SAFE SCENARIO PARAMETER REGION SYNTHESIS FOR END-TO-END AUTONOMOUS DRIVING

Anonymous authors
Paper under double-blind review

ABSTRACT

End-to-end autonomous driving models have demonstrated great potential in driving safely under *normal* conditions. Existing techniques that evaluate their behavior on long-tail scenarios focus on generating diverse scenarios to test the driving system. In this work, we characterize safety-critical scenarios by certain parameters, and study the synthesis of the set of values for these parameters under which the Autonomous Vehicle is safe. We formalize the problem of synthesizing safe and maximal scenario parameter regions, and propose a framework to solve it. Our framework, ODIN, uses a counterexample-driven algorithm to achieve its objective. We evaluate ODIN on safety-critical scenarios for efficacy in synthesizing safe regions, and to produce safe/unsafe pairs of parameters using the regions.

1 INTRODUCTION

End-to-end AI-based autonomous driving systems (ADS) have recently shown great advances, and have been adopted by auto manufacturers such as Tesla (Elluswamy, 2025), Waymo (Waymo, 2025), Rivian (Rivian, 2025), and others (Wayve, 2025; Huawei Central, 2025). These end-to-end models take as input raw camera images and optionally other sensors like LiDAR or Radar, and produce the vehicle controls for steering, braking, and acceleration directly, avoiding intermediate separate modules for perception, prediction, and planning. Unlike modular systems, end-to-end systems trained with the end goal of driving well and safely avoid formalizing the notions of what such driving entails for various components, especially at the interface of perception and planning‘Elluswamy (2025).

However, despite extensive training and testing, these models have been responsible for crashes that have led to the loss of life and property. Recently, the National Highway Traffic Safety Administration (NHTSA) has escalated an investigation into Tesla's "Full Self-Driving" systems, involving over 3.2 million Tesla vehicles (NHTSA, 2025; CNBC, 2026). Waymo recently recalled software for over 1000 vehicles due to violations of traffic safety laws (Reuters, 2025; TechCrunch, 2025).

While end-to-end autonomous driving models today work for most common driving conditions and scenarios, they tend to fail more often in particular rare scenarios that are not well represented in their training set. These include rare environment conditions (road conditions, weather) as well as rare *safety-critical scenarios*, e.g., crossing a railway track as the gate closes, stopping behind a school bus with children crossing, people darting across the road from occluded sidewalks, etc. (see (Elluswamy, 2025; Chen et al., 2024)).

The shift from modular driving systems to end-to-end AI models has made the problem of giving safety guarantees for systems in particular scenarios a much harder problem. Generating safety-critical scenarios and adversarially testing the autonomous vehicle (AV) using high-fidelity driving simulation environments has therefore become an important research direction (Hanselmann et al., 2022; Wang et al., 2021; Ding et al., 2020; Abeysirigoonawardena et al., 2019; Ding et al., 2021; Zhang et al., 2023). Key advantages include simulating safety-critical scenarios rare in datasets, avoiding physical testing that is hard/impossible in such settings (like pedestrians being run over, or heavy fog), and testing scenarios outside the distribution of the training set (Fremont et al., 2019; Kim et al., 2022; Vin et al., 2023; Lin et al., 2024; Baresi et al., 2025).

In this paper, we consider safety-critical scenarios parameterized by certain variables (called *scenario parameters*) that define varying aspects of the scenario (e.g., the time and speed at which a pedestrian crosses the road, weather parameters, speed of other vehicles on the road, etc.). Our aim is to study the problem of characterizing the set of scenario parameter valuations under which an AV is safe.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

While many existing techniques try to identify safety-critical scenarios in which an AV fails, our goal is to characterize the safe regions. These regions hence also derive a clearer *safety frontier* – i.e., the valuations of scenario parameters that are near the boundary of this region exhibiting safe and unsafe behavior. Selecting parameters that with a slight change lead the vehicle to behave unsafely is crucial to understanding and improving the limits of an end-to-end driving model. For instance, the safety regions can be used to retrain models on boundary unsafe scenarios, evaluate models and compare them (those with larger safety regions rated higher for safety), study pairs of safe/unsafe scenarios at the boundary to understand failures, etc. Of course, the problem of precisely identifying safe parameter regions is not tractable for realistic large systems. We instead seek tractable solutions that approximate this problem to learn interpretable models that aim to distill the safety region from a black-box ADS.

Our Work: We present a novel formulation of the problem of *safe parameter region synthesis*. Given a parameterized safety-critical scenario, and a set of safe configurations, the task is to synthesize a locally maximal region (subset of the parameter space) anchored around these safe, under which the AV stays safe. Furthermore, we require a statistical guarantee that region identified is safe, established by sampling the space. We give a precise formulation of the problem in Section 3 in terms of regions defined using a logic $\mathcal{L}$ and with nuanced notions to capture (local) maximality, which ensures the largest regions of scenario parameters that result in safe simulation outcomes are synthesized. Such regions effectively present an approximation of the safety frontier anchored around the initial set of safe points. Sampling a pair across this frontier gives us contrasting scenarios that are safe and unsafe.

Second, we propose a novel *counterexample-guided synthesis framework* for constructing safe and (locally) maximal scenario parameter regions in a user-provided logic $\mathcal{L}$. Our algorithm uses a learner-tester architecture: (1) The learner learns a logical formula $M \in \mathcal{L}$ that acts as a classifier of safe and unsafe parameter valuations for the scenario, using a finite set of samples of configurations, labeled safe and unsafe. (2) The simulation-based tester checks if the proposed candidate M is (locally) maximal and statistically safe. Our algorithm iteratively refines the formula using counterexamples that the tester returns, until convergence on a provably maximal and statistically safe region.

We implement a realization of our framework in a tool called ODIN that synthesizes parameter regions as decision trees ($\mathcal{L}$ is quantifier-free boolean combinations of inequality thresholds over scenario parameters). We evaluate ODIN on seven challenging safety-critical scenarios in autonomous driving, involving pedestrians for the Latent TransFuser v6 end-to-end driving model (Nguyen et al., 2026). We show that our algorithm converges for different maximum depths of the decision tree, and is effective in finding configuration pairs that lie close to the boundary of the safe region.

Contributions. In this work, we make the following main contributions:

- A novel formulation of the problem of synthesizing statistically safe locally maximal scenario-parameter regions anchored around a given set of safe configurations for end-to-end driving models (§3).
- A counterexample-driven learning framework to solve the above problem (§ 4) for parameter regions expressed using quantifier-free Boolean formulas over inequalities, expressed using decision trees.
- An instantiation and evaluation of our framework's effectiveness on seven safety-critical scenarios involving cars, pedestrians, and cyclists under various environmental conditions (§ 5 and § 6).

2 ILLUSTRATIVE EXAMPLES

Consider a driving scenario in which a pedestrian crosses the road suddenly from the left in foggy conditions. In this setting, we model the scenario using parameters that capture fog density, sun altitude angle, and pedestrian speed (assuming a specific time they cross the road).

