a mixed-integer linear program (MILP) [21], a semi-definite program (SDP) [25], a relaxed linear program [72], or a reachability problem [9, 32, 71]. Alternatively, abstraction techniques have been developed by computing Lipschitz constant bounds [25]. Furthermore, there has also been recent interest in verifying other properties of machine learning systems such as fairness [3, 13, 29], or semantic properties of computer vision via rare event simulation, including applications to safety for self-driving cars [19, 27, 40, 54].

Verifying control & hybrid systems. There has been significant interest in verifying controllers for hybrid systems. Traditional techniques rely on inferring invariant such as Lyapunov functions [18, 67] or control barrier functions [5, 57]. More recent techniques have been proposed for checking safety and reachability properties in hybrid systems [16, 44]. Compositional reasoning principles for hybrid systems have also been developed [6, 49], but their focus is mainly on decomposing the problem of reasoning about concurrent composition of hybrid automata into reasoning about individual components. Finally, there has also been work on compositional control synthesis through control verification [67]; however, these techniques are designed for state-feedback systems.

Verifying control & hybrid systems with NN components. The methods in the previous paragraph are not directly applicable to systems with NN components due to scalability issues. The first class of approaches that address this problem are compositional verification methods [52, 56]. These techniques employ assume-guarantee reasoning such that if the NN component satisfies a given input-output (IO) property (as verified using a NN verification tool [42]), then the closed-loop system is safe as well. The challenge with these methods is that in general, it is challenging to reduce a closed-loop property into an IO property for the NN (essentially, this problem is equivalent to synthesizing a loop invariant). Thus, these methods only apply when such a reduction is available.

Alternatively, researchers have developed methods to directly reason about the closed-loop system by adapting control & hybrid system reachability methods. In particular, several techniques have been developed to analyze closed-loop systems with NN controllers [22, 35, 38, 65, 68]. These works combine the above-mentioned ideas from NN verification with standard hybrid automata verification tools [16, 44]—e.g., by transforming the NN into an equivalent hybrid system [38], approximating it with a polynomial with error bounds [22], or using other set representations such as star sets [68]. Additionally, it is possible to approximate the NN with a simpler controller such as a program [69] or a decision tree [12] that is easier to analyze. In some settings, these policies can achieve performance comparable to that of NN controllers [37]; however, these kinds of models typically do not work well with LiDAR observations. Our approach makes use of closed-loop verification tools (namely, Verisig [38]) as building blocks in the compositional argument.

[5]We only consider x, ϑ, and v since y does not affect the observations.

Safe reinforcement learning and control. In addition to post-hoc verification, there exist methods to develop safe-by-design controllers. In particular, it is possible to develop a trajectory-following controller through developing a contraction metric [63] but this method only provides probabilistic guarantees. Furthermore, researchers have combined reinforcement learning and safe control in simplex-like architectures, where a safe controller overrides the NN controller when a control action is deemed unsafe [30, 47, 70, 73]. Such an architecture can be naturally augmented with a hierarchical controller, so as to enable more complex control tasks [30]. This work is the closest to ours in the sense that a hierarchical controller is developed with safety guarantees; however, our formulation not only allows us to verify safety for a given task, but it also allows us to compose tasks through bounded liveness verification (since we use Verisig to verify that the system always reaches the *post-region* when started from the *pre-region*). In addition, in the former approach [30] the user also needs to provide a safe controller, which may be challenging in high-dimensional tasks. Finally, there has also been work on safe exploration [2, 14, 50]; however, these techniques typically only scale to finite or low-dimensional state spaces.

Hierarchical reinforcement learning. Several reinforcement learning approaches have been developed where the controller has a hierarchical structure, similar to the controller employed in our paper [20, 30, 59, 64]. In these methods, a high-level controller/planner decides on the next high-level action and selects a low-level controller to implement the specific actuator commands. These methods are related in the sense that our framework requires a hierarchical controller in order for the compositional verification argument to work. In contrast to our work, these approaches do not typically verify the learned controller. At the same time, since our approach is agnostic to the methodology used to train the controllers, it would be interesting to investigate whether using these methods leads to easier verifiability or an alternative compositional argument.

Invariant synthesis. There has been work on automatically inferring program invariants from tests [24, 26]. Recent work has leveraged ideas similar to counterexample-guided inductive synthesis (CEGIS) [62], that alternate between synthesizing an invariant that satisfies the current counterexamples and using testing and verification to identify new counterexamples [55, 60, 61]. A particular challenge is handling implication examples [31], which connect different parts of the invariant. We designed a novel pre/post-region synthesis algorithm based on these ideas.

