Can You Trust Your Autonomous Car? Interpretable and Verifiably Safe Reinforcement Learning

Lukas M. Schmidt[1], Georgios Kontes[1], Axel Plinge[1] and Christopher Mutschler[1]

***Abstract*— Safe and efficient behavior are the key guiding principles for autonomous vehicles. Manually designed rule-based systems need to act very conservatively to ensure a safe operation. This limits their applicability to real-world systems. On the other hand, more advanced behaviors, i.e., policies, learned through means of reinforcement learning (RL) suffer from non-interpretability as they are usually expressed by deep neural networks that are hard to explain. Even worse, there are no formal safety guarantees for their operation.**

In this paper we introduce a novel pipeline that builds on recent advances in imitation learning and that can generate safe and efficient behavior policies. We combine a reinforcement learning step that solves for safe behavior through the introduction of safety distances with a subsequent innovative safe extraction of decision tree policies. The resulting decision tree is not only easy to interpret, it is also safer than the neural network policy trained for safety. Additionally, we formally prove the safety of trained RL agents for linearized system dynamics, showing that the learned and extracted policy successfully avoids all catastrophic events.

I. Introduction

Automated and autonomous driving poses one of the biggest challenges to the development of artificial intelligence (AI), as it is technically demanding to solve the tasks involved in order to make a car act autonomously in real world situations. However, unless autonomous systems become truly safe and dependable, they cannot be deployed in a real world setup. Operating autonomous vehicles not only efficiently but also safely and reliably is even more challenging. There are many unique tasks that need to be solved reliably for safe and efficient performance. Among them, behavioral planning is one of the hardest challenges [1].

Overtaking maneuvers under varying road types and traffic conditions, such as the one presented in Fig. 1, are typical scenarios in which autonomous vehicles operate. Here, the task is to overtake cars on a highway. The vehicle should optimize for both speed and efficiency and needs to execute the maneuver safely. While strictly adhering to regulatory safety requirements is crucial, trading off conservative, careful behavior against performance has its limits. Many situations, such as cutting into traffic on a highway, require maneuvers that involve an acceptable amount of risk. Indeed, participation in traffic already introduces a performance versus risk trade-off. Extremely conservative behavior is often less efficient and performant, and therefore unwanted.

Hence, an autonomous system that is deployed in a real-world vehicle has to meet exceptionally high safety standards. This includes a verification of safety, either using analytical formal methods [2] or by constraining the systems operation under a set of reasonable assumptions [3]. The final system should take decisions that minimize risk.

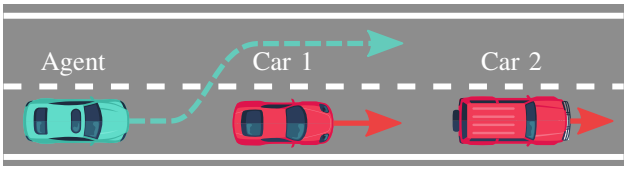

Fig. 1. Experimental scenario. The agent follows two cars on the right lane of a two-way highway. Car 1 is faster than Car 2, and will overtake. The agent has to decide when and how to overtake.

Static rules [5] or multi-criteria decision making [6] have been proposed for taking such risk-sensitive decisions. However, manually designed rule systems often suffer from being too conservative, failing to take small risks where sensible. Recently, reinforcement learning (RL) has emerged as a viable alternative [7], [8]. In RL, an agent (e.g., an autonomous vehicle) interacts with an environment (in a simulation or the real world) through actions (e.g., driving). The agent receives a reward (or penalty) for its behavior. RL aims to optimize for behavior that maximizes the reward. While state-of-the-art Deep RL approaches generally work very well, the control strategies (called *policies*) generated are black box models and thus difficult to interpret analytically. This clearly contradicts with the expressed need for transparency, fairness and explainability in AI systems from legislative bodies [9], research organizations [10], and the general public [11], [12].

While most work on neural network interpretability [13], [14] fails to explain policy networks, previous work by Bastani et al. [4] uses imitation learning on the policy to extract decision trees that are both interpretable and easy to comprehend through manual analysis. Moreover, the decision trees can be formulated as sets of logical clauses of the input because the policy is piecewise linear. Through formulating system clauses the logical rules can be used to formally prove correctness [4].

This paper provides a pipeline that builds on this technique to create policies that are both safe and interpretable. The pipeline trains a non-interpretable RL agent for safe behavior, modifying existing reward structures and training techniques. It then extracts a set of rules approximating the policy in the form of a decision tree. We show that this extracted policy is as performant and safe as the deep neural network (DNN) agent, and at the same time easily interpretable. The last component of our pipeline formally verifies (or rejects)

[1] Precise Positioning and Analytics Department, Fraunhofer Institute for Integrated Circuits IIS, Nordostpark 84, 90411, Nuremberg, Germany. {firstname}.{lastname}@iis.fraunhofer.de

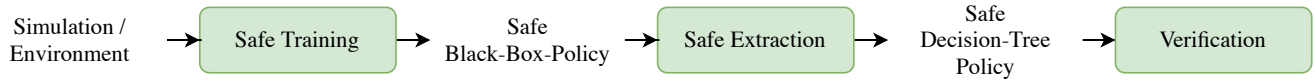

Fig. 2. Our pipeline. We first train a policy that solves the environment. We modify the safe training step by adding safety distances to the reward function (see Section III-A). Using this safe policy as a teacher, we extract a decision tree using a modified version of VIPER [4]. Section III-B details this step. We then convert the final safe decision tree policy into logical constraints and verify that the policy operates safely (see Section III-C),

the safety of our trained agents for linearized system dynamics. In summary, our pipeline allows us to benefit from recent advances in (Deep) RL, while adhering to strict safety and interpretability constraints. Although we focus on an overtaking maneuver as a running example, our method generalizes to several risk-sensitive autonomous driving situations, such as roundabout entry and exit and navigation at intersections. To the best of our knowledge, this presents the first application of VIPER [4] and its verification ideas to an autonomous driving scenario.