The safe scenario synthesis problem is to find, given a set of safe parameter valuations, a maximal region anchored around these under which the AV does not collide with the pedestrian. In this setting, ODIN synthesizes a decision tree of depth 5 capturing a safe maximal region. Maximality ensures that there are unsafe configurations close to the boundary of the safe region. Consequently, we sample configurations close to this boundary to obtain such pairs. Figure 1 illustrates such a pair: the scenario on the left, with less fog, is safe (the ego vehicle seems to recognize the pedestrian and apply the brakes), while the one on the right is unsafe (ego vehicle doesn't slow down). Except for fog, all the scenario parameter values are the same for the two scenarios presented (fog is 25.0 in the safe scenario, and 40.0 in the unsafe scenario). This pair was sampled across the boundary of a safe region learned by ODIN. One could use such pairs to better understand the failure modes of models, and to even improve the driving models.



Figure 1: (Left) Safe scenario where fog = 25.0 (Right) Unsafe scenario where fog = 40.0

Consider a second scenario, where the ego vehicle makes a left turn and immediately finds a pedestrian crossing the road. In this case, ODIN synthesizes a region depicted by the decision tree shown on the right in Figure 2. Notice that the parameters for the safety of this region involve both weather (rain) and non-weather (speed of the pedestrian) parameters. In particular, the region says that if the rain is heavy, then if the pedestrian is slow, they may get hit by the AV, while if they are fast, they will cross the road fast enough, avoiding collision. The boundaries of these regions, give opportunities to improve the model. ODIN can synthesize trees of larger size, and the tree on the right is only illustrative.

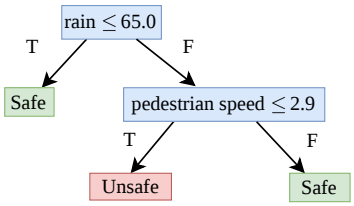


Figure 2: Example decision tree with weather and non-weather parameters

3 FORMALIZING SAFE SCENARIO PARAMETER REGION SYNTHESIS

We model the world (which includes the ego vehicle and its entire environment) using variables that range over particular domains. Let us fix a set of variables V, where each variable $x \in V$ is implicitly over a domain D_x. A valuation v of these variables captures a particular configuration of the world, and is called a *state of the world*. For any set of variables X, let $\overline{X}$ denote the set of all valuations of variables in X over their respective domains. Let $State = \overline{V}$ denote the set of states of the world. Let $SysV \subseteq V$ denote a set of *system variables*, which are explicit variables tracked by the internal software of the ego vehicle that is driven by an end-to-end driving model. Also, let $SCV \subseteq V$ be a set of scenario-controlled variables controlled by the scenario-modifying module, which captures crucial aspects of a scenario (e.g., the position of a pedestrian moving in the environment).

The transition of a state of the world from v can result in many possible states, and is determined by two sets of variables:

- Control inputs (CI) of the ego vehicle (e.g., steering, accelerator, etc.)
- Scenario parameters ($ScenarioParams$) that control both static aspects of the environment (e.g., the angle of the sun) as well as dynamic aspects of *how* actors in the environment behave (e.g., how a pedestrian moves in the environment, whether a traffic light changes green, etc.)

We assume a Dynamics Engine ($DynE$) that computes the set of next states from a given state:

$$DynE : State \times \overline{CI} \times \overline{SCV} \to 2^{State}$$

It maps a state (a valuation over V), a control input (a valuation of CI), and scenario controlled values (valuation over SCV) to a *set of* states of the world. Note that the transition can be nondeterministic.

We model the ego vehicle using two components, a sensor and a learned model for driving:

$$Sensor : State \to 2^{\overline{SensorOutput}} \qquad DrivingModel : \overline{SensorOutput} \times \overline{SysV} \to 2^{\overline{CI}}$$

In the above, $SensorOutput$ is a set of variables that captures the outputs of the sensors in the ego vehicle, such as images from a camera or LiDAR point clouds. The $DrivingModel$ is a learned module that takes the sensor outputs and returns a plan for how to drive the car (typically using *waypoints*), which are then translated to actual control inputs for driving the car (e.g., steering angle and throttle). We capture these two modules as functions (they map to sets, capturing nondeterminism). Next, we model the scenario emulator as a function

$$ScenarioEmul : State \times \overline{ScenarioParams} \to \overline{SCV}$$

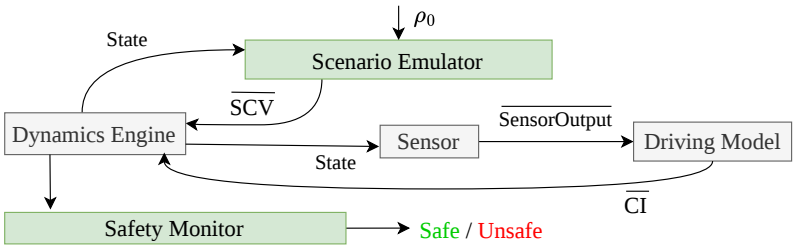


Figure 3: Effect of Scenario Parameters on execution, SCV: Scenario Controlled Variables, CI: Control Inputs

This emulator maps the current state of the world and the current value of scenario parameters ρ to new values for the scenario-controlled variables. The transition of the entire system from a state s with respect to fixed scenario parameters $\rho \in \overline{ScenarioParams}$ is defined as:

$$\delta(s,\rho) = DynE(s,\, DrivingModel(Sensor(s), s{\downarrow}SysV),\, ScenarioEmul(s,\rho))$$

In the above, we assume the use of composition of functions that map to sets (e.g., if $f : A \to 2^B$ and $g : B \to 2^C$, then $g(f(x)) = \{y \in C \mid \exists z \in B.\ z \in f(x)$ *and* $y \in g(z)\}$, etc.).

Consequently, $\delta(s,\rho)$ above is the set of new states that can result from s, when the scenario evolves according to ρ. Note that the scenario parameters remain fixed throughout a run— they are used by the scenario emulator in evolving the variables SCV across time, which affect the state.

A *trace* of the entire system, with respect to a fixed valuation of scenario parameters $\rho \in \overline{ScenarioParams}$, is a sequence of states $s_0, s_1, \ldots s_n$ such that $s_{i+1} \in \delta(s_i,\rho)$ for each $0 \leq i < n$, where fixed n is the number of execution steps.

A safety monitor is a function $SafeMonitor : State \to \{safe, unsafe\}$ that determines whether a state is safe or not (e.g., it detects a collision of the ego vehicle with another vehicle or a pedestrian, the ego vehicle going off the road or onto a barricade, etc.). The scenario developer provides this function for each scenario, potentially based on traffic laws and regulations.

Definition 1 (Safe Scenario Parameters)
A trace $s_0, s_1, \ldots s_n$ is said to be safe if all its states are safe, i.e., $SafeMonitor(s_i) = \text{`}safe\text{'}, \forall i \in [0,n]$.
A scenario parameter evaluation ρ is safe if all traces with respect to ρ are safe.
A set of scenario parameters is safe if all traces for all parameters in the set are safe.