8 CONCLUSION

We proposed a compositional framework for learning and verifying NN-based controllers. We showed that our framework can be used to learn a controller for the F1/10 system that is provably safe for arbitrary tracks consisting of sequences of five primitive segments. While we focused on the F1/10 system as a challenge problem, we believe our approach is applicable to realistic systems well beyond it.

Acknowledgements. We thank the anonymous reviewers for their helpful suggestions. This work was partially supported by ONR award N00014-20-1-2115 and by DARPA Assured Autonomy project under Contract No. FA8750-18-C-0090.

REFERENCES

[1] F1/10 Autonomous Racing Competition. http://f1tenth.org.
[2] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin. Reachability-based safe learning with gaussian processes. In *Conference on Decision and Control*, pages 1424–1431. IEEE, 2014.
[3] A. Albarghouthi, L. D'Antoni, S. Drews, and A. V. Nori. Fairsquare: probabilistic verification of program fairness. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–30, 2017.
[4] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. In *AAAI*, 2018.

[5] R. Alur. Formal verification of hybrid systems. In *International Conference on Embedded Software*, pages 273–278, 2011.

[6] R. Alur and T. Henzinger. Modularity for timed and hybrid systems. In *CONCUR '97: Eighth International Conference on Concurrency Theory*, LNCS 1243, pages 74–88. Springer-Verlag, 1997.

[7] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, pages 209–229. Springer, Berlin, Heidelberg, 1992.

[8] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[9] G. Anderson, S. Pailoor, I. Dillig, and S. Chaudhuri. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In *Programming Language Design and Implementation*, pages 731–744, 2019.

[10] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[11] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi. Measuring neural net robustness with constraints. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2016.

[12] O. Bastani, Y. Pu, and A. Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, 2018.

[13] O. Bastani, X. Zhang, and A. Solar-Lezama. Probabilistic verification of fairness properties via concentration. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–27, 2019.

[14] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems*, pages 908–918, 2017.

[15] Z. Cao, E. Bıyık, W. Z. Wang, A. Raventos, A. Gaidon, G. Rosman, and D. Sadigh. Reinforcement learning based control of imitative policies for near-accident driving. In *Robotics: Science and Systems*, 2020.

[16] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[17] S. Collins, A. Ruina, R. Tedrake, and M. Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307 (5712):1082–1085, 2005.

[18] J. Daafouz, P. Riedinger, and C. Iung. Stability analysis and control synthesis for switched systems: a switched lyapunov function approach. *IEEE Transactions on Automatic Control*, 47(11):1883–1887, 2002.

[19] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia. Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *CAV*, pages 432–442. Springer, 2019.

[20] J. Duan, S. E. Li, Y. Guan, Q. Sun, and B. Cheng. Hierarchical reinforcement learning for self-driving decision-making without reliance on labelled driving data. *IET Intelligent Transport Systems*, 14(5):297–305, 2020.

[21] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138. Springer, 2018.

[22] S. Dutta, X. Chen, and S. Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *HSCC*, pages 157–168. ACM, 2019.

[23] R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[24] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1-3):35–45, 2007.

[25] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 11427–11438, 2019.

[26] C. Flanagan and K. R. M. Leino. Houdini, an annotation assistant for esc/java. In *International Symposium of Formal Methods Europe*, pages 500–517. Springer, 2001.

[27] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 63–78, 2019.

[28] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[29] S. Galhotra, Y. Brun, and A. Meliou. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 498–510, 2017.

[30] B. Gangopadhyay, H. Soora, and P. Dasgupta. Hierarchical program-triggered reinforcement learning agents for automated driving. *arXiv preprint arXiv:2103.13861*, 2021.

[31] P. Garg, C. Löding, P. Madhusudan, and D. Neider. Ice: A robust framework for learning invariants. In *International Conference on Computer Aided Verification*, 2014.

[32] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy*, 2018.

