Towards Scalable Coverage-Based Testing of Autonomous Vehicles

Anonymous Author(s)
Affiliation
Address
email

Abstract: To safely deploy autonomous vehicles(AVs) in the real world, developers must understand the conditions in which the system is safe, unsafe, or the outcome is unknown [1]. To do this in a safe and scalable manner, AVs are often tested in simulation on parameterized scenarios. In this context, it's important to build a testing framework that partitions the scenario parameter space into safe, unsafe, and unknown regions. Existing approaches rely on discretizing continuous parameter spaces into bins, which scales poorly to high-dimensional spaces and cannot describe regions with arbitrary shape. In this work, we introduce a problem formulation which avoids discretization—by modeling the probability of meeting safety requirements everywhere, the parameter space can be paritioned using a probability threshold. Based on our formulation, we propose a scalable testing framework GUARD which leverages Gaussian Processes to model probability and levelset algorithms to efficiently generate tests. Moreover, we introduce a set of novel evaluation metrics for coverage-based testing frameworks to capture the key objectives of testing. In our evaluation suite of diverse high-dimensional scenarios, GUARD significantly outperforms existing approaches. By proposing an efficient, accurate, and scalable testing framework, our work is a step towards safely deploying autonomous vehicles at scale.

Keywords: Testing, Self-Driving, Coverage

1 Introduction

Autonomous vehicles (AVs) will soon become a staple in ground transportation—interacting with billions of people everyday. At this scale, AVs can drastically reduce accidents, relieve traffic congestion, and provide mobility for those who cannot drive. In order to realize this future, developers must first ask the question: *"Is the AV safe enough to be deployed in the real world?"* To understand how safe the AV is, industry-standard safety guidelines [1, 2] require a comprehensive understanding of which scenarios are known to be safe or unsafe and which scenarios have unknown outcomes. Towards this goal, it's important to build a testing framework which *covers* the wide range of real-world scenarios to classify them as safe, unsafe, or unknown.

To cover the wide range of real-world scenarios in a scalable manner, the AV industry often builds testing frameworks in simulation [3, 4, 5] where long-tail events can be synthesized. A popular strategy to describe real world events in simulation involves designing parameterized scenarios [6]. These scenarios describe semantics of the environment (e.g. truck merging from on-ramp) and its parameters specify low-level characteristics (e.g. velocity of traffic participants). Each parameter configuration then corresponds to a concrete test which can be executed in simulation to determine if the AV behaves in the intended manner. This is typically captured as a binary pass or fail determined by regulatory demand [7]. For example, an AV could fail if it violates a safety distance threshold.

However, directly covering all variations in a scenario's parameter space can be infeasible. Continuous parameters imply infinitely many variations and testing frameworks can only execute a finite number of concrete tests which is limited by computation budget. As a result, to cover the parameter space the testing framework must leverage observed test results to estimate if the AV will pass or

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

fail on unseen test configurations. Furthermore, the testing framework must also efficiently choose concrete tests to execute to maximize coverage and accurately estimate AV performance.

Existing approaches rely heavily on *discretization* to reduce the infinite continuous parameter space into discrete bins—making the assumption that tests in the same bin yield the same result. While effective for simple 2-dimensional scenarios [8, 9], they scale poorly to high dimensional scenarios due to exponentially growing number of bins. Alternatively, another common approach involves generating adversarial examples [10, 11, 12] to discover failures. While revealing failure cases is useful, these methods forgo covering the parameter space and cannot provide an understanding of AV performance across the parameter space.

In this work, we introduce a formulation for coverage-based testing which avoids discretization altogether. To start, we can leverage observed test results to model the probability of passing or failing across the *entire* parameter space. Intuitively, if the AV easily passed a particular test, it can pass similar tests with high probability. Moreover, for an unseen test with little information from similar tests, the pass/fail outcome should be ambiguous. By modeling the probability everywhere, it follows that a probability threshold can partition the parameter space into 1) pass regions, 2) failure regions, and 3) uncertain regions, which is the output prescribed by standard safety guidelines [1, 2].

Based on our formulation, we propose our testing framework GUARD. Specifically, we leverage Gaussian Processes (GPs) to model the probability of passing and failing. To efficiently sample concrete tests, we adopt levelset algorithms which adaptively generates tests that balance exploration (of high uncertainty regions) and exploitation (by sampling near the pass/fail boundary). Moreover, GUARD leverages online GP kernel learning, allowing it to scale to different parameterized scenarios with minimal hyperparameter tuning.

To evaluate testing frameworks, we propose a suite of novel evaluation metrics which capture the key objectives of testing: achieving high coverage, evaluating safety of the AV, and exposing failure cases for autonomy development.Under these metrics, GUARD is able to outperform existing testing frameworks across a diverse set of complex and high-dimensional driving scenarios. Moreover, we find that by avoiding discretization, our approach scales much more efficiently as the dimensionality of test scenarios grow. Finally, GUARD can also benchmark different iterations of autonomy models while identifying specific examples of scenarios where regression occurs.