The remainder of this paper is structured as follows. Sec. II introduces related work. Sec. III presents the details of our pipeline. Sec. IV describes the implementation of our verification pipeline and experiments. Sec. V present our experimental results that show how our pipeline significantly improves the safety of the behavior. Sec. VI concludes.

II. Related Work

In autonomous driving, Deep RL has been applied to different problems and scenarios. These include the double merging problem [15], highway driving scenarios [16], [17], [18], [19], and tactical decision making for roundabouts [1]. RL has also been used for overtaking maneuvers in extreme driving scenarios such as racing [20] or high-speed obstacle avoidance [21]. Some of these approaches include features that aim to make the resulting policy more performant and safer, e.g., by explicitly learning model uncertainty [17]. However, in their vanilla variants such approaches neither provide interpretability nor formal safety guarantees.

To yield safe behavior, research in the transportation domain adds explicit safety-related components to RL algorithms. Ye et al. [18] use proximal policy optimization (PPO) [22] and add a penalty term to the reward function that ensures minimum safety distances. However, while the resulting policies are empirically safe, they still remain non-interpretable. Recent work [19], [23] augments the generated policies with components that can reject actions if they are deemed unsafe, building on previous work in general RL research [24], [25]. These approaches are based on formal verification for safety distances [19] or future lane occupancy [23] and reject behavior by the black box agent that cannot be verified as safe. A similar approach by Chen et al. [26] adds a safety component that evaluates the risk of actions based on a predefined regret-based human lane change model. However, all the aforementioned approaches require online computations and verifications, and inherently need to make conservative approximations in their reasoning. This severely restricts the actions they can take, reducing both performance and efficiency.

Safe RL is a subfield of RL research that deals with safety-relevant tasks. Usually, the problem is modelled as a constrained Markov decision process (CMDP) [27] and the goal is to maximize the expected reward subject to safety constraints. Lagrangian methods [28] transform the problem into a regular Markov decision process (MDP) by adding constraint violations to the reward signal with an adaptive or constant multiplier. Constrained Policy Optimization (CPO) [29] optimizes policies by carefully optimizing the policy within a trust region that is deemed safe. However, while both methods can optimize for safe behavior, they build on non-interpretable neural networks.

III. Safe policy generation pipeline

An overview of our pipeline for the training and extraction of safe interpretable policies is illustrated in Fig. 2. First, the *safe training* step trains an RL agent using PPO [22] towards a safe policy (see Sec. III-A). We formulate the task as a CMDP and use reward modifications for safety distances (similar to Ye et al. [18]) to solve for safe behavior. Second, the *safe extraction* step extracts an interpretable policy with an imitation learning algorithm adapted to CMDPs (see Sec. III-B; our algorithm is based on VIPER [4]). Third, the *verification* step formally proves the safety of the agent policy (see Sec. III-C). Similar to the ideas of Bastani et al. [4], we reformulate both the environment and the trained decision trees into sets of mathematical constraints. Then, we show that, when these constraints hold, a crash is formally impossible. This verifies the safety of the trained policies for linearized dynamics. We use a linearized lane change scenario as a running example.

A. Safe Training

In a Markov decision process (MDP) an agent interacts with an environment by taking actions based on the current state of the environment. More formally, a MDP is a tuple $(S,\ A,\ P_a,\ R_a)$, where S is a state space, A is an action space, $P_a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the transition probability to move from state s_t to state s' with an action a_t, and $R_a(s, s')$ is a reward the agent receives after transitioning from s to s' with action a at time step t. A policy π maps states to actions, i.e., it decides which action to take in each state. Often, MDPs are *episodic*. In episodic MDPs, the process ends (or is reset) according to some conditions, e.g., after a certain time or after the goal is reached. A *trajectory* is a set of states, actions, and rewards collected by an agent in one such episode. The goal of RL is to find a policy that maximizes the expected return of the agent.

To include strict safety requirements, we formulate the problem as a constrained Markov decision process (CMDP). In a CMDP the agent learns to optimize its reward subject to a set of constraints (e.g., safety). Our pipeline adds a reward component for safety distances to solve for safe behavior. This component r_s is calculated as

$$r_s = 1 - \min\left\{\frac{\min_v \{d_{a,v}\}}{d_\mathrm{safety}}, 1\right\} \tag{1}$$

where $d_{a,v}$ is the distance between the agent and the vehicle with index v and d_safety is the desired safety distance. r_s is equal to 1 if the agent has at least a distance of d_safety to all other vehicles. Otherwise, this reward component scales down to zero.

These safety distances act in two ways: First, they add a direct and dense reward signal that allows the agent to gather experience on situations with high risk, i.e., driving close to other vehicles, without experiencing catastrophic events directly. This improves the training behavior and the safety of the policy. Second, it explicitly models the codified (and intuitive) necessity of keeping a safe distance: Without the reward modification, the agent cannot distinguish between consistent *close misses of catastrophe* and safe behavior. While it might be optimal to execute a maneuver close to other vehicles (e.g. when training in a simulation) these same behaviors can lead to catastrophic outcomes in a less predictable real world scenario. Similar to how we mandate safety distances on the highway to human drivers, we should also mandate them for autonomous vehicles.

