Suraksha: A Framework to Analyze the Safety Implications of Perception Design Choices in AVs

Hengyu Zhao[‡], Siva Kumar Sastry Hari[†], Timothy Tsai[†], Michael B. Sullivan[†], Stephen W. Keckler[†], Jishen Zhao[‡]
[†]NVIDIA, [‡]University of California, San Diego

***Abstract*—Autonomous vehicles (AVs) employ sophisticated computer systems and algorithms to perceive the surroundings, localize, plan, and control the vehicle. With several available design choices for each of the system components, making design decisions without analyzing system-level safety consequences may compromise performance and safety. This paper proposes an automated AV safety evaluation framework called Suraksha to quantify and analyze the sensitivities of different design parameters on AV system safety on a set of driving situations. In this paper, we employ Suraksha to analyze the safety effects of modulating a set of perception parameters (perception being the most resource demanding AV tasks) on an industrial AV system. Results reveal that (a) the perception demands vary with driving scenario difficulty levels; (b) small per-frame inaccuracies and reduced camera processing rate can be traded off for power savings or diversity; (c) tested AV system tolerates up to 10% perception noise and delay even in harder driving scenarios. These results motivate future safety- and performance-aware system optimizations.**

I. Introduction

Vehicles that perform some or all driving tasks autonomously are appearing on roads led by the continuing evolution of automotive and computer technologies [13], [15], [18]. An autonomous vehicle (AV) employs sensors to sense surroundings, software algorithms to process the sensor data and make driving decisions, and a hardware platform that executes the software algorithms in real-time. The AV system performs tasks that are similar to that of a human driver of a conventional vehicle such as perception, localization, planning, and control [21].

AV products must meet stringent safety requirements [10], [24], [36], [52]. AV safety issues stem from numerous sources, such as system design defects, software bugs, or reckless road users [4]. To avoid safety hazards, industrial AV technology developers recently proposed several design standards [10], [11], [44] and safety models [27], [39]. As part of the AV design process, the hazard analysis and risk assessment (HARA) is performed to drive the end-to-end safety requirements, which are then converted to component-level requirements [10], [23]. While component-level testing is performed, no clear process exists to validate these requirements and analyze the effect they have on AV safety. Without a methodology or framework to analyze the sensitivities of these component-level requirements on AV safety, the designed system may be over-provisioned for some components but under-provisioned for others, which may compromise the overall functionality.

Perception is one of the most compute-intensive AV tasks in an AV stack [5], [51]. Perception must process the data from the sensors and detect objects both at sufficiently high precision and low latency [20], [50]. Little research has been conducted to understand the perception requirements, trade-offs offered by different design choices, and implications they may have on safe and optimal system design. For example, for a given AV hardware platform, should the perception task provision lower camera frame per second (FPS) with a high accuracy obstacle perception model or higher camera FPS with a faster but slightly less accurate perception module? Moreover, perception requirements may vary significantly with driving scenarios. For example, the perception requirements for safe driving at 25 MPH on a straight road with no obstacles should be less than a scenario in which the AV is traveling at 45 MPH and reacting to a suddenly-appearing obstacle.

During the development and test process, an AV stack can be configured with different component-level parameters and evaluated in a set of driving scenarios. These scenarios are selected based on an operational design domain (ODD) [33] to limit the scope of testing. Despite the limited scope, the variations in the driving scenarios (e.g., different speeds and maneuver timings) can be too large. System designers can benefit from analyzing the effects of varying component-level parameters on a set of driving scenarios with specific settings (e.g., speeds that result in near collisions in a given ODD). In such evaluations, using metrics that are independent of the scenarios and AV's internal states can provide a uniform method to quantify safety of different AV versions. To the best of our knowledge, no study has analyzed the effects of varying AV's component-level parameters on a targeted set of driving scenarios using system-independent safety metrics.

In this paper, we propose Suraksha (Suraksha means safety in Sanskrit), an AV safety evaluation framework that automates the analysis of the safety effects of different component-level design choices. Suraksha automatically generates AV versions based on user-specified low-level component parameters, and driving scenarios based on the desired difficulty levels. It quantifies safety using a set of system-independent metrics, and analyzes safety sensitivity of different component parameters. Automation of all these steps accelerates the discovery of the performance vs. safety trade-offs offered by different design choices. The insights derived from the trade-offs and sensitivity analysis of component-level design choices enabled by Suraksha allow the AV engineers to better design the system for performance and safety.