2 Related Work

2.1 Coverage-Based Testing

Autonomous vehicles are often tested in simulation using parameterized scenarios [6, 13, 14] and evaluated based on safety regulation [7] to output a pass or fail. Each parameter configuration corresponds to a concrete test and coverage-based testing frameworks execute concrete tests to cover the scenario parameter space. Since there can be an infinite amount of scenario variations due to continuous parameters, existing works *discretize* the parameter space into bins—letting one concrete test represent each bin and assuming all pass/fail outcomes in a bin are identical. This discretization introduces error in understanding where the AV passes and fails, which grows with the bin size.

Exhaustively testing all parameter bins [14, 15] is a widely adopted approach in the industry. Since the parameter space is exponentially large, this approache suffer from coarse discretization resolutions when testing high-dimensional scenarios. To increase sample efficiency, t-way testing [16] samples points such that for every subset of t parameters, every parameter combination is covered at least once. This affords higher discretization resolutions, but leads to inaccuracies stemming from its assumption that tests sharing a small subset of t parameters yield the same pass/fail result. More recently, [8] leverage levelset estimation algorithms to classify the pass/fail outcomes more efficiently. On the other hand, [9] leverages adaptive discretization resolutions. While these works demonstrate promising results on 2-dimensional scenarios, they are still difficult to scale to high-dimensional parameter spaces.

2.2 Adversarial Example Generation

A widely studied approach for test generation in parameterized scenarios is through adversarial example generation, or generating worst case failures [17, 18, 19, 20, 21, 22]. This is typically done by optimizing scenario parameters with a black-box optimization algorithm (e.g. Bayeisan Opti-

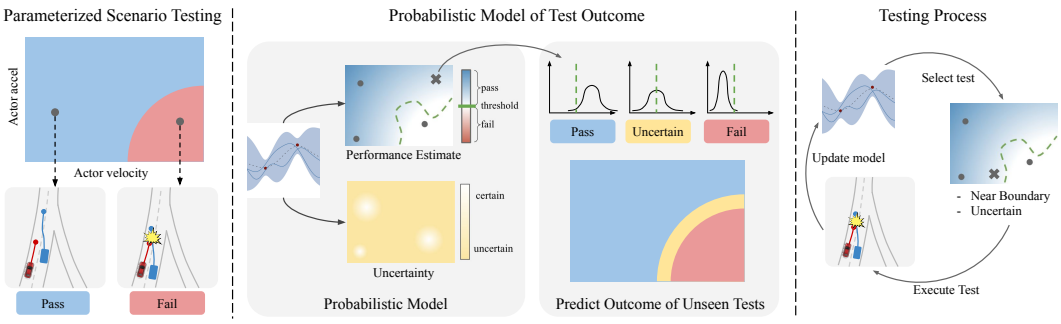

Figure 1: **Left:** Parameterized scenario testing. **Middle:** Using probabilistic model to estimate pass/fail outcomes across parameter space. **Right:** Testing process—sampling and executing informative tests and updating model.

mization [10, 23, 24, 25, 26]) to maximally degrade safety metrics. These methods terminate upon discovering an adversarial scenario and do not cover the parameter space. On the other hand, [27] showed that BO with the GP-LCB acquisition function can validate that an AV passes all scenario variations if the LCB is positive everywhere and a failure isn't discovered. This achieves coverage but only in the case where the AV is perfect. Acquisition functions for adversarial example generation can also be used to sample tests in our framework. However, this oversamples severe failures, whereas oversampling on the pass/fail boundary is more efficient for coverage-based testing.

3 Scalable Coverage-Based Testing

We first present our problem formulation which leverages probabilistic models to partition the scenario parameter space into pass, fail, and unknown regions. Following this formulation, we use Gaussian Processes (GPs) as our probabilistic model. Then, we discuss adopting levelset algorithms to efficiently sample concrete tests. Finally, we leverage learnable kernel parameters to adaptively construct accurate GP priors. As result, our framework can perform well on a wide range of scenarios with minimal hyperparamter tuning. An overview of our method can be found in Figure 1.

3.1 Problem Formulation

The AV software stack is often tested on parameterized scenarios commonly known as *logical scenarios* [13, 6], which are the standard in the self-driving industry. Each logical scenario features a set of d configurable parameters $\boldsymbol{\theta} \in \mathbb{R}^d$, where each specific configuration of the parameters results in a concrete test. The scenario parameters are bounded [13, 15], i.e., $\theta_i \in [a_i, b_i]$, making the entire parameter search space a closed set $\boldsymbol{\Theta}$ in $\mathbb{R}^d$. Executing a test in simulation outputs continuous measures of safety or other performance metrics, for which we denote an individual measure as $f^*(\boldsymbol{\theta}; \mathcal{A}) \in \mathbb{R}$. $\mathcal{A}$ denotes the autonomy software under test, which we omit to use $f^*(\boldsymbol{\theta})$ for brevity. In compliance with regulatory demand [7], a binary pass or fail $y = \mathbf{1}[f^*(\boldsymbol{\theta}) \geq \gamma]$ is computed using a threshold γ. It follows that we can denote regions in the parameter space $\boldsymbol{\Theta}$ where the system passes sand fails as

$$\mathcal{P}^* = \{\boldsymbol{\theta} \in \Theta, f^*(\boldsymbol{\theta}) \geq \gamma\}. \tag{1}$$
$$\mathcal{F}^* = \{\boldsymbol{\theta} \in \Theta, f^*(\boldsymbol{\theta}) < \gamma\}. \tag{2}$$

Then, the testing process can be thought of as estimating $\mathcal{P} \approx \mathcal{P}^*$, $\mathcal{F} \approx \mathcal{F}^*$ by executing concrete tests, which can be thought of as querying test points $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N\}$ and observing the ground truth values $\{f^*(\boldsymbol{\theta}_1), \ldots, f^*(\boldsymbol{\theta}_N)\}$. Estimating pass/fail across a continuous parameter domain using a finite set of test points inevitably leads to estimation errors, especially if the number of concrete tests are limited (e.g. due to testing resource constraints). At the same time, estimation errors and false positive passes in particular can be detrimental to safety.