B. Safe Extraction

VIPER [4] is an imitation learning algorithm that extracts *student* decision trees from a *teacher* policy. This is done by creating a dataset $\mathcal{D}$ of transitions, each with an observation and action probabilities from the teacher. $\mathcal{D}$, weighted by the strength of the preference of the teacher for the action taken, is then used to train decision trees (the students) using the CART algorithm [30]. With each student, an additional set of transitions is gathered by running the student policy in the environment.

The vanilla VIPER algorithm is not adapted to CMDPs. This means that even when extracting, i.e., learning, from a safe teacher policy, VIPER occasionally produces unsafe trees. In preliminary experiments, we identified three major causes for this, and modified VIPER accordingly.

First, VIPER's reweighting step does not consider constraints. Samples where the teacher would have taken a critical decision to avert catastrophic outcomes could be weighted lower than uncritical, performance-optimizing decisions. Hence, we label steps in a trajectory as *critical* if (a) the trajectory ended in a constraint violation, and (b) the student disagreed with the teacher on the action to take. To make sure that subsequent students learn from this mistake, we introduce a secondary dataset of transitions for these critical samples $\mathcal{K}$. This second dataset is used to generate every tree, and the samples in it are assigned higher weights than the uncritical ones. (We used a relative weight of 5.) Over the VIPER training procedure, the algorithm thus gathers these critical transitions, and learns to precisely imitate the teacher policy at the crucial points. This modification relies on the teacher to provide a safe action.

Second, VIPER does not include a mechanism to exclude unsafe students. As a simple counter-measure we remove students that occasionally crashed while gathering experience (line 14). We also sort students by safety and reward in ascending order (instead of only by the reward) when selecting the final student to return. This makes sure that the final student returned is (empirically) as safe as possible.

Third, during tree extraction VIPER minimizes the difference between the actions taken by the decision tree policy and the neural network policy. This often produces wide and deep trees, which are difficult to interpret and to formally prove safe. Hence, we constrain the depth of the trees. This might also mitigate overfitting of rule sets. We found a depth of five to be a sweet spot between model accuracy and interpretability in our scenario.

Algorithm 1 shows the pseudo-code of our adapted VIPER algorithm SafeVIPER. We first initialize a dataset of transitions $\mathcal{D}$, a dataset for critical transitions $\mathcal{K}$, the initial policy for experience aggregation $\hat{\pi}_0$ and the safe student set Π_s (lines 2-5). Then, we repeatedly gather experience from the current policy $\hat{\pi}_i$ (line 7) and add this experience to our dataset $\mathcal{D}$ (line 8). We extract critical trajectories, i.e., those trajectories where $\hat{\pi}_i$ led to a constraint violation, and add these to $\mathcal{K}$ (lines 9 and 10). We then train a new student

Algorithm 1 Safe extraction algorithm based on VIPER [4]. We introduce a few modifications: We make sure that each student learns from critical samples (lines marked with **C**), and delete students that evaluate as unsafe (lines marked with **S**). Additionally, we penalize constraint violations during the final tree evaluations (marked with **V**). Unmarked lines are unchanged from the original algorithm.

1: **procedure** SafeVIPER($(S, A, P, R), \pi^*, Q^*, M, N$)
2: Initialize dataset $\mathcal{D} \leftarrow \varnothing$
3: Initialize constraint dataset $\mathcal{K} \leftarrow \varnothing$ ▷ **C**
4: Initialize policy $\hat{\pi}_0 \leftarrow \pi^*$
5: Initialize safe student set $\Pi_s \leftarrow \varnothing$ ▷ **S**
6: **for** $i \leftarrow 1$ **to** N **do**
7: Sample M trajectories $\mathcal{D}_i \leftarrow \{s, \pi^*(s) \sim d^{\hat{\pi}_{i-1}}\}$
8: Aggregate dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
9: Find critical trajectories $\mathcal{K}_i \leftarrow \mathrm{Critical}(\mathcal{D}_i)$ ▷ **C**
10: Aggregate constraint dataset $\mathcal{K} \leftarrow \mathcal{K} \cup \mathcal{K}_i$ ▷ **C**
11: Resample dataset $\mathcal{D}' \leftarrow \mathrm{Resample}(\mathcal{D})$
12: Add constraint dataset $\mathcal{D}' \leftarrow \mathcal{D}' \cup \mathcal{K}$ ▷ **C**
13: Train decision tree $\hat{\pi}_i \leftarrow \mathrm{TrainDecisionTree}(\mathcal{D}')$
14: **if** CheckSafety($\hat{\pi}_i$) **then** ▷ **S**
15: Update safe students $\Pi_s \leftarrow \Pi_s \cup \{\hat{\pi}_i\}$ ▷ **S**
16: **end if**
17: **end for**
18: **return** Best *safe* policy $\hat{\pi} \in \Pi_s$ ▷ **V**
19: **end procedure**

policy with a dataset containing $\mathcal{K}$ and a subset of $\mathcal{D}$ (lines 11-13). The trained student is added to Π_s if it does not violate constraints in an evaluation test (lines 14 and 15). Eventually, after we trained N students, we evaluate all policies in Π_s. We return the policy that achieves the highest reward, penalizing constraint violations with a high multiplier.