3.1 Predicates on Scenario Parameters

We describe sets of scenario parameters in a logical form as predicates. Let us fix a logic $\mathcal{L}$ to express properties on scenario parameters $ScenarioParams$, thereby describing regions of scenario parameters. A predicate M in $\mathcal{L}$ maps each scenario parameter evaluation to one of $\{True, False\}$, and hence identifies a subset of scenario parameters (those that evaluate to $true$). Let $[\![M]\!]$ denote the set of parameter valuations that M maps to $True$.

In the rest of the paper, we refer to a scenario parameter evaluation as a *configuration* for brevity. We next precisely define the notion of the (local) maximality of the safe predicates.

Definition 2 (Expansions) *An* expansion *of the predicate M is a predicate M', such that $M \Rightarrow M'$. Hence M' defines a larger parameter space than M.*

Let us fix a function $Expand$ that defines a finite set of expansions for every predicate $M \in \mathcal{L}$.

Definition 3 (Maximality) *A predicate M is a maximal safe predicate if it is safe and every expansion of M (i.e., $Expand(M)$) is not safe.*

Informal Problem Formulation: The central problem we tackle in this paper is, informally, the following *ideal* safe scenario parameter synthesis problem: given an initial set of positively labeled points P, find a *maximal safe* predicate $M(ScenarioParams)$ expressed in $\mathcal{L}$ that includes P.

We formalize below the precise problem we tackle.

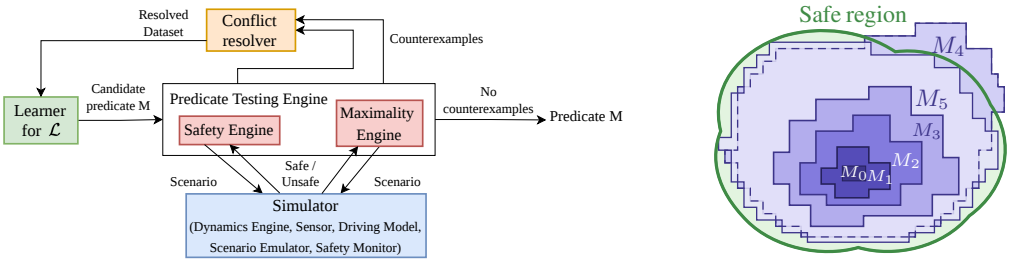


Figure 4: (a) ODIN architecture (b) Safe expansions of Scenario Parameter Regions

3.2 SAFE SCENARIO PARAMETER SYNTHESIS AS ANCHORED LEARNING

Intuitively, the informal problem formulated above asks for which valuations of parameters that control the scenario lead the ego vehicle to behave safely in the scenario. The above problem is, of course, not tractable. In fact, even given a fixed predicate M on scenario parameters, checking whether all traces according to it are safe is essentially impossible to verify, as the driving model is based on end-to-end ML models that are very large. Furthermore, the Dynamics Engine, even when formulated as simulators, is a complex software module that has realistic rendering components, additional ML models for rendering, etc.

Consequently, we define the precise Safe Scenario Parameter Synthesis Problem that we tackle in this paper as follows. First, let us assume that various components, such as the Dynamics Engine, Sensor, and Driving Model, are made deterministic by introducing a set of seeds, $Seeds$, that characterize the nondeterminism they exhibit.

Let us define the problem of *Anchored Learning with Statistical Guarantees*. We are given a set of positively labeled (safe) configurations P and a set of negatively labeled (unsafe) configurations N. We seek an active learning algorithm with membership queries only (which return whether a configuration is safe or unsafe), which could discover a larger set of positive and negative points, $P' \supseteq P$ and $N' \supseteq N$. We require this active learning algorithm to learn a maximal region defined by a predicate M that is anchored around the initial set of positive points P and avoids all the negative points N'. Furthermore, we require that the region of configurations learned is *maximal* in the sense that every expansion of the region is provably unsafe, in that the expansion has a discovered negative point in N'. Finally, we require that there is a statistical guarantee of safety of the learned region M.

Unlike usual learning problems, our learning does not require finding positive regions in far disconnected regions from the anchored points P; rather, we are required only to find (unions of) maximal continuous statistically-safe regions around the anchored points P. This is crucial in order to build tractable learning as the entire space of configurations is too large to learn from using effective sampling.

Definition 4 (Anchored Learning with Statistical Safety Guarantees Testing:) *Given a set of positively labeled points P and a set of negatively labeled points N, the problem of safe scenario parameter synthesis is to define an active learning algorithm that makes membership queries, discovering a further set of positive and negative points, $P' \supseteq P$ and $N' \supseteq N$, and outputs a predicate $M \in \mathcal{L}$ that satisfies the following:*

- ***Conformance:*** *$P \subseteq [\![M]\!]$ and $N' \cap [\![M]\!] = \emptyset$*
- ***Maximality:*** *For every expansion M' of M such that $M \not\equiv M'$, $[\![M']\!] \cap N' \neq \emptyset$*
- ***Statistical safety:*** *Given n uniformly sampled configurations $\rho_1, \rho_2, ..., \rho_n \in [\![M]\!]$, let the estimated probability of safety be $\hat{p} = |\{\rho_i : \rho_i \text{ is safe}, i \in [1..n]\}|/n$, a provided lower bound p_0, confidence level c_0, and p be the probability that a ρ sampled uniformly from $[\![M]\!]$ is safe, with its c_0-level confidence interval being $[p_L, p_U]$. Then M is statistically safe if $p_L \geq p_0$.*

Note that the learned predicate M need not include all new positive configurations discovered ($P' \setminus P$) and can exclude some of them; this may be necessary as the logic of predicates may not be powerful enough to define a perfect classifier, and also may be necessary in order to satisfy the safety guarantee.

4 A SYNTHESIS FRAMEWORK FOR SAFE SCENARIO PARAMETER REGIONS

We now propose a *counterexample-guided synthesis framework* ODIN for constructing safe and maximal scenario parameter regions in a logic $\mathcal{L}$. Our approach, illustrated in Figure 4a, uses a

learner-tester architecture. The learner learns a logical formula $M \in \mathcal{L}$ that acts as a classifier of safe and unsafe parameter valuations, using a finite sample of configurations, labeled safe and unsafe. The candidate M proposed by the learner is tested for safety and maximality (*Predicate Testing Engine*) using simulation-based testing (*Simulator* in the figure).

If ODIN does not find counterexamples to safety and maximality, it outputs the predicate. Otherwise, counterexamples are filtered through a *conflict resolver* that updates the set of safe/unsafe configurations, and ODIN asks the learner to learn a new predicate. The conflict resolver adds the new counterexamples, but may change the labels of some samples from safe to unsafe when they are close to truly unsafe configurations (this is described below). The algorithm iterates till a predicate is found that passes both safety and maximality tests. Note that successive iterations can produce both safe and unsafe regions as depicted in Figure 4b. We provide ODIN's learning algorithm in Appendix C.