Thus, we design an uncertainty-aware formulation where the framework quantifies the confidence of its estimation. This allows the framework to say it is uncertain instead of predicting a pass/fail outcome with insufficient information. Specifically, we compute the probability that the system will

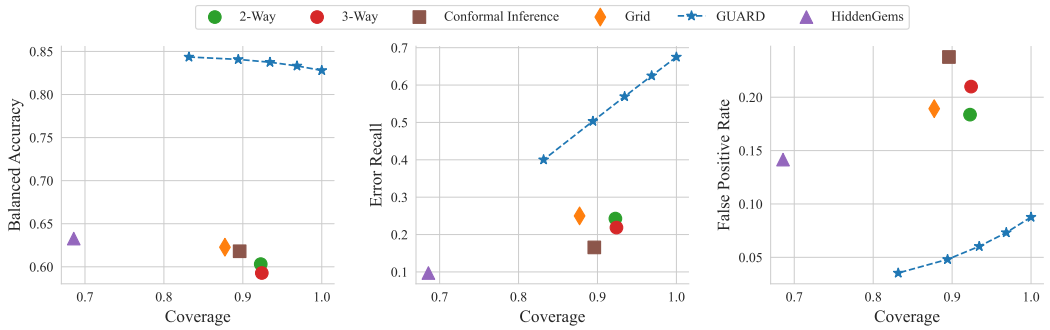

Figure 2: GUARD significantly outperforms other testing frameworks. GUARD also a sweepable confidence threshold to a trade-off between coverage and the other metrics.

pass or fail at any point in the parameter space. To achieve this, we model $f^*(\boldsymbol{\theta})$ with a random variable $f(\boldsymbol{\theta})$. Under this random variable, the estimated pass and fail regions can be defined as

$$\begin{aligned} \mathcal{P} &= \{\boldsymbol{\theta} \in \boldsymbol{\Theta}, P(f(\boldsymbol{\theta}) \geq \gamma) \geq \alpha\}. \\ \mathcal{F} &= \{\boldsymbol{\theta} \in \boldsymbol{\Theta}, P(f(\boldsymbol{\theta}) < \gamma) \geq \alpha\}. \end{aligned} \tag{3}$$

with respect to confidence threshold α. It follows that the unknown region can be obtained as $\mathcal{U} = \boldsymbol{\Theta} - \mathcal{P} - \mathcal{F}$, where more information is required to make a prediction. Note that the size of $\mathcal{P}$ and $\mathcal{F}$ increase monotonically as α decreases. The configurable confidence threshold α allows the testing framework to control the tradeoff between the quality and quantity of predictions.

3.2 Gaussian Process

As outlined in Equation 3, our formulation requires a probabilistic estimate $f(\boldsymbol{\theta})$ for the ground truth test function $f^*(\boldsymbol{\theta})$. We adopt GPs as they perform well in low-data regimes, making them suitable in our setting as the number of concrete tests are limited. Additionally, GPs are non-parametric and make less assumptions about the function being modeled, which is important in allowing our testing framework to scale to a wide variety of logical scenarios.

A GP is a collection of random variables in which every subset is assumed to distributed from a multivariate Gaussian. The observed test samples $X = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N]$, $Y = [f^*(\boldsymbol{\theta}_1), \ldots, f^*(\boldsymbol{\theta}_N)]$ then become observations of GP random variables. Conditioned on these observations, the posterior distribution at a new datapoint $\boldsymbol{\theta}$ can be computed, yielding the posterior mean and variance

$$\mu(\boldsymbol{\theta}) = k(\boldsymbol{\theta}, X)^T (K + \epsilon^2 I)^{-1} Y \tag{4}$$

$$\sigma^2(\boldsymbol{\theta}) = k(\boldsymbol{\theta}, \boldsymbol{\theta}) - k(\boldsymbol{\theta}, X)^T (K + \epsilon^2 I)^{-1} k(\boldsymbol{\theta}, X). \tag{5}$$

Here $k(\cdot, \cdot)$ is the kernel function which provides a similarity measure between two GP variables, $k(\boldsymbol{\theta}, X) = [k(\boldsymbol{\theta}, \boldsymbol{\theta}_i), \ldots, k(\boldsymbol{\theta}, \boldsymbol{\theta}_N)]$, $K_{ij} = k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$, and ϵ models noise in the observed values Y.