C. Verification

For the extracted decision tree policies, we formally prove that they operate in a safe manner that avoids catastrophic situations in our scenario. This verification is exhaustive by design, more so than any empirical evaluation can feasibly be.

Our verification approach is based on work by Bastani et al. [4]. The general idea is to translate the system dynamics, the agent policy, and the definition of a catastrophic episode into sets of constraints. Let $\mathcal{S}$ be the set of clauses that describe our system, $\mathcal{P}$ the set of clauses that describe the agent policy, and $\mathcal{C}$ the set of clauses that describe catastrophic situations, i.e., crashes. Formally, we prove that the following holds:

$$\mathcal{S} \wedge \mathcal{P} \implies \neg\mathcal{C} \tag{2}$$

In other words, if both the system dynamics and the policy hold, there is no solution that satisfies the crash conditions. As a consequence, such a system is deemed *safe*.

To prove (2), we use the satisfiability modulo theories (SMT) solver Z3 [31] to find solutions to the negation of Equation 2:

$$\neg(\mathcal{S} \wedge \mathcal{P} \implies \neg\mathcal{C}) \equiv \cdots \equiv \mathcal{S} \wedge \mathcal{P} \wedge \mathcal{C} \tag{3}$$

If there are no assignments that satisfy (3), (2) must hold, and thus the system and the agent together are safe.

For completeness, we also prove the satisfiability of the following sets of constraints:

$$\mathcal{S} \wedge \mathcal{P} \tag{4}$$

$$\mathcal{S} \wedge \mathcal{C} \tag{5}$$

Intuitively, if (4) were unsatisifiable, the agent policy contradicts the system dynamics. If (5) were unsatisfiable, the system was already safe without our agent policy. While desirable, this is usually not the case.

IV. Experimental Setup

A direct one-to-one implementation of the dynamics, the observations, and the vehicle behavior for the highway environment into clauses that we can use for verification is, while theoretically possible, a prohibitively large effort. Moreover, the non-linear dynamics would slow down verification or prohibit it entirely, due to limits inherent to SMT solvers.

Hence, we split our experiments: We evaluate our pipeline numerically using a state-of-the-art highway simulation, and verify the safety of agents in a linearized version of it. Sec. IV-A describes the implementation of the environment used for the experiments that do not involve verification and Sec. IV-B explains the linearized environment and how we convert the simulation into logical clauses for verification.

A. Environment implementation

We evaluated our pipeline on a specific lane change scenario based on the highway environment [32]. For our scenario (see Fig. 1) we place three cars on the right lane of a highway. The foremost car (Car 2) drives slower than the middle car (Car 1). The agent's car is fastest but behind the two. The longitudinal decisions of non-vehicle cars are made by standard intelligent driver model [33] rules, while lane change decisions are made according to the *minimizing overall braking induced by lane change* (MOBIL) model [5]. This is a complex situation: The agent car cannot simply overtake both vehicles as the second vehicle might decide to overtake on its own. Thus, the agent has to anticipate the likely reaction of the second vehicle, and either overtake both cars decisively or wait until the second car has passed first. Each episode starts with a random initialization of the relative distances and velocities of the vehicle, and is simulated for 40 time steps or ends upon a crash.

State space. The state observed by the agent car is a list of the closest vehicles. The list is sorted by their distance to the agent. For each vehicle, this list includes its lane identifier, its relative lane distance to the agent car, and its relative speed. The lane identifier is 0 on the right lane, and 1 on the left lane. The agent car also observes its own lane identifier and speed. Among different observation spaces we found this observation space to lead to safe policies and interpretable trees.

Action space. We use a set of semantic (and thereby explainable) actions. In each time step, the agent can decide to stay idle (keeping its lane and speed), to accelerate, to brake, or to change its lane to the left or to the right. These actions set a desired state, i.e., a desired velocity v_d and a desired lane l_d, for low-level controllers implemented in the environment, which execute the desired action and track the specified speed and lane. Actions are computed by the policy at every time step. It may take multiple time steps for the vehicle to reach the desired state via its lateral and longitudinal controllers. If an action changes the desired state in the meantime, the controllers adapt and act towards the new desired state. This also allows to abort lane change maneuvers. The verification environment uses the same set of actions, but differs in the implementation of the low-level controllers, as the system dynamics (and the controllers) are linearized.

Reward structure. The reward is given to the agent by

$$r = r_v \cdot w_v + r_s \cdot w_s - r_c, \tag{6}$$

where r_v is a reward component for high speed with weight w_v, r_s is a reward component for safety distances with weight r_s, and r_c is a reward penalty for crashes. The reward r is then linearly mapped to a the range of $[0, 1]$. The high speed r_v reward maps the agent velocity to a range of 0 and 1. We do this by specifying a minimum rewarded velocity $v_{r,\min}$ and $v_{r,\max}$, mapping the agent velocity between these values within $[0, 1]$, and clamping the resulting value to $[0, 1]$. Velocities below $v_{r,\min}$ are not rewarded, while those

TABLE I
HYPERPARAMETERS USED IN THE SAFE TRAINING AND EXTRACTION STEPS

Name	Value
PPO / Safe Training	
γ	0.98
learning rate	0.00037
steps	32
minibatches	1
entropy coefficient	0.0
optimization epochs	20
λ	0.8
clip range	0.3
policy network size	2 layers with 256 nodes
activation function	tanh
VIPER / SafeVIPER	
iterations	80
batch rollouts	100
maximum samples	2000000
test rollouts	1000

above $v_{r,\max}$ are rewarded equally to $v_{r,\max}$. This allows the agent to brake as necessary while it does not reward excessive speeding. In our experiments, we set $v_{r,\min} = 20\,\frac{m}{s}$ and $v_{r,\max} = 30\,\frac{m}{s}$. The safety distance component r_s is given by Equation 1. r_c is 1 if the vehicle crashed in the last time step, and 0 otherwise. We end the episode after a crash, which lead to reduced rewards for the agent. Note that for our verification environment, we verify with a slightly smaller crash distance ($5\,m$) than used for training ($7.5\,m$).