With the focus on the perception system, we employ

Suraksha to evaluate the safety effects by degrading perception quality with multiple component-level parameters, including camera FPS and neural network model precision. To further understand the perception quality limits that allow for safe driving in the selected scenarios, we developed models that introduce inaccuracies directly into the perceived surroundings before it is sent to the planning and control tasks. We study the safety tolerance in an industry-implemented AV by injecting noise in every frame's perception output and introducing intermittent delay in perception. The results interestingly reveal that the tested AV tolerated significant world model inaccuracies even in hard driving scenarios where hard braking was originally required to remain safe. These findings indicate potential system optimization opportunities and highlights the benefits of using Suraksha. In summary, the paper makes the following key contributions.

- We present an automated AV safety evaluation framework called Suraksha that quantitatively analyzes safety sensitivity of different AV versions configured using component-level parameters while running driving scenarios based on a user-selected difficulty level.
- We employ Suraksha to study the safety effects of deteriorating perception quality by changing component-level parameters. We also introduce inaccuracies to the perceived world model directly to study the limits of the AV's perception requirements for safe driving.
- The results demonstrate that an industry-developed AV tolerates small per-frame inaccuracies and reduced camera FPS even in the hardest tested scenarios. We show that the Suraksha's sensitivity analysis identifies the perception parameters that affect safety the most when altered.

II. Background and Challenges

A. AV Stack

A typical AV stack consists of software and hardware stacks. The software stack performs three major AV tasks – perception, localization, and planning control. Perception detects and interprets surrounding static (e.g., lane lines, trees, and traffic lights) and moving (e.g. other vehicles and pedestrians) objects or obstacles, based on camera, radar, ultrasonic sensors, and/or light detecting and ranging (LiDAR) [21]. Localization identifies the current AV location in a high-precision map using GPS/IMU sensors and/or scan-based methods [48]. Planning control employs perception and localization results to plan for the trajectory, predict other obstacles' behaviors, and generate actuator commands to the vehicle controller. The hardware stack typically adopts multiple sensors and a heterogeneous computing system with CPUs, GPUs, and accelerators to execute the software modules [21].

B. AV Safety Validation

Safety engineering and testing: An AV system is often tested and validated in constrained driving situations defined by an operational design domain (ODD) [14], [33]. An ODD may include but not limit to environmental, geographical, time-of-day, and traffic constraints. AV designers specify a suite of scenario categories (e.g., cut-in and cut-out) based on the selected ODD to test whether the AV under test follows traffic rules and etiquette while keeping the AV safe.

An AV is also tested under hazardous situations. ISO 26262 safety standard requires a Hazard Analysis and Risk Assessment (HARA) to be performed to determine vehicle level hazards. This process guides the safety engineers towards safety goals that are then used to create functional safety requirements [10], [23]. These requirements guide the system development process, which is then decomposed into hardware and software development processes. The AV manufacturers will conduct extensive analyses beyond what is specified by ISO 26262-HARA. Analyzing the effects of varying component-level requirements on AV safety under challenging operating scenarios can discover better resource usage opportunities, which is the focus of this paper.

AV simulation: AV simulators are widely used to develop techniques for testing [7], [32], [38]. Simulation-based testing is salable and safe. An AV simulator has four key tasks. (1) It models the driving scenarios. A *driving scenario* consists of the information about the surroundings, such as traffic lights and signs, pedestrians, traffic, weather, and time of the day (e.g., sun position). (2) It models sensors such as cameras, LiDAR, radar, GPS/IMU, and Ultrasonic devices, and offers an interface for the AV to receive the sensor data. (3) It models the physical characteristics of the real world, such as visibility due to cloudy weather and less friction on wet roads. (4) It models vehicle dynamics and offers an interface to receive actuator commands (e.g., steering) from an AV.

Real-world testing: Real-world testing on public roads will be employed by AV manufacturers to ensure that their designs are fully operational and safe [13], [18]. However, such testing is expensive to perform at the design stage and difficult to scale to analyze design choices.