Under the posterior distribution $f(\boldsymbol{\theta})$, the probability of a observing a value above or below a certain threshold can be computed. Specifically, the probability of observing a value greater than the pass-fail threshold γ is

$$P(f(\boldsymbol{\theta}) \geq \gamma) = \Phi\left(\frac{\mu(\boldsymbol{\theta}) - \gamma}{\sigma(\boldsymbol{\theta})}\right). \tag{6}$$

where Φ is the CDF of the normal distribution. The probability of observing a value under γ can be computed in a similar fashion. An illustration of using the probability to predict pass and fail region can be seen in Figure 1.

3.3 Test Generation

Note that precisely modelling f^* when the output is far from γ adds little value since it won't affect the pass or fail outcome. In contrast, it's important to accurately model regions near the γ-levelset. Hence, we leverage levelset algorithms to efficiently upsample points near the boundary. Specifically, we adopt *Straddle* [28] as it directly integrates GPs, and alternative GP-levelset algorithms

Method	Coverage(%)	Bal. Acc(%)	Pos Acc(%)	Neg Acc(%)	Err. Recall(%)	FPR(%)
Grid	87.7 ± 0.4	62.3 ± 0.1	97.4 ± 0.03	27.2 ± 0.2	25.0 ± 0.3	18.9 ± 0.2
2-way [16]	92.2 ± 0.2	60.3 ± 0.6	96.0 ± 0.1	24.6 ± 1.2	24.3 ± 1.2	18.2 ± 0.4
3-way [16]	92.4 ± 1.0	59.7 ± 1.7	96.2 ± 0.3	23.2 ± 3.7	22.5 ± 3.6	20.9 ± 1.3
HiddenGems [8]	72.8 ± 0.6	66.2 ± 2.2	98.3 ± 0.1	33.7 ± 4.4	14.8 ± 2.4	16.0 ± 0.4
Conformal Inference [9]	89.6 ± 0.4	61.8 ± 0.8	97.0 ± 0.7	26.7 ± 1.4	16.6 ± 1.5	23.8 ± 0.9
GUARD	$\mathbf{94.3 \pm 0.9}$	$\mathbf{84.2 \pm 5.3}$	$\mathbf{99.2 \pm 0.2}$	$\mathbf{70.3 \pm 10.6}$	$\mathbf{58.9 \pm 5.3}$	$\mathbf{5.89 \pm 3.1}$

Table 1: Benchmarking GUARD against baselines

operate on finite input domains [29, 8, 30]. *Straddle* iteratively queries test points based on an exploration incentive promoting points with high variance and an exploitation incentive promoting points close to the γ-levelset. Specifically, these two incentives are captured by the acquisition function

$$h_t(\boldsymbol{\theta}) = \beta\sigma_t(\boldsymbol{\theta}) - |\mu_t(\boldsymbol{\theta}) - \gamma|. \tag{7}$$

where β is a weighting coefficient balancing the first exploration term and the second exploitation term. At each iteration, we solve a lightweight maximization problem to find the next query point:

$$\boldsymbol{\theta}_t = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, h_t(\boldsymbol{\theta}). \tag{8}$$

We start by sampling P initial random points from $\boldsymbol{\Theta}$. Then since $h_t(\boldsymbol{\theta})$ is differentiable with respect to $\boldsymbol{\theta}$, we leverage gradient based optimization to refine the top Q candidates from the initial samples. This optimization is inexpensive since the cost of running the GP model and performing backpropagation to evaluate $\mu(\boldsymbol{\theta}), \sigma(\boldsymbol{\theta})$ is negligible compared to running simulation to evaluate $f^*(\boldsymbol{\theta})$. Finally, the concrete test corresponding to $\boldsymbol{\theta}_t$ is then executed in simulation to obtain $f^*(\boldsymbol{\theta}_t)$. The observation $(\boldsymbol{\theta}_t, f^*(\boldsymbol{\theta}_t))$ is then added to the GP model to update the GP posterior.

3.4 Automatic Kernel Tuning

In addition to observed test points, the posterior distribution outlined in Equation 4 and 5 also depend heavily on the kernel function $k(\cdot,\cdot)$. Thus, the estimates $\mathcal{P}$ and $\mathcal{F}$ are sensitive to the kernel function parameters which we denote as $\boldsymbol{\kappa}$. At the same time, tuning kernel parameters can be tedious and time consuming [8]. Moreover, in our task, each individual scenario has different parameters with varying scales and effects on the final output, requiring different kernels to accurately model different scenarios. This makes manually tuning kernels infeasible for large scale testing.

To circumvent this issue, we first normalize the scenario parameter space $\boldsymbol{\Theta}$ to the unit hypercube $[0,1]^d$ to address parameters with different scales. Furthermore, instead of manually tuned and fixed kernel parameters, we optimize the kernel parameters to maximize the marginal likelihood of the observations. In particular, at iteration t, we update the kernel parameters $\boldsymbol{\kappa}$ towards

$$\boldsymbol{\kappa}_t = \underset{\boldsymbol{\kappa}}{\operatorname{argmax}}\, P(f(\boldsymbol{\theta}_1)\dots f(\boldsymbol{\theta}_t) \mid \boldsymbol{\theta}_1 \dots \boldsymbol{\theta}_t; \boldsymbol{\kappa}). \tag{9}$$

Similar to Equation 8, the marginal likelihood is differentiable with respect to the kernel parameters $\boldsymbol{\kappa}_t$, which allows us to perform gradient based optimization. We perform this optimization every K iterations. The kernel learning process can be prone to overfitting to a small initial batch of observations. Therefore, we perform an initial sampling step where we query M tests using only the variance term $\sigma_t(\boldsymbol{\theta})$ of the acquisition function outlined in Equation 7.