We evaluate two different reward settings: A *baseline* configuration without a reward for safety distances and a *safety* configuration that includes this reward. For the baseline, we use $w_v = 0.4$, and $w_s = 0$. For the experiments with the safety distance, we use $w_v = 0.1$ and $w_s = 1$. This reward is used both in our original environment, and in the linearized version. We use a discount factor of $\gamma = 0.98$ for learning.

Our pipeline uses proximal policy optimization (PPO) as the training algorithm [22], based on the implementation in stable-baselines [34]. We experimented with a variety of hyperparameters, and found the set summarized in Table I to work well. We train for two million time steps.

To evaluate the trained policies, we report the final longitudinal position of the agent car as a good score for the agent performance and the probability of crashes in 20 000 time steps of simulation. Additionally, we evaluate the policies in a *randomized* test environment. In this environment the non-vehicle cars randomly change their speed every five time steps. This makes crashes more likely and a performant, safe operation harder for the agent.

B. Translating a highway simulation into logical constraints

We verify the safety of our agents by training and verifying them in a linearized version of the environment. This environment uses the same state and action spaces and reward as the full evaluation environment described earlier, but uses linearizations of both system mechanics and non-agent policies to allow for verification.

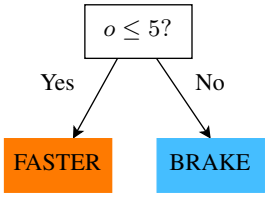

Fig. 3. A single node in a decision tree. This node splits on the observation variable o. If $o \leq 5$, the agent vehicle will accelerate. If $o > 5$, it will break.

In the following we describe how we express the system dynamics $\mathcal{S}$, the policy constraints $\mathcal{P}$, and the catastrophic constraints $\mathcal{C}$. Note that we describe each constraint for a single time step only and we hence omit this time step in the notation for clarity. The complete constraint for $\mathcal{S}$ and $\mathcal{P}$ is a conjunction for the constraints per time step, as these constraints need to always hold. For $\mathcal{C}$, the complete constraint is a disjunction, as a single crash is sufficient to find a catastrophic outcome. We verify the behavior over 40 time steps (i.e., as long as the simulation) to find an exhaustive proof of safety in our scenario.

System dynamics $\mathcal{S}$. The highway environment uses a non-linear bicycle-style model [35] for the car kinematics. To linearize the kinematics we first approximated the longitudinal dynamics (i.e. along the lane) using basic Newtonian equations for point masses. We followed different ideas for the lateral kinematics (i.e., movement between the lanes y) in the simulation. Initially, we did not model y directly, and instead assumed that lanes can be changed within a single time step. While this allowed for a comparatively fast verification, it also removes many interesting aspects of the dynamics in the lane change scenario (e.g., the agent cannot abort lane changes). Unfortunately, a much more realistic linearized bicycle-style model of the kinematics, using a Stanley controller [36] for lateral control, led to excessive verification times.

Thus, we decided to use a constant step size for the lateral position during lane changes. This allows complex behaviors (like aborting a lane change), while keeping verification times acceptable. The non-agent behavior in the verification environment is given via manual implementations, but based on the same observations that the agent has. Note that these form part of $\mathcal{S}$, not the policy constraints $\mathcal{P}$, as the latter ones only relate to the current agent vehicle that is to be verified.

Policy constraints $\mathcal{P}$. The policy constraints $\mathcal{P}$ can be created by translating the decision trees generated by VIPER into nested implications. For instance, assume a node that chooses an action a based on a decision variable o, as seen in Fig. 3. We express this node as the following constraint:

$$\begin{aligned} o \leq 5 &\implies (a = \texttt{accelerate}) \\ \wedge\, o > 5 &\implies (a = \texttt{brake}) \end{aligned} \tag{7}$$

To create constraints from a tree instead of a single node we apply this transformation recursively.

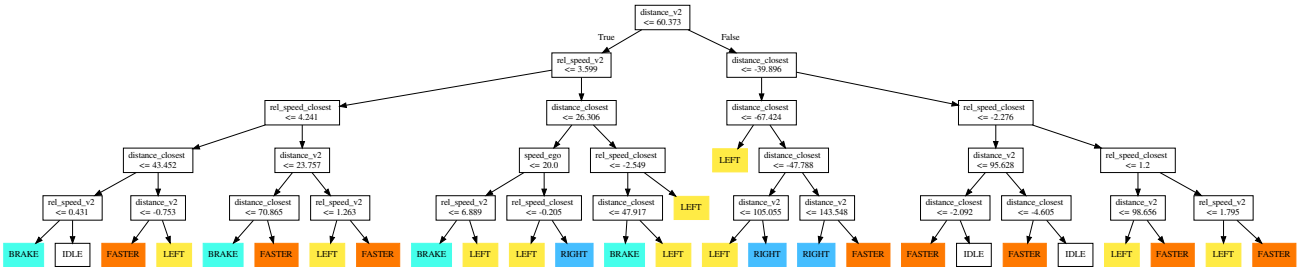

Fig. 4. Policy tree extracted from a PPO policy trained with the *safety* training modifications. Each node represents at the decision. To compute the output of the decision tree, start at the top node, and evaluate each node, stepping left or right, until a leaf node is reached. Leaf nodes are colored according to the action: **Change lane to the right**, **change lane to the left**, **accelerate**, **brake**, and **idle** (no dedicated action to take).