We now describe some of the components of our framework. We describe components both in generic terms (for any logic $\mathcal{L}$ and any notions of expansions) as well as in the concrete context where $\mathcal{L}$ is decision trees, which we realize in our tool.

Quantifier-free Boolean Logic, Decision Trees, and their Expansions One particular logic $\mathcal{L}$ that we are interested in is quantifier-free Boolean combinations of constraints over the scenario parameters and their thresholds. This restricts the predicate syntax to the grammar:

$$p := \texttt{true} \mid \texttt{false} \mid q > \alpha \mid q \leq \alpha \mid p \vee p \mid p \wedge p \mid \neg p \mid \texttt{ite}(p,p,p)$$

where $\alpha \in \mathbb{R}$, $q \in \mathit{ScenarioParams}$, `ite` is the *if-then-else* expression. A predicate M in $\mathcal{L}$ can be naturally represented as a decision tree (a normal form for formulas that uses only `ite` constructs, with atomic predicates decorating internal nodes and leaves being `true`/`false`). This makes $\mathcal{L}$ amenable to synthesis using fast decision-tree learning algorithms (Breiman et al., 1984). A decision tree corresponds to a formula that is a disjunction of conjunctions, $\bigvee_i \bigwedge_j \varphi_{ij}$ where φ_{ij} is any atomic predicate or its negation. The inner conjuncts correspond to constraints along a particular path.

Expansions of M depicted as a decision tree: Let us now define the set of predicates explored when expanding M. Expansions are parameterized by Δ, which identifies a positive real number to each scenario parameter (e.g., $\Delta(\mathit{fog}) = 10.0$). This determines the level of relaxation in constraints involving each parameter $q \in \mathit{ScenarioParams}$. $\mathit{Expand}(\varphi, \Delta)$ is defined inductively as follows:

$$\mathit{Expand}(q > \alpha, \Delta) = q > \alpha - \Delta(q) \qquad \mathit{Expand}(q \leq \alpha, \Delta) = q \leq \alpha + \Delta(q)$$
$$\mathit{Expand}(p \wedge p', \Delta) = \mathit{Expand}(p, \Delta) \wedge \mathit{Expand}(p', \Delta)$$
$$\mathit{Expand}(p \vee p', \Delta) = \mathit{Expand}(p, \Delta) \vee \mathit{Expand}(p', \Delta)$$

For any formula φ, $\varphi \Rightarrow \mathit{Expand}(\varphi, \Delta)$. The expansion of a decision tree DT is now defined to be $\{\mathit{Expand}(DT, \Delta)\}$ if $\mathit{Expand}(DT, \Delta) \not\Rightarrow DT$, and the empty set otherwise. Essentially, we relax each atomic predicate along a true path, by its parameter-specific Δ values. And the expansion of a decision tree is this expanded formula provided it's strictly weaker. In definitions of maximality, we require that the expansion includes at least one demonstrably unsafe configuration.

Predicate Testing Engine. The predicate testing engine contains two components: the ***Safety Engine*** and the ***Maximality Engine***. The Safety Engine randomly samples scenario configurations that satisfy the proposed predicate M, and checks the safety of the resulting traces. If all the sampled configurations yield safe traces, it declares M to be safe. The Maximality Engine samples configurations that are in some expansion of the predicate M, but not included in M. If at least one of the sampled configurations from each expansion yields an unsafe trace, it declares M to be maximal. Counterexamples to safety (unsafe configurations) and maximality (safe configurations) are returned otherwise.

Our instantiation of the safety engine for decision trees works by taking each conjunctive formula (for a path of the decision tree), expressing the conjunctions as interval constraints on each parameter, and then sampling these intervals according to a distribution. ODIN's Maximality Engine for decision trees works similarly. It takes each conjunctive formula (for a path in the decision tree), expresses it as well as its expansion as interval constraints, and samples configurations that lie in the expanded intervals but not in the original intervals. It also discards samples if the decision tree ignored them previously.

Our outer iterative loop in general may not *converge* (or not converge fast) as counterexamples can cause regions to grow slowly. To avoid such slow growth, we ensure that intervals for each scenario parameter q grow by at least $\epsilon(q)$, where $\epsilon(q)$ is chosen in $[0, \Delta(q)]$. We do this by having the maximality engine sample configurations that are in $\mathit{Expand}(M, \Delta)$ but not in $\mathit{Expand}(M, \epsilon)$.

Figure 4b shows an illustration of how the maximality testing expands the initial predicate M_0. The initial predicate is expanded to M_1,M_2,M_3, which turn out to be safe. But expanding to M_4 includes unsafe configurations. In successive iterations of testing by the safety engine, M_4 was found to contain unsafe configurations, so the learner shrank the predicate to M_5.

Another reason for non-convergence is that the decision tree learner ignores the maximality counterexamples (refusing to include them in the safe region). This can happen for several valid reasons (e.g., the logic being expressive to include this point while excluding other unsafe points). If we kept returning maximal counterexamples that the learner excludes, the process can diverge. We avoid this by not sampling in Δ-ϵ expansions that already contain a previously returned maximality counterexample.

Conflict Resolver. The role of the conflict resolver is to reclassify certain safe configurations returned by the maximal counterexample generator as *unsafe* for the decision tree learner. The conflict resolver relabels a configuration ρ from safe to unsafe if there is a configuration ρ' (already in the set of sampled configurations) that is *close* to ρ but is demonstrably unsafe. ODIN's conflict resolver uses a discretization-based definition of *closeness*, i.e., we divide the range of possible values of the scenario parameters into buckets (of size determined by Δ for each scenario parameter), and declare two configurations as being close if they fall into the same buckets for all scenario parameters. We provide the algorithms for the predicate testing engine and the conflict resolver in Appendix C.

Learner for logic $\mathcal{L}$. The learner takes in the dataset of configurations and their labels as input, and returns the predicate M in some representation. Our instantiation of the framework uses decision tree learning algorithms (Breiman et al., 1984; Quinlan, 1986) to learn decision trees that represent predicates in the logic we handle. We learn decision trees with a fixed maximum depth d to (a) ensure convergence (as otherwise the learner can specialize the tree to finer divisions), and (b) interpretability for users who examine the safe region (e.g., to sample pairs of safe/unsafe configurations at the boundary).