[33] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.
[34] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
[35] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.
[36] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.
[37] J. P. Inala, O. Bastani, Z. Tavares, and A. Solar-Lezama. Synthesizing programmatic policies that inductively generalize. In *International Conference on Learning Representations*, 2020.
[38] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *International Conference on Hybrid Systems: Computation and Control*, 2019.
[39] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Case study: Verifying the safety of an autonomous racing car with a neural network controller. In *International Conference on Hybrid Systems: Computation and Control*, 2020.
[40] G. Izatt and R. Tedrake. Generative modeling of environments with scene grammars and variational inference. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6891–6897. IEEE, 2020.
[41] K. Jothimurugan, R. Alur, and O. Bastani. A composable specification language for reinforcement learning tasks. In *Advances in Neural Information Processing Systems*, pages 13041–13051, 2019.
[42] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.
[43] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar. Graph policy gradients for large scale robot control. In *Conference on Robot Learning*, 2020.
[44] S. Kong, S. Gao, W. Chen, and E. Clarke. dreach: δ-reachability analysis for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 200–205. Springer, 2015.
[45] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS*, pages 3675–3683, 2016.
[46] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 2016.
[47] S. Li and O. Bastani. Robust model predictive shielding for safe reinforcement learning with stochastic dynamics. In *International Conference on Robotics and Automation*, pages 7166–7172. IEEE, 2020.
[48] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
[49] N. Lynch, R. Segala, F. Vaandrager, and H. Weinberg. Hybrid I/O automata. In *Hybrid Systems III: Verification and Control*, LNCS 1066, pages 496–510, 1996.
[50] T. M. Moldovan and P. Abbeel. Safe exploration in markov decision processes. In *International Conference on Machine Learning*, 2012.
[51] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.
[52] N. Naik and P. Nuzzo. Robustness contracts for scalable verification of neural network-enabled cyber-physical systems. In *2020 18th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, pages 1–12. IEEE, 2020.
[53] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In *Hybrid Systems*, pages 149–178. Springer, Berlin, Heidelberg, 1992.
[54] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi. Scalable end-to-end autonomous vehicle testing via rare-event simulation. In *Advances in Neural Information Processing Systems*, pages 9827–9838, 2018.
[55] S. Padhi, R. Sharma, and T. Millstein. Data-driven precondition inference with learned features. In *PLDI*, 2016.
[56] C. S. Păsăreanu, D. Gopinath, and H. Yu. Compositional verification for autonomous systems with deep learning components. In *Safe, Autonomous and Intelligent Vehicles*, pages 187–197. Springer, 2019.
[57] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *International Workshop on Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.
[58] R. Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.
[59] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *Proc. of NIPS Workshop Learn. Inference Control Multi-Agent Syst*, 2016.
[60] R. Sharma and A. Aiken. From invariant checking to invariant inference using randomized search. *Formal Methods in System Design*, 48(3):235–256, 2016.
[61] R. Sharma, S. Gupta, B. Hariharan, A. Aiken, and A. V. Nori. Verification as learning geometric concepts. In *International Static Analysis Symposium*, pages 388–411. Springer, 2013.
[62] A. Solar-Lezama and R. Bodik. *Program synthesis by sketching*. Citeseer, 2008.

[63] D. Sun, S. Jha, and C. Fan. Learning certified control using contraction metric. *Conference on Robot Learning*, 2020.
[64] S.-H. Sun, T.-L. Wu, and J. J. Lim. Program guided agent. In *International Conference on Learning Representations*, 2019.
[65] X. Sun, H. Khedr, and Y. Shoukry. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 147–156. ACM, 2019.
[66] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.
[67] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.
[68] H. Tran, F. Cai, D. M. Lopez, P. Musau, T. T. Johnson, and X. Koutsoukos. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems*, 18(5s):105, 2019.
[69] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5045–5054. PMLR, 2018.
[70] K. P. Wabersich and M. N. Zeilinger. Linear model predictive safety certification for learning-based control. In *Conference on Decision and Control (CDC)*, pages 7130–7135. IEEE, 2018.
[71] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pages 6367–6377, 2018.
[72] T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, L. Daniel, D. Boning, and I. Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pages 5273–5282, 2018.
[73] H. Zhu, Z. Xiong, S. Magill, and S. Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *Programming Language Design and Implementation*, pages 686–701, 2019.