4 Experiments

In this section, we first introduce the test scenarios, evaluation metrics, and relevant baselines. Then, we present a series of experiments benchmarking GUARD against the baselines and ablating our design choices. Finally, we discuss how GUARD can be used in practice, identifying concrete tests where the autonomous vehicle is safe, unsafe, or unknown. Moreover, we can compare the safety between different autonomous systems and reveal specific cases of regression. Additional experiments are also included in the supplementary materials.

4.1 Experimental Setup

Test Scenarios: We use a suite of 10 test scenarios each with $d = 5$ parameters. These scenarios capture a diverse set of common safety-critical driving situations. Detailed descriptions of these test

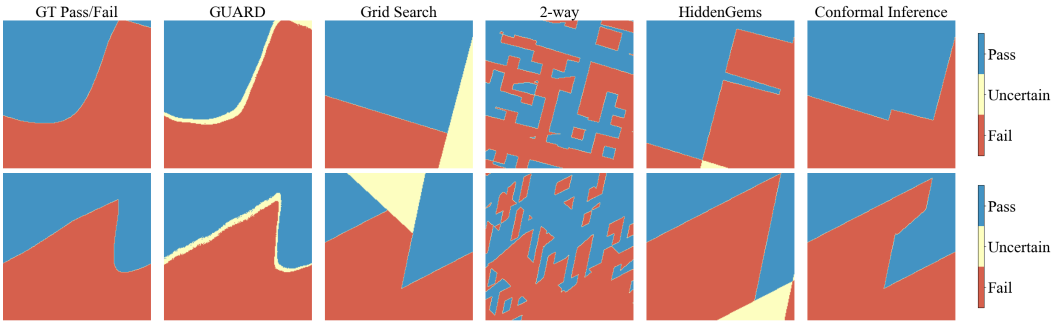

Figure 3: 2D slice of 5D parameter space. GUARD performs better by avoiding discretization.

scenarios and their parameterization and bounds can be found in the supplementary material. We evaluate the PLT motion planner [31], using minimum distance(m) to collision as the safety metric and choosing the safety threshold as $\gamma = 1.0$. We use $N = 1000$ test samples for experiments unless otherwise specified. In the following experiments, we repeat 5 trials for each test scenario to collect mean and standard deviation of metrics. And we report the average mean and standard deviation across the 10 scenarios. To support large scale experimentation, we construct an offline dataset by evaluating each scenario space over a fine grid and construct an approximation via linear interpolation.

Metrics: To evaluate performance, we propose several metrics which capture important aspects of testing for safety and autonomy development.

- *Coverage* is the percentage of the search space $\mathbf{\Theta}$ that is covered by the estimates $\mathcal{P}$ and $\mathcal{F}$:

$$\text{coverage} = \frac{|\mathcal{P} \cup \mathcal{F}|}{|\Theta|}. \tag{10}$$

- *Balanced Accuracy* evaluates how accurate the estimates $\mathcal{P}$ and $\mathcal{F}$ are and also addresses class imbalance due to the fact that the scenario space is dominated by passes:

$$\text{balanced acc} = \underbrace{\frac{|\mathcal{P} \cap \mathcal{P}^*|}{|\mathcal{P}^*| \cap (|\mathcal{P} \cup \mathcal{F}|)}}_{\text{positive acc}} + \underbrace{\frac{|\mathcal{F} \cap \mathcal{F}^*|}{|\mathcal{F}^*| \cap (|\mathcal{P} \cup \mathcal{F}|)}}_{\text{negative acc}} \tag{11}$$

- *Error Recall* measures the percentage of the failures the testing framework can recall, which is very useful for autonomy development. This computed as:

$$\text{error recall} = \frac{|\mathcal{F} \cap \mathcal{F}^*|}{|\mathcal{F}^*|}. \tag{12}$$

- *False Positive Rate* is the percentage of predicted passes that are incorrect, which is crucial to safety as incorrectly predicting the system to be safe can lead to accidents. This is computed as:

$$\text{false positive rate} = \frac{|\mathcal{P} \cap \mathcal{F}^*|}{|\mathcal{P}|}. \tag{13}$$

Note that some of these quantities such as $|\mathcal{P}^*|$ cannot be computed exactly. Therefore, we randomly sample $20,000$ points in the scenario parameter space to estimate the terms in the above equations.

Baselines: We compare GUARD with several baselines outlined in Section 2. These include traditional testing methodologies [16, 14, 15] and works proposed in the literature [8, 9].

- *Grid Search* [15, 14] discretizes continuous parameters, and uses one discrete test point to represent each bin. Given budget of N test samples which cannot cover all bins, we sample a subset of the bins at random. In our scenarios with $d = 5$ dimensions, we use discretization resolution 4.
- *T-way testing* [16] leverages the assumption that most faults are caused by interactions of at most t parameters. This method draws test samples such that for every t-parameter subset, all possible combinations of parameter values can be found in at least one test. In our experiments, we consider $t = 2, 3$ and choose a discretization resolution of 30 for $t = 2$ and 12 for $t = 3$.