5 EXPERIMENTAL METHODOLOGY

We instantiate our framework with the decision tree learner from Scikit-learn (Pedregosa et al., 2011) as the learner, and we use the CARLA (Dosovitskiy et al., 2017) simulator to simulate driving scenarios and synthesize the safe scenario parameter regions for the Latent TransFuser v6 (Nguyen et al., 2026) end-to-end driving model (that achieves top performance on major Carla Leaderboard 2.0 benchmarks). The model takes images from front-view and rear-view cameras as sensor inputs to predict the waypoints to be followed and the target speed to be achieved by the ego vehicle. We study seven scenarios with different modes of interaction between the autonomous vehicle and pedestrians, cyclists, and other cars, taken from prior work (Jia et al., 2024; Gerstenecker et al., 2026)

- *Pedestrian crossing from an occluded sidewalk*: A pedestrian, standing behind an advertisement board on the sidewalk, comes onto the road, crossing the ego vehicle's path from its right side.
- *Pedestrian crossing neighboring lane*: A pedestrian, standing behind an advertisement board on a divider, walks across a lane and crosses the ego vehicle's path from its left.
- *Pedestrian crossing road at intersection*: A pedestrian standing at the exit of an intersection and visible to the ego vehicle comes onto the ego vehicle's path just after it has crossed the intersection.
- *Traffic hazard at parking lot*: Three stationary cars are in the ego vehicle's lane, in a parking lot, and there is oncoming traffic.The ego vehicle should drive around the stationary cars.
- *Cyclists in front of the ego car* : Two cyclists are riding their bicycles in the lane of the ego vehicle. The ego vehicle is expected to either overtake them safely or keep driving behind them safely.
- *Curbside car entering traffic in the ego car's lane* : A curbside-parked car enters traffic in the ego car's lane. The ego car is expected to allow the car to enter the lane safely.
- *Pedestrians in the ego car's lane*: Three pedestrians are walking in the lane of the ego vehicle, in the direction of the ego vehicle. The ego vehicle is expected to drive around/behind them safely.

Safety. In all scenarios, we declare a trace of the world to be safe if there is no collision between the ego vehicle and the other actors in the scenario (pedestrians, cyclists, or other cars), and in cases where a pedestrian crosses the ego vehicle's lane, the ego vehicle applies the brakes within 3 meters of its interaction with the pedestrian. The scenarios where a pedestrian crosses the ego vehicle's lane are set up such that the vehicle has 2.0s to react to the pedestrian to avoid a collision. In the set of scenario parameters, we include weather parameters: Fog Density (**FD**), Cloudiness (**CD**), Precipitation (**PC**), Sun altitude angle (**SAL**), Sun azimuth angle (**SAA**), and actor parameters: Pedestrian speed (**NPC**), Blocking length (**BL**), Oncoming traffic frequency (**OTR**), Cyclist offset (**CO**), Cyclist speed

Table 1: Time and number of samples for convergence for varying tree depths

Scenario	Depth of Safe Region	Converges? (iterations)	Simulations sampled #	Time	Statistical check
Pedestrian crossing from an occluded sidewalk (parameters: FD, CD, PC, SAL, SAA, NPC)	2	✓(5)	80	1.32 h	0.9
	3	✓(16)	284	5.16 h	1.0
	4	✓(26)	1036	18.80 h	1.0
	5	✓(28)	1562	19.00 h	1.0 [0.84, 1.0]
	6	✓(20)	2638	35.40 h	0.9 [0.70, 0.98]
Pedestrian crossing neighboring left lane (parameters: NPC, FD, SAA)	2	✓(3)	50	0.33 h	1.0
	3	✓(3)	60	0.41 h	1.0
	4	✓(12)	215	3.50 h	1.0
	5	✓(18)	593	3.93 h	1.0 [0.84, 1.0]
	6	✓(26)	1165	11.6 h	0.9 [0.70, 0.98]
Pedestrian crossing road at intersection (parameters: NPC, PC, SAA)	2	✓(3)	45	0.54 h	1.0
	3	✓(3)	60	0.53 h	1.0
	4	✓(17)	452	8.70 h	1.0
	5	✓(35)	2176	16.10 h	1.0 [0.84, 1.0]
	6	✓(64)	6226	50.07 h	1.0 [0.84, 1.0]
Traffic hazard in a parking lot (parameters: BL, OTR)	2	✓(2)	30	1.75 h	0.96
	3	✓(2)	30	1.75 h	0.94
	4	✓(3)	50	2.92 h	0.85
	5	✓(4)	90	5.25 h	1.0 [0.84, 1.0]
	6	✓(4)	90	5.25 h	1.0 [0.84, 1.0]
Cyclists in front of the ego car (parameters: CO, CYS)	2	✓(7)	75	1.82 h	0.91
	3	✓(7)	110	2.65 h	0.87
	4	✓(10)	180	4.33 h	0.96
	5	✓(12)	245	5.92 h	0.9 [0.70, 0.98]
	6	✓(10)	235	5.65 h	0.85 [0.65, 0.95]
Curbside car entering traffic in the ego car's lane (parameters: RT, CS, MD)	2	✓(2)	25	0.16 h	0.81
	3	✓(6)	115	1.57 h	0.82
	4	✓(9)	240	3.37 h	1.0
	5	✓(8)	205	3.52 h	0.8 [0.59, 0.92]
	6	✓(19)	1185	16.49 h	0.95 [0.77, 0.99]
Pedestrians in the ego car's lane (parameters: WO, DTW, WS)	2	✓(4)	55	3.96 h	0.8
	3	✓(4)	60	4.30 h	0.8
	4	✓(5)	125	9.05 h	1.0
	5	✓(15)	475	34.35 h	1.0 [0.84, 1.0]
	6	✓(14)	465	33.67 h	0.95 [0.77, 0.99]

(**CYS**), Reaction time (**RT**), Cut-in speed (**CS**), Minimum trigger distance (**MD**), Walkers offset (**WO**), Distance to walkers (**DTW**), Walkers speed (**WS**).

In Appendix A, Table 2 describes the scenario parameter space spanned by the parameters and their range of values. For each scenario, we learn the safe regions over a subset of all the scenario parameters (active parameters), and vary this subset over their full range of values and restrict the values of the other scenario parameters to safe ranges ("Canonical Safe Values" in Table 2). We provide the Δ and ϵ values for all scenario parameters in Table 2. In our experiments evaluating ODIN, we use $K = 5$ (the number of samples to draw from each region for safety and maximality checks). We instantiate sampling with uniform distribution in the ODIN algorithm. The learner-tester procedure begins by sampling N points from the space of scenario parameters. We sample at least one point that covers the extremal values from the domain of each active parameter, to avoid degenerate seed sets (e.g., all seeds are safe).We use $N = 20$ in our evaluations of ODIN, and experiment with different maximum depths d of the decision tree that ODIN learns. To give a statistical safety lower bound, we used Binomial test with Wilson's continuity correction (SciPy Developers, 2025) with confidence $c_0 = 0.9$ and 20 samples.

6 EXPERIMENTAL EVALUATION

Table 1 presents our experimental results. Column 1 presents the scenario and its scenario parameters (with abbreviated names and ranges from Table 2). Column 2 presents the depth of the learned decision tree. Column 3 reports whether the algorithm converged, and the number of learner-tester iterations it

took. Columns 4 and 5 report the total number of configurations simulated and the total running time of the procedure. Column 6 reports the result of the statistical check on the safe regions.