A PROOF OF THEOREM 3.9

In this section, we give a proof of safety and liveness assuming VCs 1 and 2. Let π be a compositional controller such that VCs 1 and 2 hold. Let ρ be a non-Zeno trajectory of the automaton generated by π,

$$\rho = (z_0 \xrightarrow{t_0} z_1 \xrightarrow{t_1} \cdots).$$

Let us the denote the cumulative times using $T_i = \sum_{j=0}^{i-1} t_i$. We define $\mathcal{Z}_{\text{pre}} = \cup_{q\in Q}\mathcal{Z}^q_{\text{pre}}$ and $\mathcal{Z}_{\text{post}} = \cup_{q\in Q}\mathcal{Z}^q_{\text{post}}$. Note that VC 1 implies $\mathcal{Z}_{\text{pre}} \subseteq \mathcal{Z}_{\text{safe}}$ and WLOG we can also take $\mathcal{Z}_{\text{post}} = \mathcal{Z}_{\text{post}} \cap \mathcal{Z}_{\text{safe}} \subseteq \mathcal{Z}_{\text{safe}}$. Let $I_{\mathcal{T}}$ denote the set of indices at which a mode transition occurs from a state in $\mathcal{Z}_{\text{post}}$—i.e., $I_{\mathcal{T}} = \{i \mid z_i \rightarrow_{\mathcal{T}} z_{i+1} \text{ and } z_i \in \mathcal{Z}_{\text{post}}\}$. We first show that $I_{\mathcal{T}}$ is infinite, thereby proving liveness.

Lemma A.1. *$I_{\mathcal{T}}$ contains infinitely many indices.*

Proof. We give a proof by contradiction. Suppose $I_{\mathcal{T}}$ is finite. If $I_{\mathcal{T}} = \emptyset$, let $z = z_0 \in \mathcal{Z}_0 \subseteq \mathcal{Z}_{\text{pre}}$. If $I_{\mathcal{T}}$ is nonempty, let $i_{\max}$ be the largest index in $I_{\mathcal{T}}$ and $z = z_{i_{\max}+1}$. We have $z_{i_{\max}} \rightarrow_{\mathcal{T}} z_{i_{\max}+1}$ and from VC 2 we get that $z = z_{i_{\max}+1} \in \mathcal{Z}_{\text{pre}}$.

In either case, we have $z = z_i \in \mathcal{Z}^q_{\text{pre}}$ for some $q \in Q$ and $i \geq 0$. From VC 1 we get that there is a $t \in \mathbb{R}_{\geq 0}$ such that $f(z, \pi, t) \in \mathcal{Z}^q_{\text{post}}$ and for all $t' \in [0, t)$, $f(z, \pi, t') \notin \mathcal{Z}_F$. Since the run ρ is non-Zeno, there is an index $j \geq i$ such that $T_j - T_i \leq t \leq T_{j+1} - T_i$. Since $f(z_i, \pi, t') \notin \mathcal{Z}_F$ if $t' < t$, we get that $z_k \rightarrow_f z_{k+1}$ for all k with $i \leq k \leq j-1$. We now have two cases to consider.

- Case 1: $t = T_j - T_i$. In this case, $z_j = f(z_i, \pi, t) \in \mathcal{Z}^q_{\text{post}}$. Since $z_j \in \mathcal{Z}_F$ we must have $z_j \rightarrow_{\mathcal{T}} z_{j+1}$ in ρ and hence $j \in I_{\mathcal{T}}$.
- Case 2: $t > T_j - T_i$. In this case, $z_j = f(z_i, \pi, T_j - T_i) \notin \mathcal{Z}_F$. Hence we must have $z_j \rightarrow_f z_{j+1}$ in ρ. From the definition of $\rightarrow_f$, it follows that $f(z_i, \pi, t') \notin \mathcal{Z}_F$ for all $t' < T_{j+1} - T_i$. Therefore $t = T_{j+1} - T_i$ and $z_{j+1} = f(z_i, \pi, t) \in \mathcal{Z}^q_{\text{post}}$ and we can conclude that $j + 1 \in I_{\mathcal{T}}$

Therefore, in either case, we reach a contradiction as we showed that there is a $k \in I_{\mathcal{T}}$ with $k \geq i = i_{\max} + 1 > i_{\max}$. □

We now prove safety.

Lemma A.2. *For all $i \geq 0$, $f(z_i, \pi, t) \in \mathcal{Z}_{safe}$ for all $t \in [0, t_i]$.*

Proof. Let $i_1 < i_2 < \cdots$ be the ordered sequence of indices in $I_{\mathcal{T}}$. Let $i_0 = -1$. We show that for all $k \geq 0$ and any $i \geq 0$ with $i_k \leq i < i_{k+1}$, $f(z_i, \pi, t') \in \mathcal{Z}_{\text{safe}}$ for all $t' \in [0, t_i]$.