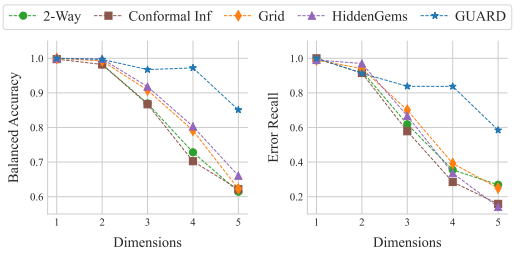

Figure 4: Scaling with scenario dimensionality.

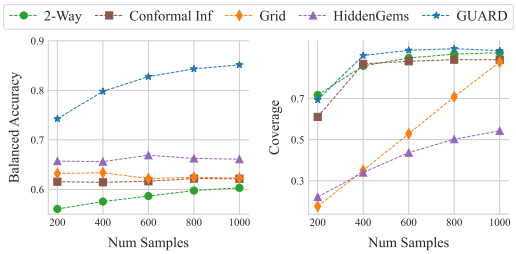

Figure 5: Scaling with testing budget.

- *HiddenGems* [8] employs levelset estimation algorithms to more efficiently infer pass/fail outcomes of the bins. The authors use a 33x33 discretization for a 2-D parameter space. However, this resolution becomes infeasibly expensive with $d = 5$ parameters. We adopt a resolution of 6 and further analyze the exponentially increasing runtime in the supplementary materials.
- *Conformal Inference* [9] uses conformal inference in each parameter bin to determine the outcome. Furthermore, this approach iteratively partitions the parameter space while adaptively selecting the bin sizes.

Implementation Details: The initial exploration batch defined in Section 3.4 has size $M = 200$. Our GP kernel is a product composition of separate Matern32 kernels [32] for each parameter dimension. The lengthscale is initialized to 0.02 and we initialize the kernel variance using the variance of the exploration batch. The kernel is fitted every 50 samples with 100 steps of optimization using Adam [33] with learning rate 0.01. Our acquisition function uses $\beta = 0.5$. During acquisition maximization, we randomly sample $P = 600$ candidates and perform gradient-based maximization on the top $Q = 30$ candidates using Adam with learning rate 0.01 for 30 steps.

4.2 Experiment Results

Testing Performance: We demonstrate the effectiveness of GUARD in Figure 2 where we plot test coverage versus balanced accuracy, error recall, and false positive rate (FPR). First, note that we can adjust the trade-off between coverage and the other metrics in GUARD by adjusting the confidence threshold α as outlined in Equation 3. This is useful in controlling how conservative we are in testing. Along the trade-off curve, GUARD significantly outperforms other testing frameworks across all metrics. We provide numerical results in Table 1 where we set $\alpha = 0.7$ for direct comparison.

To visually illustrate how GUARD is able achieve better metrics, we visualize the partition outputs in Figure 3. Here we use the visualization technique proposed in [34] to visualize a 2D slice of the 5D parameter search space. Specifically we sample one boundary point and multiple slices around the boundary point, choosing the slice which exhibits the highest variance. On each slice we visualize the ground truth value pass and regions as well as the predicted outcomes. GUARD is able to more accurately predict the outcome since it's not limited by any discretization.

Scalability: We evaluate GUARD against baselines on lower dimensional functions to analyze scalability with the number of scenario parameters. The 10 test scenarios have $5 - c$ parameters fixed to reduce the dimension to c. We increase the discretization resolution of baselines at lower resolutions, choosing the resolutions to maintain a similar amount of bins as in the 5-D scenarios. Figure 4 shows that baselines perform well for 1-D and 2-D functions where they can afford a high resolution. As dimensionality increases the baselines degrade significantly compared to GUARD.

Sample Efficiency: In Figure 5, we evaluate the models using a varying number of test samples $N = [200, 400, 600, 800, 1000]$ and show that GUARD outperforms the alternative methods with fewer test samples as well. With increasing testing budget, GUARD gradually increases in accuracy and coverage. Other approaches which rely on discretization grow in coverage. However, their accuracy is heavily limited by the discretization resolution and improves slowly or not at all.

Ablations: We ablate some design choices of GUARD in Table 2. First, without the initial exploration phase, there is a noticeable drop across the metrics. Next, we consider removing online kernel learning, parameter normalization, and both. Removing either leads to drops in performance while removing both leads to degenerate solutions which predict pass everywhere. This is expected as without normalization or learnable lengthscales, a fixed lengthscale cannot model parameters with different scales (i.e. velocity $\in [20, 30] m/s$, road curvature $\in [-0.002, 0.002]$). Finally, we ablate