Convergence (RQ1). We check if the learning procedure converges, and if so, in how many iterations of the learner-tester loop. As reported in Table 1, ODIN converges for all the considered values of maximum depth of the decision tree. With increasing maximum depth, ODIN can explore more expressive regions and more configurations of scenario parameters to synthesize deeper trees that can capture interactions between the parameters better, as well as complicated shapes of safe regions. We also report the number of configurations simulated by ODIN, and the total time taken (including the simulator time, decision tree learning time, and the time for sampling configurations) along with convergence.

Statistical check (RQ2). After the procedure converges, we perform the statistical test to measure the quality of the synthesized region. We do this by uniformly drawing configuration samples from the safe region and checking how many of them are safe. We report the results of this testing in Table 1, under Statistical check. We report the statistical guarantee for the trees of depth 5 and 6, since they are learnt from a substantial set of training data and are more expressive. For each scenario, we find that at least one of these trees passes the statistical check with a lower bound for the mean of 0.7 at a confidence level of 0.9. For trees of depth 2-4, we report only the mean obtained on a smaller number of samples. We focused the computational effort to provide statistical guarantee on depths 5 and 6, as they are more expressive.

Algorithm performance across tree depths (RQ3). The trees of larger depths present more opportunity to find pairs of safe/unsafe configuration pairs, since their boundaries are more elaborate. We present such pairs ODIN produced in Appendix B, and discuss the learning times in Appendix D.

7 RELATED WORK

Testing autonomous driving systems. Prior works (Wang et al., 2021; Dreossi et al., 2019) have proposed techniques to test autonomous driving systems. Some approaches include using kinematics gradients to guide scenario generation (Hanselmann et al., 2022), probabilistic scene generation techniques (Fremont et al., 2019), and others (Zhang et al., 2023; Abeysirigoonawardena et al., 2019; Ding et al., 2021). These works aim to evaluate driving systems under rare scenarios, but do not characterize the scenarios using parameters and explore different parameters or regions. Our work provides a framework to characterize the safe regions of a driving model in the scenario parameter space.

Operational Design Domains. The safe conditions for the deployment of an autonomous system, also called Operational Design Domain (Thorn et al., 2018), are used by practitioners to decide the safe configurations for deployment. Prior works (Torfah et al., 2022) have proposed a formal notion of operational design domains, and synthesized monitors to predict if a vehicle is about to exit its ODD and violate a system-level safety property. Such an ODD monitor observes *semantic features* of the environment and predicts the satisfaction of the safety property across multiple scenarios. ODIN on the other hand uses *scenario-specific* semantic features to learn safe regions specific to each scenario.

Learning safety contracts. Our work is inspired by work on learning preconditions that keep software safe. For instance, see work in (Astorga et al., 2021), which explores safe and maximal precondition synthesis modulo testing. Similarly, (Astorga et al., 2023) proposes learning perception contracts that characterize the safety of an autonomous driving system under imperfect perception, and previous techniques in (Hsieh et al., 2022) and (Joshi et al., 2024) enable learning abstractions of safe driving trajectories to characterize safety in autonomous driving systems.

8 CONCLUSION

We proposed a framework to synthesize safe and maximal scenario parameter regions for end-to-end driving models. We demonstrate the effectiveness of the framework by learning the safe region for a state-of-the-art driving model, and demonstrate the pairs of safe and unsafe configurations sampled across the boundary of the safe and maximal region. Our formulation assumed that all valuations of scenario parameters are possible/equally likely. However, actors in a scene have particular behaviors, often modeled by another learning model (Tan et al., 2021; Bergamini et al., 2021). Extending our work to such natural environmental behavior is an interesting future direction. Our sampling-based approach is not designed to capture problems that occur only for a particular value of a parameter. We evaluated ODIN using a recent open source model for autonomous driving (Latent TransFuser v6). Evaluating ODIN on industry-grade (i.e., car manufacturers') models is left for future work.

9 AI USE STATEMENT

In this work, we have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, design or provide feedback on research methodology or experiments, implement methods, support qualitative and thematic data analysis, or interpret results. Assisting in the writing of proofs, proposing or refining hypotheses, assisting with translation, cleaning and reformatting datasets do not apply to this work. We used an AI-based tool to scan the paper for errors (grammatical, logical, notational bugs) after the paper was written by us. We reviewed all the potential errors surfaced by the tool. We take responsibility for the final content of this work.

REFERENCES

Yasasa Abeysirigoonawardena, Florian Shkurti, and Gregory Dudek. Generating adversarial driving scenarios in high-fidelity simulators. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8271–8277. IEEE, 2019.

Angello Astorga, Shambwaditya Saha, Ahmad Dinkins, Felicia Wang, P. Madhusudan, and Tao Xie. Synthesizing contracts correct modulo a test generator. *Proc. ACM Program. Lang.*, 5(OOPSLA), October 2021. doi: 10.1145/3485481. URL https://doi.org/10.1145/3485481.

Angello Astorga, Chiao Hsieh, P Madhusudan, and Sayan Mitra. Perception contracts for safety of ml-enabled systems. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA2): 2196–2223, 2023.

Luciano Baresi, Davide Yi Xian Hu, Andrea Stocco, and Paolo Tonella. Efficient domain augmentation for autonomous driving testing using diffusion models. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 398–410. IEEE, 2025.

Luca Bergamini, Yawei Ye, Oliver Scheel, Long Chen, Chih Hu, Luca Del Pero, Błażej Osiński, Hugo Grimmett, and Peter Ondruska. Simnet: Learning reactive self-driving simulations from real-world observations. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5119–5125. IEEE, 2021.

Leo Breiman, Jerome Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman and Hall/CRC, 1st edition, 1984. doi: 10.1201/9781315139470. URL https://doi.org/10.1201/9781315139470.

Li Chen, Penghao Wu, Kashyap Chitta, Bernhard Jaeger, Andreas Geiger, and Hongyang Li. End-to-end autonomous driving: Challenges and frontiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):10164–10183, 2024. doi: 10.1109/TPAMI.2024.3435937.

CNBC. Tesla faces intensifying NHTSA probe of FSD in reduced visibility. https://www.cnbc.com/2026/03/19/tesla-nhtsa-full-self-driving-fsd-reduced-visibility.html, March 2026. Accessed 2026-05-03.

Wenhao Ding, Baiming Chen, Minjun Xu, and Ding Zhao. Learning to collide: An adaptive safety-critical scenarios generating method. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2243–2250. IEEE, 2020.

Wenhao Ding, Baiming Chen, Bo Li, Kim Ji Eun, and Ding Zhao. Multimodal safety-critical scenarios generation for decision-making algorithms evaluation. *IEEE Robotics and Automation Letters*, 6(2): 1551–1558, 2021.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

Tommaso Dreossi, Daniel J Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A Seshia. Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *International Conference on Computer Aided Verification*, pages 432–442. Springer, 2019.

Ashok Elluswamy. Building an autonomous future. Keynote, 2nd Workshop on Distillation of Foundation Models for Autonomous Driving (WDFM-AD), IEEE/CVF International Conference on Computer Vision (ICCV), Honolulu, HI, USA, October 2025. Video recording, https://www.youtube.com/watch?v=IRu-cPkpiFk.