To compute the policy actions using decision trees trained on the simulation environment, the observations need to be expressed as constraints as well. This means that concepts that can be efficiently expressed algorithmically (like, e.g., finding a list of the closest vehicles) need to be translated into constraints that lead to equivalent outcomes.

Catastrophic constraints $\mathcal{C}$. The crash constraints $\mathcal{C}$ are comparatively easy to express. The catastrophic clauses encode the definition of a catastrophic situation. For two vehicles a and b, we encode a catastrophic situation as

$$\begin{aligned}\text{CRASH}(a,b) \equiv |x_a - x_b| < d_{x,\text{crash}} \\ \wedge\, |y_a - y_b| < d_{y,\text{crash}},\end{aligned} \tag{8}$$

where x_v is the longitudinal position of the vehicle v, y_v is its lateral position, and $d_{\{x,y\},\text{crash}}$ are crash distances that define a longitudinal and lateral crash, respectively.

V. Experimental Results

In the following, we give detailed results of our evaluation. We discuss the effect of the safe training step (Sec. V-A), investigate performance and safety of the extracted decision trees for both our pipeline and a baseline configuration (Sec V-B), and show the interpretability of the extracted tree by example (Sec. V-C). Finally, we discuss the verification of the extracted decision tree policy (Sec. V-D).

A. Safe training

We show performance and the probability of constraint violations of our training method and the baseline in Table II (top rows, respectively). Compared to the unmodified *baseline*, our training method (*safety*) achieves far safer behavior (i.e., number of crashes) at a trade-off against performance of only less than 10%. This holds not only for the *constant* but also for the *randomized* evaluation scenarios, where the effect is even more significant. This shows that our training method significantly improves robustness to changes in the environment. Individual replications of our method did achieve a score of up to 1358 in the *constant* environment (without any constraint violations) and managed to achieve perfect safety in the *randomized* evaluation.

B. Safe extraction

Our evaluations, summarized in the second row of Table II, convincingly show that our full pipeline can generate almost perfectly safe and at the same time performant trees. Trees trained with safe training and extracted with SafeVIPER showed zero constraint violations in the *standard* evaluation, and even generalize robustly to the harder *randomized* simulation. This compares extremely well to our baseline: Trees extracted using vanilla VIPER from *baseline* teachers showed poor results, with a crash probability of 12.3% even in the relatively simple *constant* evaluation.

For both teacher policy types, SafeVIPER generated decision trees that were on average as safe or safer than their original teacher policies. Interestingly, this comes at no performance cost: the trees generated with SafeVIPER show an average performance that is within 1% of the average performance of the teacher policy. Vanilla VIPER generated trees that showed unsafe behavior and often worse performance.

The results also emphasize the importance of the safe training step: A safe teacher policy (trained with the *safety*

TABLE II
Evaluation Results

Policy	*constant* Score	*constant* Crashes	*randomized* Score	*randomized* Crashes
RL policies				
baseline	1385± 2	0.4±0.5%	1337± 45	6.2± 5.2%
safety	**1270**±62	**0.1**±0.1%	**1234**± 74	**0.3**± 0.4%
Trees extracted using VIPER				
baseline	1268±79	12.3±9.4%	1219±123	18.3±13.4%
safety	**1267**±35	**0.1**±0.2%	**1229**± 36	**4.2**± 9.0%
Trees extracted using SafeVIPER[1]				
baseline	1373± 5	0.5±0.3%	1352± 15	3.3± 1.3%
safety	**1259**±37	**0.0**±0.0%	**1230**± 39	**0.2**± 0.4%

Evaluation performance and crash frequencies for the *baseline* and *safety* training methods. For performance, higher is better, for crashes, lower is better. Results show mean and standard deviation across 5 replications. All policies are evaluated on the standard (*constant*) environment, and on a noisy alternative where the other cars change their speed randomly (*randomized*). Trees are extracted both using the original VIPER algorithm [4], and with our modifications (SafeVIPER). The safest results in each section are **bold**.
[1] In the safe extraction, our algorithm did not find a single safe tree that passed the safety check during the evaluation for three of five *unsafe* teacher policies. These were omitted from the averages.