C. Challenges

Despite the current development in AV system design and safety validation, substantial challenges still remain in designing a resource-efficient and safe AV. (1) As AV designs are composed of many low-level components, assigning requirements to them and ensuring an efficient and optimal assignment for the overall AV safety is challenging. (2) Component-level metrics such as DNN model precision, which has been used by prior work [19], [46], do not provide sufficient insights into the safety impact of different component-level design choices. (3) Even in a selected ODD, the set of available scenarios can be too large. While the safety depends on the driving scenarios [26], studying the effects of component-level design choices on a potentially unbounded set of scenarios is impractical.

III. Suraksha Framework

We address the above listed challenges by developing an automated framework, Suraksha and employing it on an industry AV system to improve our understanding of the perception component-level requirements and allow AV designers to make better trade-offs. Suraksha quantifies safety

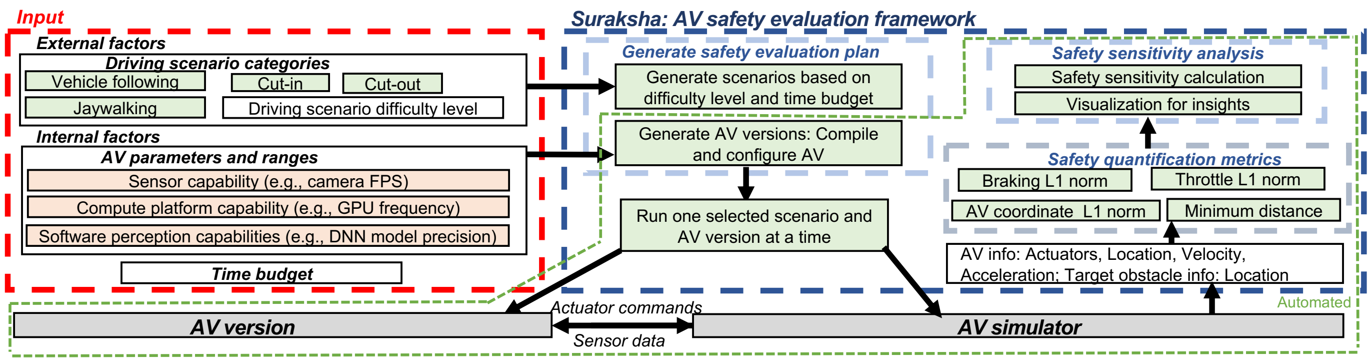

Input
Suraksha: AV safety evaluation framework
External factors
Driving scenario categories
Vehicle following
Cut-in
Cut-out
Jaywalking
Driving scenario difficulty level
Internal factors
AV parameters and ranges
Sensor capability (e.g., camera FPS)
Compute platform capability (e.g., GPU frequency)
Software perception capabilities (e.g., DNN model precision)
Time budget
Generate safety evaluation plan
Generate scenarios based on difficulty level and time budget
Generate AV versions: Compile and configure AV
Run one selected scenario and AV version at a time
Safety sensitivity analysis
Safety sensitivity calculation
Visualization for insights
Safety quantification metrics
Braking L1 norm
Throttle L1 norm
AV coordinate L1 norm
Minimum distance
AV info: Actuators, Location, Velocity, Acceleration; Target obstacle info: Location
Automated
AV version
Actuator commands
Sensor data
AV simulator

Fig. 1. Overview of our AV safety evaluation framework Suraksha.

for different AV versions. An **AV version** is an AV system compiled with specific options or configured with specific parameter settings, e.g., FP16 models running on an 800MHz GPU processing 30 camera FPS. The ability to focus on a set of scenarios by specifying a difficulty level in a specific ODD can make Sraksha's analysis practical. Figure 1 summarizes the Suraksha's components. Suraksha consists of three main components – methods to generate AV versions and driving scenarios for safety evaluation, safety quantification metrics for a single experiment (for a chosen AV version and driving scenario combination), and safety sensitivity analysis across a set of scenarios for the different AV versions being considered. The inputs to Suraksha are the following: (1) a set of AV parameters that define different AV versions, (2) a set of driving scenario categories based on user-selected ODD, and (3) scenario difficulty level, which is used by Suraksha to initialize conditions, e.g., speeds of actors in the scenarios. The outputs of Suraksha are safety quantification metrics for different AV versions and safety sensitivity quantification for each of the AV parameters; these results will aid the AV designers to identify sensitive factors and optimize the system for safety and performance. Suraksha can also be leveraged to enable product-specific customization based on the context/ODD (e.g., local roads vs. highway) and available hardware resources.