Let $z = z_{i_k+1} \in \mathcal{Z}_{\text{pre}}$ (VC 2). Then by VC 1, there is a $t \in \mathbb{R}_{\geq 0}$ such that $f(z, \pi, t) \in \mathcal{Z}_{\text{post}}$, $F(z, \pi, t) \subseteq \mathcal{Z}_{\text{safe}}$ and for all $t' \in [0, t)$, $f(z, \pi, t') \notin \mathcal{Z}_F$. Let $j \geq i_k + 1$ be the smallest index after i_k such that $z_j \in \mathcal{Z}_F$. Such a j exists since $z_{i_{k+1}} \in \mathcal{Z}_F$. Let $T = T_j - T_{i_k+1}$. Then we have $z_i \rightarrow_f z_{i+1}$ for all i with $i_k + 1 \leq i < j$. Hence $z_j = f(z, \pi, T)$ and for all $t' < T$, $f(z, \pi, t') \notin \mathcal{Z}_F$. From this we can conclude that $t = T$ and $z_j = f(z, \pi, t) \in \mathcal{Z}_{\text{post}}$. Therefore, j is also the smallest index after i_k such that $j \in I_{\mathcal{T}}$, so $j = i_{k+1}$. Now for any i with $i_k + 1 \leq i < i_{k+1}$, we have $f(z_i, \pi, t') = f(z, \pi, t' + T_i - T_{i_k+1}) \in \mathcal{Z}_{\text{safe}}$ for all $t' \in [0, t_i]$ since $t' + T_i - T_{i_k+1} \leq t$. If $k > 0$, we have $z_{i_k} \in \mathcal{Z}_{\text{post}} \subseteq \mathcal{Z}_{\text{safe}}$ and therefore $f(z_{i_k}, \pi, t') = z_{i_k} \in \mathcal{Z}_{\text{safe}}$ for all $t' \in [0, t_i] = \{0\}$. □

Lemmas A.1 and A.2 together imply the theorem. □

B CHECKING VERIFICATION CONDITIONS

In this section, we describe how we check each of the VCs for a given controller π and a hybrid automaton $\mathcal{A}$.

Verification condition 1. We observe that VC 1 is local to the dynamics of a single mode—i.e., it suffices to verify a safe reachabililty property for the dynamics $\dot{x} = f(z, \pi(z))$, where the mode of z does not change. Thus, we can drop the mode and express these dynamics as $\dot{x} = f^q(x, \bar{\pi}(x))$, where $f^q : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}^n$ and we have defined $\bar{\pi} : \mathcal{X} \rightarrow \mathcal{U}$ by $\bar{\pi}(x) = \pi(h(q, x))$. We let $F^q(x, \bar{\pi}, t) \subseteq \mathcal{X}$ denote the trajectory generated by evolving the system according to this differential equation from state $x \in \mathcal{X}$ for time $t \in \mathbb{R}_{\geq 0}$.

Although verification tools like Verisig can only check safety properties, we can encode the reachability condition as a safety condition by considering a time-limit $T_{\max}$ within which we require the system to reach $\mathcal{X}^q_{\text{post}}$ when started in any state in $\mathcal{X}^q_{\text{pre}}$. Note that the mode predictor has a discrete output; we model each output as a separate mode of the hybrid system.

Verification condition 2. Next, for VC 2, we need to check that for all $q, q' \in Q$, we have $\{x' \mid (q, x) \rightarrow (q', x') \in \mathcal{T},\ x \in \mathcal{X}^q_{\text{post}}\} \subseteq \mathcal{X}^{q'}_{\text{pre}}$. In other words, every state reachable from $x \in \mathcal{X}^q_{\text{post}}$ is contained in $\mathcal{X}^{q'}_{\text{pre}}$ for some $q' \in Q$. This check is problem-specific. For instance, in our F1/10 example, the transitions $(x, q) \rightarrow (x', q') \in \mathcal{T}$ involve an affine change of coordinates—i.e., $x' = A^{q \rightarrow q'} x + b^{q \rightarrow q'}$. Thus, assuming $\mathcal{X}^q_{\text{post}}$ and $\mathcal{X}^{q'}_{\text{pre}}$ are represented by convex polytopes P^q_{post} and $P^{q'}_{\text{pre}}$, respectively, then we can verify VC2 by checking for $q, q' \in Q$, whether $A^{q \rightarrow q'} P^q_{\text{post}} + b^{q \rightarrow q'} \subseteq P^{q'}_{\text{pre}}$. To check this, it suffices to check that each vertex of the polytope $A^{q \rightarrow q'} P_{\text{post}} + b^{q \rightarrow q'}$ is contained in $P^{q'}_{\text{pre}}$, which corresponds to checking feasibility of a system of linear inequalities, which we can do efficiently via linear programming.