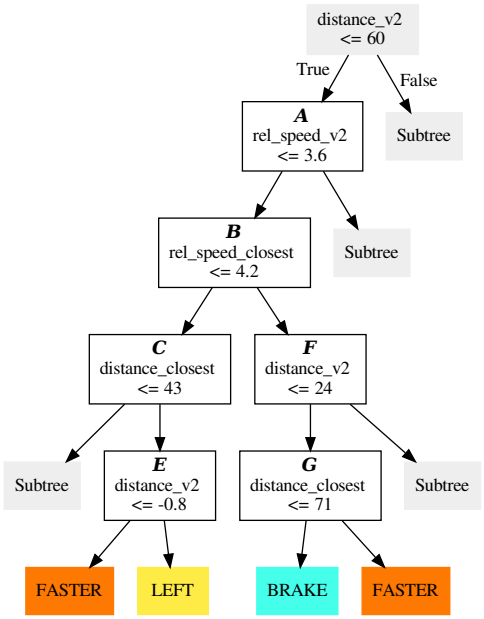

Fig. 5. Subtree of the full extracted tree shown in Fig. 4. Grey nodes show parts of the tree that were cut away. This subtree splits on the relative speed (`rel_speed`, i.e., $v_{\text{other}} - v_{\text{ego}}$) and the lane distance (`distance`, i.e., $x_{\text{other}} - x_{\text{ego}}$) to the two vehicles in our scenario. `closest` denotes the vehicle with the smaller absolute lane distance to our vehicle, while `v2` is the vehicle further away. In this subtree, we can explain the decisions in the following way: Nodes ***A*** and ***B*** check for the relative velocity of the two non-agent vehicles. Thus, if ***C*** is activated, we know that both vehicles are slower or only barely faster than the agent is. ***C*** splits on the distance to `closest`, going into the unlabeled subtree on the left if `closest` is either behind the ego vehicle, or less than $43\,\text{m}$ in front of it. ***E*** uses the distance to `v2`: if it is behind us, the agent accelerates, if it is in front of us, the agent changes lane to the left. In our scenario, this is sensible: If `closest` is the slower vehicle and `v2` (the faster one) is behind us, we can assume that we already changed the lane to overtake `closest`. Thus, with our lane clear, we can accelerate. If `v2` is the slower vehicle, and both vehicles are at most around $4\,\text{m/s}$ bit faster than we are (recall nodes ***A*** and ***B***), we can change our lane to indicate that we will overtake, and expect the other vehicles to cooperate. On the other labeled subtree, nodes ***F*** and ***G*** seem to perform a distance control task.

modifications) is paramount for the extraction of safe decision tree policies. Trees extracted from policies trained in the *safety* setting consistently achieve better safety than trees extracted from *baseline* policies.

C. Tree interpretability

Fig. 4 shows a visualization of a tree extracted from a *safety* teacher policy using SafeVIPER. Its small size and clear decision criteria make the tree highly interpretable. Each decision can be understood easily by manually retracing the decisions made by the tree. This gives the opportunity to inspect the generated tree in detail, and to figure out if certain decisions are not optimal. We found that many decisions and subtrees permit a hierarchical interpretation: Tree nodes at the top of the tree establish a certain situation (e.g., slow non-agent-vehicles), while the sub-trees below these nodes perform detailed controlling tasks for, or reactions to, these situations. In Fig. 5 a detailed look at a part of the full tree is shown.

TABLE III
TREE VERIFICATION

Extraction Algorithm	Teacher Policy	Provably safe?
VIPER	*baseline*	✗
VIPER	*safety*	✗
SafeVIPER	*baseline*	✗
SafeVIPER	*safety*	✓

Overview over the verification results. We tested multiple trees extracted from both *baseline* and *safety* configurations, using VIPER and SafeVIPER for the extraction. Only a *safety* teacher policy and SafeVIPER led to a tree that was verifiably safe.

D. Verification of decision trees

Running the verification on a local machine with an AMD Ryzen 7 2700 eight-core processor and 16GB of RAM took around two hours, but varied wildly depending on the policy size and implementation details in the verification environment. Results were usually available quicker if the verification failed (i.e., if Z3 found a counterexample that shows that the policy acted wrong). With a full implementation of the more complicated linearized bicycle model described in Sec. IV-B verification timed out after 24h. While we did not test this extensively, we noticed that single-core performance was crucial for the verification. Z3 did not run many calculations in parallel, even though we activated and tuned the appropriate settings to the best of our knowledge.

We tested the safety verification for trees extracted from policies trained with the *baseline* and *safety* modifications, and extracted with VIPER and SafeVIPER. We found that only our full pipeline, i.e., the combination of a teacher policy with *safety* modifications and the safe extraction procedure SafeVIPER generated decision trees that verified as safe. Trees extracted using the vanilla VIPER algorithm often showed safe behavior in the linearized environment, but our verification approach was able to identify specific conditions where they failed to take the right decision. Table III summarizes the results. This highlights the importance of extensive testing and verification methods: identifying small errors in policies can be infeasible through empirical validation (e.g., when their circumstances are comparatively rare). Here, verification provides an effective way to reliably find catastrophic behavior of a policy in known environment dynamics.

VI. CONCLUSION

Through the combination of prior work on safe reinforcement learning (RL) and interpretability, we constructed a pipeline for the creation of interpretable, safe policies. To the best of our knowledge, we are the first to propose an adoption of VIPER [4] to tasks in autonomous vehicles with a verification of the safety of the generated trees. We presented SafeVIPER, adapting VIPER for extraction in constrained Markov decision processes (CMDPs). Our policies achieve perfect safety in the environment they are trained in, and can be verified to be provably safe in linearized environments. They also generalize robustly to more demand-

ing evaluations. Furthermore, the policies are small decision trees, which makes them highly interpretable. While this paper focused on an autonomous driving use-case our pipeline is generally applicable to a wide range of tasks and domains.

The presented pipeline can not only transform the result of the highly flexible and versatile RL into a both understandable and provably safe autonomous driving strategy, but is a fully *automated* procedure towards dependable, trusted artificial intelligence (AI) in critical real world applications.