Exp	Norm	Learn	Prod	Coverage(%)	Bal. Acc(%)	Pos. Acc(%)	Neg. Acc(%)	Err. Recall(%)	FPR (%)
✓	✓	✓	✓	**94.3** $\pm$ **0.9**	**84.2** $\pm$ **5.3**	99.2 $\pm$ 0.2	**70.3** $\pm$ **10.6**	**58.9** $\pm$ **5.3**	5.9 $\pm$ 3.1
—	✓	✓	✓	92.2 $\pm$ 1.9	82.4 $\pm$ 4.9	**99.4** $\pm$ **0.2**	65.4 $\pm$ 10.0	52.1 $\pm$ 8.8	7.5 $\pm$ 3.3
✓	—	✓	✓	93.2 $\pm$ 1.2	83.7 $\pm$ 2.3	99.0 $\pm$ 0.2	68.3 $\pm$ 4.6	56.3 $\pm$ 4.4	**5.8** $\pm$ **1.6**
✓	✓	—	✓	92.3 $\pm$ 0.8	84.1 $\pm$ 1.9	98.7 $\pm$ 0.2	69.7 $\pm$ 3.8	48.6 $\pm$ 3.9	6.2 $\pm$ 1.0
✓	—	—	✓	0.04 $\pm$ 0.03	–	–	–	–	–
✓	✓	✓	—	93.3 $\pm$ 1.9	81.4 $\pm$ 4.1	99.0 $\pm$ 0.2	63.8 $\pm$ 8.3	51.6 $\pm$ 8.1	7.3 $\pm$ 1.9

Table 2: Ablating design choices of GUARD

the choice of the GP kernel. As described in Section 4.1, GUARD uses a product of Matern32 kernels on each parameter because each parameter affects the function output differently. Using a single kernel which doesn't consider each parameter individually degrades performance as expected.

4.3 Using GUARD In Practice

Planner	Pass Rate	Fail Rate
Ref	69.8%	18.5%
Agg	66.9%	24.0%

Table 3: Comparing 2 planners.

In this section, we demonstrate how GUARD can be used to evaluate autonomy systems and aid development. We use PLT as a reference planner and introduce a variation adjusted to make the planner more *aggressive*. First, we evaluate both planners on our suite of 10 test scenarios and gather the aggregate pass and fail rates. As shown in Table 4.3, GUARD identifies the aggressive planner fails more scenarios and is less safe. If the aggressive planner is an upcoming iteration, then we would have prevented a regression in safety. Furthermore, we can dig deeper and triage the regressions by visualizing the pass/fail landscape for both planners which we show in Figure 6 below.

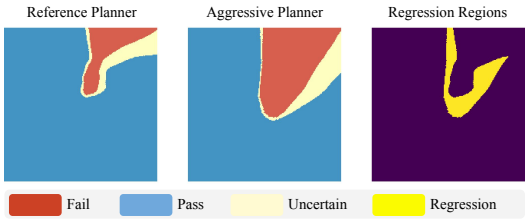

Figure 6: Identifying regression regions - reference planner passes, aggressive planner fails.

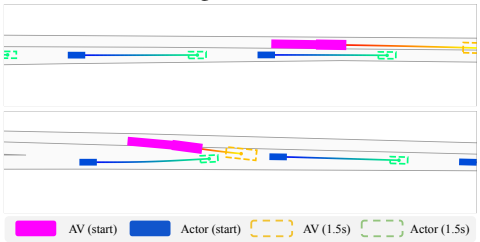

Figure 7: Regression example. **Top**: Reference. **Bottom**: aggressive planner collides.

To aid autonomy development, we can sample any point from the regression region to yield a concrete test. As an example, we show one example of such a test in Figure 7. In this test, the ego truck attempts to change into a neighboring lane while actors from an on-ramp are merging. The aggressive planner does not lane change safely which ultimately leads to a collision. Revealing all of these regressions in an automatic and scalable way is invaluable to autonomy development.

5 Limitations

This work limits the scope of parameterized scenario testing in a few ways. First, we assume the pass/fail result is thresholded on a single measure of safety, whereas it can be a combination of multiple measures. We also did not consider discrete scenario parameters which can be categorical (e.g. actor vehicle type) or numerical (e.g. number of lanes). Future works can incorporate these considerations to build a more complete testing framework. In addition, we propose a formulation built on probabilistic models and identify GPs as a suitable choice. Investigating alternative probabilistic models such as neural GPs [35] and probabilistic SVMs [36] is another exciting direction.

6 Conclusion

In this work, we study coverage-based testing of AVs in parameterzied scenarios and propose a formulation that eliminates discretization of the scenario parameter space. Based on this formulation we propose a testing framework GUARD. We thoroughly benchmark our framework on a suite of diverse and high-dimensional test scenarios and using metrics that capture the key objectives of testing. The experiments demonstrate that GUARD can evaluate autonomy systems much more efficiently and accurately, while also scaling better with scenario dimensionality. By proposing an efficient, and scalable testing framework, our work contributes to streamlined autonomy development, rigorous testing, and is ultimately a step towards safely deploying autonomous vehicles.

References

[1] Iso 21448:2022 road vehicles — safety of the intended functionality. https://www.iso.org/standard/77490.html, 2022.

[2] Iso 26262-1:2018 road vehicles — functional safety. https://www.iso.org/standard/68383.html, 2018.

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[4] P. Kaur, S. Taghavi, Z. Tian, and W. Shi. A survey on simulators for testing self-driving cars. In *2021 Fourth International Conference on Connected and Autonomous Driving (MetroCAD)*, pages 62–70. IEEE, 2021.