C SYSTEM MODELING

LiDAR model. There are 21 LiDAR rays giving us a 21-dimensional observation $o \in \mathbb{R}^{21}$. The rays range from −115 to 115 degrees relative to the car's orientation—i.e., there are rays at $-115, -103.5, \ldots, 115$ degrees relative to the car's orientation. Each LiDAR ray can be modeled as a function of the car's state relative to the current track segment. Figure 8 illustrates the scenario of a ray reaching the right wall in a straight segment. The specific equation for such a ray is $o_i = h(\vec{x})_i = \frac{d_r}{\cos(\vartheta - \alpha_i)}$, where d_r is the distance to the right wall, and α_i is the relative angle (in

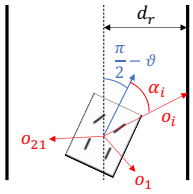


Fig. 8. LiDAR observation for a straight segment.

radians) of ray i with respect to the car's orientation ϑ. Rays for other walls and segments can be modeled similarly, depending on which walls are in range.

D CONTROLLER TRAINING

Reward function. The reward function we used for a right turn (the left-turn case is symmetric) is

$$r(\vec{x},\vec{u}) = \begin{cases} g_s - g_i \cdot \phi^2 - g_m \cdot d(\vec{x}) & \text{if before turn} \\ g_s - g_h \cdot h(\vec{x}) & \text{if during turn} \\ g_s + g_f \cdot \delta(\vec{x},\vec{u}) - g_m \cdot d(\vec{x}) & \text{if after turn} \\ g_c & \text{if crash,} \end{cases}$$

where $g_s = 5$ is a reward for each safe step, g_i is a loss penalizing high steering angle ϕ, g_m is a penalty on the car's distance $d(\vec{x})$ from the middle of the lane, $g_h = 3$ is a loss penalizing the difference $h(\vec{x})$ between the car's orientation and the turn angle (either 90 or 120 degrees), $g_f = 10$ is a reward for the distance $\delta(\vec{x},\vec{u})$ covered on the current step after the turn in the new segment direction, and $g_c = 100$ is a penalty for crashing. The values g_i and g_m depend on the mode and are chosen as follows: (i) $g_i = -0.05, g_m = -2$ for state-feedback; (ii) $g_i = 0, g_m = -3$ for a 90-degree turn and LiDAR-feedback; and (iii) $g_i = -0.05, g_m = -0.5$ for a 120-degree turn and LiDAR-feedback.

Each controller has two hidden layers, with 32 neurons per layer, each with a tanh activation. We train each controller for 2e5 number of simulation steps, using the TD3 algorithm [28]. The training code takes roughly 15 minutes to run.

E MODE PREDICTOR TRAINING

As described in Section 5, the mode predictor is trained using standard supervised training, with the data labeling illustrated in Figure 5. The mode predictor consists of a total of six NNs: a *new mode predictor* and five *exit detectors*, one for each track segment. All NNs have two hidden layers, with 32 neurons per layer; the hidden layers have tanh activations, whereas the output layer is linear. We train the *new mode predictor* for 30 epochs, whereas each *exit detector* is trained for 15 epochs, with a batch size of 32 and learning rate of 0.001. Training is fairly fast and takes a few minutes per NN.

F VERIFICATION OF STRAIGHT SEGMENTS

In this section, we describe our inductive approach for verifying straight segments of all lengths $\ell \geq \ell_{\min}$ where $\ell_{\min}$ is a bound on the shortest straight segment. We first choose $\mathcal{Z}^q_{\text{post}}$ to be of the form $\mathcal{Z}^q_{\text{post}} = \{(q,(x,y,\vartheta,v)) \mid (x,\vartheta,v) \in \tilde{\mathcal{X}}_{\text{post}}, y \in [y^\ell_{\text{inf}}, y^\ell_{\text{sup}}]\}$, where $\tilde{\mathcal{X}}_{\text{post}}$[6] $\subseteq \mathbb{R}^3$ is a set of possible values for x, ϑ and v. Next, we identify a *recurrent* set $\tilde{\mathcal{X}}_{\text{rec}} \subseteq \tilde{\mathcal{X}}_{\text{post}}$ and define $\mathcal{Z}^q_{\text{rec}} = \{(q,(x,y,\vartheta,v)) \mid (x,\vartheta,v) \in \tilde{\mathcal{X}}_{\text{rec}}\}$. Then we split VC 1 into two VCs as follows.