Daniel J Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit 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.

Simon Gerstenecker, Andreas Geiger, and Katrin Renz. Fail2drive: Benchmarking closed-loop driving generalization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026. to appear.

Niklas Hanselmann, Katrin Renz, Kashyap Chitta, Apratim Bhattacharyya, and Andreas Geiger. King: Generating safety-critical driving scenarios for robust imitation via kinematics gradients. In *European Conference on Computer Vision*, pages 335–352. Springer, 2022.

Chiao Hsieh, Yangge Li, Dawei Sun, Keyur Joshi, Sasa Misailovic, and Sayan Mitra. Verifying controllers with vision-based perception using safe approximate abstractions. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 41(11):4205–4216, November 2022. ISSN 0278-0070. doi: 10.1109/TCAD.2022.3197508. URL https://doi.org/10.1109/TCAD.2022.3197508.

Huawei Central. Huawei launches ADS 4.0 and HarmonyOS Cockpit 5 driving solutions. https://www.huaweicentral.com/huawei-launches-ads-4-0-and-harmonyos-cockpit-5-driving-solutions/, April 2025. Launch coverage of Huawei's WEWA end-to-end architecture. Accessed 2026-05-03.

Xiaosong Jia, Zhenjie Yang, Qifeng Li, Zhiyuan Zhang, and Junchi Yan. Bench2drive: Towards multi-ability benchmarking of closed-loop end-to-end autonomous driving. In *NeurIPS 2024 Datasets and Benchmarks Track*, 2024.

Keyur Joshi, Chiao Hsieh, Sayan Mitra, and Sasa Misailovic. Gas: Generating fast & accurate surrogate models for simulations of autonomous vehicle systems. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, pages 260–271. IEEE, 2024.

Seulbae Kim, Major Liu, Junghwan" John" Rhee, Yuseok Jeon, Yonghwi Kwon, and Chung Hwan Kim. Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1753–1767, 2022.

Shenghao Lin, Fansong Chen, Laile Xi, Gaosheng Wang, Rongrong Xi, Yuyan Sun, and Hongsong Zhu. Tm-fuzzer: fuzzing autonomous driving systems through traffic management. *Automated Software Engineering*, 31(2):61, 2024.

Long Nguyen, Micha Fauth, Bernhard Jaeger, Daniel Dauner, Maximilian Igl, Andreas Geiger, and Kashyap Chitta. Lead: Minimizing learner-expert asymmetry in end-to-end driving. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

NHTSA. National highway traffic safety administration: Office of defects investigation, preliminary evaluation PE25012: Tesla full self-driving traffic safety violations. https://static.nhtsa.gov/odi/inv/2025/INOA-PE25012-19171.pdf, October 2025. Accessed 2026-05-03.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. URL http://jmlr.org/papers/v12/pedregosa11a.html.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

Reuters. Waymo recalls 1,200 self-driving vehicles in US after minor collisions. https://www.reuters.com/business/autos-transportation/alphabets-waymo-recalls-over-1200-vehicles-after-collisions-with-roadway-2025-05-1 May 2025. Accessed 2026-05-03.

Rivian. Rivian autonomy & AI day. https://stories.rivian.com/rivian-autonomy-ai-day, December 2025. Rivian Stories. Accessed: 2026-05-06.

SciPy Developers. *SciPy v1.15.2 Manual: scipy.stats._result_classes.BinomTestResult.proportion_ci*, 2025. URL https://docs.scipy.org/doc/scipy-1.15.2/reference/generated/scipy.stats._result_classes.BinomTestResult.proportion_ci.html. Accessed: 2026-09-25.

Shuhan Tan, Kelvin Wong, Shenlong Wang, Sivabalan Manivasagam, Mengye Ren, and Raquel Urtasun. Scenegen: Learning to generate realistic traffic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 892–901, 2021.

TechCrunch. Waymo recalls 1,200 robotaxis following low-speed collisions with gates and chains. https://techcrunch.com/2025/05/14/waymo-recalls-1200-robotaxis-following-low-speed-collisions-with-gates-and-chains, May 2025. Accessed 2026-05-03.

Eric Thorn, Shawn Kimmel, and Michelle Chaka. A framework for automated driving system testable cases and scenarios. Technical Report DOT HS 812 623, National Highway Traffic Safety Administration, Washington, DC, 2018. URL https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13882-automateddrivingsystems_092618_v1a_tag.pdf.

Hazem Torfah, Carol Xie, Sebastian Junges, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. Learning monitorable operational design domains for assured autonomy. In *Proceedings of the International Symposium on Automated Technology for Verification and Analysis (ATVA)*, October 2022.

Eric Vin, Shun Kashiwa, Matthew Rhea, Daniel J Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. 3d environment modeling for falsification and beyond with scenic 3.0. In *International Conference on Computer Aided Verification*, pages 253–265. Springer, 2023.

Jingkang Wang, Ava Pun, James Tu, Sivabalan Manivasagam, Abbas Sadat, Sergio Casas, Mengye Ren, and Raquel Urtasun. Advsim: Generating safety-critical scenarios for self-driving vehicles. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9909–9918, 2021.

Waymo. Demonstrably safe AI for autonomous driving. https://waymo.com/blog/2025/12/demonstrably-safe-ai-for-autonomous-driving, December 2025. Describes the Waymo Foundation Model 'Think Fast, Think Slow' architecture (Sensor Fusion Encoder + Driving VLM). Accessed 2026-05-03.

Wayve. Wayve AI Driver: Redefining automated driving technology. https://wayve.ai/product/wayve-ai-driver/, 2025. Accessed 2026-05-03.

Linrui Zhang, Zhenghao Peng, Quanyi Li, and Bolei Zhou. Cat: Closed-loop adversarial training for safe end-to-end driving. In *Conference on Robot Learning*, pages 2357–2372. PMLR, 2023.

A SCENARIO PARAMETER VALUES AND HARDWARE SETUP

Table 2: Space of scenario parameters

Scenario Parameter	**Values**	**Canonical Safe Values**	Δ	ϵ
Fog Density (FD)	0.0 to 100.0	0.0 to 20.0	20.0	15.0
Cloudiness (CD)	0.0 to 100.0	0.0 to 20.0	20.0	15.0
Precipitation (PC)	0.0 to 100.0	0.0 to 20.0	20.0	15.0
Sun altitude angle (SAL)	-90.0° to +90.0°	+30° to +60°	20.0	15.0
Sun azimuth angle (SAA)	0.0° to 360.0°	+0° to +60°	30	15.0
Pedestrian speed (NPC)	1.5 to 10.0	4.5 to 10.0	1.0	0.5
Blocking length (BL)	3.0 to 12.0	3.0 to 6.0	1.5	1.0
Oncoming traffic frequency (OTR)	15.0 to 90.0	70.0 to 90.0	5.0	2.0
Cyclist offset (CO)	0.0 to 1.1	0.9 to 1.1	0.2	0.1
Cyclist speed (CYS)	6.0 to 25.0	20.0 to 25.0	2.5	1.5
Reaction time (RT)	0.5 to 2.4	2.0 to 2.4	0.37	0.3
Cut-in speed (CS)	2.9 to 40.0	25.0 to 40.0	7.4	6.0
Minimum trigger distance (MD)	1.9 to 12.0	8.0 to 12.0	2.1	2.0
Walkers offset (PDO)	0.0 to 2.0	1.6 to 2.0	0.4	0.3
Walkers speed (WS)	0.5 to 3.0	0.5 to 1.2	0.5	0.4
Distance to walkers (DTW)	30.0 to 150.0	120.0 to 150.0	24.0	18

The value of sun altitude angle ranges from -90° (nighttime) to +90° (daytime). The value of the sun azimuth angle parameter is the compass direction of the sun in the horizontal plane. The value of Pedestrian speed is in meters per second.