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[6] H. Weber, J. Bock, J. Klimke, C. Roesener, J. Hiller, R. Krajewski, A. Zlocki, and L. Eckstein. A framework for definition of logical scenarios for safety assurance of automated driving. *Traffic injury prevention*, 20(sup1):S65–S70, 2019.

[7] (grva) proposal for the 01 series of amendments to un regulation no. 157 (automated lane keeping systems).

[8] A. Petrov, C. Fang, K. M. Pham, Y. H. Eng, J. G. M. Fu, and S. D. Pendleton. Hiddengems: Efficient safety boundary detection with active learning. *arXiv preprint arXiv:2210.13956*, 2022.

[9] C. Fan, X. Qin, Y. Xia, A. Zutshi, and J. Deshmukh. Statistical verification of autonomous systems using surrogate models and conformal inference. *arXiv preprint arXiv:2004.00279*, 2020.

[10] J. Wang, A. Pun, J. Tu, S. Manivasagam, A. Sadat, S. Casas, M. Ren, and R. 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.

[11] A. Nonnengart, M. Klusch, and C. Müller. Crisgen: Constraint-based generation of critical scenarios for autonomous vehicles. In *International Symposium on Formal Methods*, pages 233–248. Springer, 2019.

[12] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi. Scalable end-to-end autonomous vehicle testing via rare-event simulation. *Advances in neural information processing systems*, 31, 2018.

[13] T. Menzel, G. Bagschik, and M. Maurer. Scenarios for development, test and validation of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1821–1827. IEEE, 2018.

[14] MS Windows NT kernel description. https://www.foretellix.com/resources_post/coverage-driven-verification-for-ensuring-av-and-adas-safety-gtc-2020/. Accessed: 2022-07-31.

[15] T. Zhao, E. Yurtsever, J. Paulson, and G. Rizzoni. Formal certification methods for automated vehicle safety assessment. *IEEE Transactions on Intelligent Vehicles*, 2022.

[16] D. R. Kuhn, R. N. Kacker, Y. Lei, et al. Practical combinatorial testing. *NIST special Publication*, 800(142):142, 2010.

[17] D. Rempe, J. Philion, L. J. Guibas, S. Fidler, and O. Litany. Generating useful accident-prone driving scenarios via a learned traffic prior. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17305–17315, 2022.

[18] C. Yan, W. Xu, and J. Liu. Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle. *Def Con*, 24(8):109, 2016.

[19] J. Tu, M. Ren, S. Manivasagam, M. Liang, B. Yang, R. Du, F. Cheng, and R. Urtasun. Physically realizable adversarial examples for lidar object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13716–13725, 2020.

[20] A. Boloor, K. Garimella, X. He, C. Gill, Y. Vorobeychik, and X. Zhang. Attacking vision-based perception in end-to-end autonomous driving models. *Journal of Systems Architecture*, 110:101766, 2020.

[21] R. Duan, X. Ma, Y. Wang, J. Bailey, A. K. Qin, and Y. Yang. Adversarial camouflage: Hiding physical-world attacks with natural styles. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1000–1008, 2020.

[22] J. Tu, H. Li, X. Yan, M. Ren, Y. Chen, M. Liang, E. Bitar, E. Yumer, and R. Urtasun. Exploring adversarial robustness of multi-sensor perception systems in self driving. *arXiv preprint arXiv:2101.06784*, 2021.

[23] B. Gangopadhyay, S. Khastgir, S. Dey, P. Dasgupta, G. Montana, and P. Jennings. Identification of test cases for automated driving systems using bayesian optimization. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 1961–1967. IEEE, 2019.

[24] Y. Abeysirigoonawardena, F. Shkurti, and G. Dudek. Generating adversarial driving scenarios in high-fidelity simulators. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8271–8277. IEEE, 2019.

[25] S. Silvetti, A. Policriti, and L. Bortolussi. An active learning approach to the falsification of black box cyber-physical systems. In *Integrated Formal Methods: 13th International Conference, IFM 2017, Turin, Italy, September 20-22, 2017, Proceedings 13*, pages 3–17. Springer, 2017.

[26] T. Dreossi, A. Donzé, and S. A. Seshia. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning*, 63:1031–1053, 2019.

[27] S. Ghosh, F. Berkenkamp, G. Ranade, S. Qadeer, and A. Kapoor. Verifying controllers against adversarial examples with bayesian optimization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7306–7313. IEEE, 2018.

[28] B. Bryan, R. C. Nichol, C. R. Genovese, J. Schneider, C. J. Miller, and L. Wasserman. Active learning for identifying function threshold boundaries. *Advances in neural information processing systems*, 18, 2005.

[29] A. Gotovos. Active learning for level set estimation. Master's thesis, Eidgenössische Technische Hochschule Zürich, Department of Computer Science,, 2013.

[30] A. Zanette, J. Zhang, and M. J. Kochenderfer. Robust super-level set estimation using gaussian processes. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II 18*, pages 276–291. Springer, 2019.

[31] A. Sadat, M. Ren, A. Pokrovsky, Y.-C. Lin, E. Yumer, and R. Urtasun. Jointly learnable behavior and trajectory planning for self-driving vehicles. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3949–3956. IEEE, 2019.

[32] D. Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[33] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[34] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[35] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[36] V. Franc, A. Zien, and B. Schölkopf. Support vector machines as probabilistic models. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 665–672, 2011.