ACKNOWLEDGEMENTS

This work was supported by the Bavarian Ministry for Economic Affairs, Infrastructure, Transport and Technology through the Center for Analytics-Data-Applications (ADA-Center) within the framework of "BAYERN DIGITAL II".

REFERENCES

[1] E. Leurent, "Safe and efficient reinforcement learning for behavioural planning in autonomous driving," Ph.D. dissertation, University of Lille, France, 2020.

[2] S. A. Seshia, D. Sadigh, and S. S. Sastry, "Towards verified artificial intelligence," *arXiv preprint arXiv:1606.08514*, 2016.

[3] "Assumptions for models in safety-related automated vehicle behavior," https://sagroups.ieee.org/2846/, accessed: 2020-10-30.

[4] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Advances in neural information processing systems*, 2018, pp. 2494–2504.

[5] A. Kesting, M. Treiber, and D. Helbing, "General lane-changing model mobil for car-following models," *Transportation Research Record*, vol. 1999, no. 1, pp. 86–94, 2007.

[6] A. Li, L. Sun, W. Zhan, and M. Tomizuka, "Multiple criteria decision-making for lane-change model," *arXiv preprint arXiv:1910.10142*, 2019.

[7] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen, "Flow: Architecture and benchmarking for reinforcement learning in traffic control," *arXiv preprint arXiv:1710.05465*, 2017.

[8] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *arXiv preprint arXiv:2002.00444*, 2020.

[9] B. Goodman and S. Flaxman, "European union regulations on algorithmic decision-making and a "right to explanation"," *AI magazine*, vol. 38, no. 3, pp. 50–57, 2017.

[10] T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai, "Man is to computer programmer as woman is to homemaker? debiasing word embeddings," in *Advances in neural information processing systems*, 2016, pp. 4349–4357.

[11] J. Larson, S. Mattu, L. Kirchner, and J. Angwin, "How we analyzed the COMPAS recidivism algorithm," https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm, 2016, accessed: 2020-10-30.

[12] N. Burkart and M. F. Huber, "A survey on the explainability of supervised machine learning," *Journal of Artificial Intelligence Research*, vol. 70, pp. 245–317, 2021.

[13] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proc. IEEE Intl. Conf. Computer Vision*, 2017, pp. 618–626.

[14] A. Binder, G. Montavon, S. Lapuschkin, K. Müller, and W. Samek, "Layer-wise relevance propagation for neural networks with local renormalization layers," in *Artificial Neural Networks and Machine Learning (ICANN)*, Barcelona, Spain, 2016, pp. 63–71.

[15] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[16] C.-J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *IEEE Int. Conf. on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2148–2155.

[17] C. Hoel, T. Tram, and J. Sjöberg, "Reinforcement learning with uncertainty estimation for tactical decision-making in intersections," in *IEEE Int. Conf. on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–7.

[18] F. Ye, X. Cheng, P. Wang, and C.-Y. Chan, "Automated lane change strategy using proximal policy optimization-based deep reinforcement learning," *arXiv preprint arXiv:2002.02667*, 2020.

[19] B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker, "High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning," in *IEEE Int. Conf. on Intelligent Transportation Systems*, 2018, pp. 2156–2162.

[20] M. Kaushik, V. Prasad, K. M. Krishna, and B. Ravindran, "Overtaking maneuvers in simulated highway driving using deep reinforcement learning," in *IEEE Intelligent Vehicles Symp.*, 2018, pp. 1885–1890.

[21] G. Kontes, D. Scherer, T. Nisslbeck, J. Fischer, and C. Mutschler, "High-speed collision avoidance using deep reinforcement learning and domain randomization for autonomous vehicles," in *IEEE Int. Conference on Intelligent Transportation Systems*, 2020.

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

[23] H. Krasowski, X. Wang, and M. Althoff, "Safe reinforcement learning for autonomous lane changing using set-based prediction," in *IEEE Int. Conf. on Intelligent Transportation Systems*, 2020.

[24] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Association for the Advancement of Artificial Intelligence Conf. on Artificial Intelligence*, S. A. McIlraith and K. Q. Weinberger, Eds., New Orleans, Louisiana, USA, Feb. 2018, pp. 2669–2678.

[25] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Association for the Advancement of Artificial Intelligence Conf. on Artificial Intelligence*, Honolulu, Hawaii, USA, Jan. 2019, pp. 3387–3395.

[26] D. Chen, L. Jiang, Y. Wang, and Z. Li, "Autonomous driving using safe reinforcement learning by incorporating a regret-based human lane-changing decision model," in *Proc. American Control Conf.*, 2020, pp. 4355–4361.

[27] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[28] A. Stooke, J. Achiam, and P. Abbeel, "Responsive safety in reinforcement learning by PID lagrangian methods," in *Int. Conf. on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 9133–9143.

[29] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Int. Conf. on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 22–31.

[30] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[31] L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in *TACAS 2008*, ser. Lecture Notes in Computer Science, C. R. Ramakrishnan and J. Rehof, Eds., vol. 4963. Springer, 2008, pp. 337–340.

[32] E. Leurent, "An environment for autonomous driving decision-making," https://github.com/eleurent/highway-env, 2018.

[33] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical Review E*, vol. 62, no. 2, p. 1805–1824, Aug 2000.

[34] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[35] P. Polack, F. Altché, B. d'Andréa-Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *IEEE Intelligent Vehicles Symp.*, Los Angeles, CA, USA, Jun. 2017, pp. 812–818.

[36] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *American Control Conf.*, New York, NY, USA, Jul. 2007, pp. 2296–2301.