Hardware Platform. We perform the experiments on devices with: AMD Ryzen Threadripper PRO 3975WX 32-Cores, 512 GB RAM, 1 NVIDIA RTX A5000 GPU, 1 NVIDIA A6000 GPU and 1 NVIDIA A10G GPU. To report runtimes on a consistent scale, we scale measurements taken on the A5000 and A10G by the TFLOPS ratio between them and the A6000 GPU, and report the measurements on A6000 as is.

B CASE STUDIES

Safe/Unsafe Configuration Pairs. An artifact of ODIN's synthesis procedure is that it produces evidence of maximality from its attempts to expand the safe region. The algorithm stops the expansion in a region when it discovers an unsafe configuration in that expansion. This means some tested configurations are *just outside* the safe region, with safe configurations nearby (inside the safe region). Below, we present a few configuration pairs that differ meaningfully in only one scenario parameter, but only one of them was unsafe, found using ODIN.



(a) Safe scenario (b) Unsafe scenario

Figure 5: In this instance of a pedestrian crossing a neighboring lane (scenario in row 2 of Table 1), configuration (a) is safe, but (b) results in a collision. The main difference between the two is the speed of the pedestrian. In (a), the pedestrian is moving at 5.3 m/s, fast enough to quickly cross both lanes. But in (b), the pedestrian moving at 4.3 m/s has just reached the middle of the road and will end up in a collision with the ego vehicle.



(a) Safe scenario (b) Unsafe scenario

Figure 6: In this instance of pedestrian crossing the road immediately after the ego vehicle turns at the intersection (scenario in row 3 of Table 1), configuration (a) is safe, but (b) results in a collision. The key difference between the two is the value of Sun altitude angle, which is 30° for (a), and 7° for (b). In this scenario, the camera occasionally gets occluded by drops of precipitation.



(a) Safe scenario (b) Unsafe scenario

Figure 7: In this instance of a pedestrian crossing from an occluded sidewalk (scenario in row 1 of Table 1), configuration (a) is safe, but (b) results in a collision. The main difference between the two is the speed of the pedestrian. In (a), the pedestrian is much faster and has almost crossed the lane when the ego vehicle is further away, unlike in (b).

It should be noted that these are just a few instances of the safe/unsafe configuration pairs obtained from the ODIN's learning procedure. More such pairs can be sampled, where a difference in other scenario parameters causes a collision.

C ODIN ALGORITHMS

We present the algorithms of ODIN below:

Algorithm 1 ODIN: Safe scenario parameter region synthesis

```
1:  procedure SAFESCENARIOPARAMSYNTH(N,d,K,Δ,ϵ)
2:      X ← sample_for(N)                          ▷ contains positive & negative points (P,N)
3:      Y ← SIMULATOR(X)                           ▷ positive points P are anchors
4:      done,M ← False,false
5:      while ¬done do
6:          M ← LEARNER(X,Y,d)
7:          cexs,pred_labels ← PREDICATETESTINGENGINE(M,K,Δ,ϵ)
8:          test_labels ← SIMULATOR(cexs)
9:          if pred_labels = test_labels then
10:             done ← True
11:         else
12:             X,Y ← CONFLICTRESOLVER(X :: cexs,Y :: test_labels,Δ)
13:         end if
14:     end while
15:     return M
16: end procedure
```

Algorithm 2 ODIN: Predicate Testing Engine

```
1:  procedure PREDICATETESTINGENGINE(M,K,Δ,ϵ)
2:      cexs,labels ← {},{}
3:      for p ∈ M do                               ▷ iterate over disjuncts in M
4:          cex ← sample((p),K)                    ▷ Safety check
5:          cexs.add(cex)
6:          rlx_del,rlx_eps ← Expand(p,Δ),Expand(p,ϵ)   ▷ expanded conjunct
7:          for (r_d,r_e) in (rlx_del,rlx_eps) do   ▷ pairwise matched atomic predicates
8:              cex ← sample(r_d ∧ ¬r_e,K)
9:              cexs.add(cex)
10:         end for
11:     end for
12:     for c ∈ cexs do
13:         labels.add(M(c))
14:     end for
15:     return (cexs,labels)
16: end procedure
```

Sampling for maximality may lead to slow expansion if only the configurations close to the boundary of M are sampled. (all the parameter-specific expansion values in ϵ are less than those in Δ). We call this space the Δ-ϵ region for M. Intuitively, the maximality engine samples configurations *at least* ϵ outside of M, and *at most* Δ outside of M (lines 6- 9, Algorithm 2).

ODIN's conflict resolver (Algorithm 3) uses a discretization-based definition of *closeness*, it declares two configurations as being close if they fall into the same buckets for all scenario parameters (line 4, Algorithm 3). We use $\oslash$ to indicate the pointwise division in the algorithm.

Algorithm 3 ODIN: Conflict Resolver

procedure CONFLICTRESOLVER(X,Y,Δ)
  $resolvedX$,$resolvedY \leftarrow [\,],[\,]$
  for $(x,y) \in (X,Y)$ **do** ▷ Pairwise matched lists
    if $y = \texttt{true} \wedge \exists (x',y') \in (X,Y).\ \lfloor x \oslash \Delta \rfloor = \lfloor x' \oslash \Delta \rfloor \wedge y' = false$ **then**
      $resolvedX.\text{append}(x)$
      $resolvedY.\text{append}(false)$
    else
      $resolvedX.\text{append}(x)$
      $resolvedY.\text{append}(y)$
    end if
  end for
  return $(resolvedX, resolvedY)$
end procedure

D DISCUSSION OF EVALUATION

Learning Algorithm Time. Unsurprisingly, the algorithm takes longer for trees of greater depths, since the number of configurations to test increases. Most of the time is spent in running CARLA and the driving model for inference. Only a small fraction of the total time is taken up by the decision tree learner, and the configuration sampling procedure. Although the times increase with more configurations to sample, there is some variation in time taken across scenarios. This can be explained by the diversity in the workloads (simulating a scenario with fewer time steps takes lesser compute, similarly simulating different weathers take different amounts of processing).