[6]The superscript q is omitted since it is clear from context.

Mode	# instances	# branches	Verification time (seconds)	
			Composed system	NN only
90-degree right	24	1.50	258	40
120-degree right	24	1.75	285	38
straight initial	30	1.00	71	17
straight inductive	1	1.00	11	3

Table 2. Verification results for state-feedback system. Verification times and number of branches are averaged across all the instances for that mode. "Composed system" is the (average) time for fully verifying a single instance, and "NN only" is the time spent propagating reachable sets through the NNs during closed loop verification. All times are in seconds.

Definition F.1 (VC 1.1)**.** For any $z \in \mathcal{Z}^q_{\text{pre}}$, there is a $t \in [0, \ell_{\min}/v_{\max}]$ such that $f(z, \pi, t) \in \mathcal{Z}^q_{\text{rec}}$ and $F(z, \pi, t) \subseteq \mathcal{Z}_{\text{safe}}$.

This VC says that the car safely reaches $\mathcal{Z}^q_{\text{rec}}$ from any state in $\mathcal{Z}^q_{\text{pre}}$ within time $\ell_{\min}/v_{\max}$ where $v_{\max}$ is the maximum speed of the car. The time bound guarantees that the car does not reach a state in $\mathcal{Z}_F$ before reaching $\mathcal{Z}^q_{\text{rec}}$. For the next VC, we define the progress region with respect to a state $z = (q, (x, y, \vartheta, v)) \in \mathcal{Z}^q_{\text{rec}}$ as $\mathcal{Z}^q_{>z} = \{(q, (x', y', \vartheta', v')) \mid (x', \vartheta', v') \in \tilde{\mathcal{X}}_{\text{rec}},\ y' \geq y + \varepsilon\}$ where $\varepsilon \in \mathbb{R}_{>0}$ is a lower bound on the increase in y-position[7].

Definition F.2 (VC 1.2)**.** For any $z \in \mathcal{Z}^q_{\text{rec}}$, there is a $t \in \mathbb{R}_{>0}$ such that $f(z, \pi, t) \in \mathcal{Z}^q_{>z}$ and $F(z, \pi, t) \subseteq \tilde{\mathcal{Z}}^q_{\text{post}}$ where $\tilde{\mathcal{Z}}^q_{\text{post}} = \{(q, (x, y, \vartheta, v)) \mid (x, \vartheta, v) \in \tilde{\mathcal{X}}_{\text{post}}\}$.

This VC says that the car stays in $\tilde{\mathcal{Z}}^q_{\text{post}}$ while making progress in the y-direction. Since the observation in a straight segment is independent of the y position, it is enough to verify VC 1.2 for a fixed starting value of y. Furthermore, the progress criterion ensures that the car will safely reach the post-region of any straight segment of length $\ell \geq \ell_{\min}$ when started in its pre-region.

G VERIFICATION RESULTS FOR STATE-FEEDBACK SYSTEM

In this section, we provide verification results for the F1/10 system with a state-feedback controller. In this setting, it is sufficient to verify the 90-degree right, 120-degree right and the straight segments since the left turns are symmetric. This is not the case with the LiDAR-feedback system since the mode predictor is not symmetric.

Similar to the LiDAR-feedback system, pre-region is the same for all modes, given by $x \in [0.6, 0.9]$, $y \in [0, 0.24]$, $\vartheta \in [-0.005, 0.005]$ and $v \in [2.4, 2.4]$; the post-regions are similar. To reduce the overapproximation error introduced during verification, we split the initial set by increments of 0.05 along the x-dimension and 0.06 along the y-dimension, thus ending up with 24 verification instances per turn. For the initial straight segment, we split the initial set by increments of 0.01 along the x-dimension and there is no uncertainty in the y-dimension.

Finally, in order to apply inductive reasoning in straights, we clip the y-distances to a maximum value of 5 (before feeding the state to the NN controller) which makes the observations independent of y in the straights. The verification results are summarized in Table 2.

[7]Here y and y' denote y-distances from the start of the straight segment.