A. *Generating AV Versions and Driving Scenarios*

Suraksha receives high-level inputs such as AV design parameters and a set of driving scenario categories the designers want to use for AV safety analysis, and creates an evaluation plan. While creating a plan, Suraksha considers (1) the number of AV versions that can be generated based on the selected parameters and their setting ranges (changing one parameter at a time), (2) the number of driving scenario categories provided for which challenging scenario settings need to be generated, (3) the duration of each of the scenario categories, and (4) the provided time-budget. Each experiment in the created plan specifies an AV version and a driving scenario for simulation. While we primarily investigate changing one parameter setting at a time, which helped us uncover new insights, the plan generation algorithm can also consider AV versions that change multiple parameter settings. Such an investigation increases the exploration space significantly. We study a preliminary strategy to explore this space in Section VI-F, paving the way for future research in this direction.

AV version generation: For a given experiment, Suraksha generates an AV version using a set of selected AV parameters and settings. It compiles the AV software, configures it, or changes hardware configuration, as needed.

Driving scenario generation: An input to Suraksha is a high-level description of a scenario category, without the details such as the speeds of actors and their relative locations. For a given scenario category, the goal is to generate a scenario that is challenging but does not include an accident that is unavoidable. Based on the user-specified difficulty level, Suraksha generates driving scenarios by setting initial conditions, i.e., velocities of different actors and their relative positions on the road. To achieve the best evaluation efficiency, users may select categories based on an ODD and the following principles. (1) The selected scenario category should be realistic enough to be encountered by human drivers on the road. (2) While being realistic, the scenarios should be more challenging than most driving scenarios to expose safety issues.

B. *Safety Quantification Metrics*

As the goal is to study AV safety effects by changing AV system parameters and settings, the low-level metrics such as system latency, sensor processing rate, and system power consumption are inappropriate as they fail to capture high-level safety effects. The high-level AV behavior may change less compared to internal factors. So, we consider the following high-level behavioral aspects (also summarized in Table I).

Distance from obstacles: An AV's goal is to avoid collisions and stay safe on the roads. There can be many actors around the AV in a driving scenario, and the distance from each of them can be tracked over time. The minimum distance to the closest actor in the entire scenario is an indicator to how close the AV came to a collision. If the minimum distance is below a threshold (e.g., 0), the AV collides with an obstacle. We use minimum distance to the closet actor during a scenario (or simply minimum distance) as a key safety metric.

AV trajectory: When an AV system changes, the AV's driving behavior and trajectory can also change. This change is quantified by computing an L1 norm using the AV's coordinates over time in a driving scenario and the coordinate values from

TABLE I
SAFETY QUANTIFICATION METRICS.

Metric	Description
Minimum distance	The minimum distance between the AV and any target during the simulation
AV coordinate (x, y, z) L1 norm	L1 norm of AV coordinate (x, y, z) between default configuration and any AV version
Braking value L1 norm	L1 norm [42] of braking value between default configuration and any AV version
Throttle value L1 norm	L1 norm of throttle value between default configuration and any AV version

a different execution using the same scenario but with the default (or baseline) AV version.

Actuator values: Actuators include brake, throttle, and steering. If the AV stack changes, these values will also change, and can eventually affect safety. We quantify the change in the actuator values due to a change in the AV version by computing L1 norm per actuator.

C. Safety Sensitivity Analysis

For each of the AV parameters used to generate AV versions, Suraksha also computes a sensitivity metric by leveraging the previously computed safety metrics (e.g., minimum distance) for different driving scenarios. As part of the sensitivity metric computation, Suraksha computes the relative safety metric value difference between two neighboring settings of the selected AV parameter; we call it *delta*. Delta reflects the relative change in safety due to a change in a setting to the next or previous AV parameter setting. The sensitivity metric is obtained by aggregating the delta values. In this aggregation, worst-case and best-case delta values can be weighted differently and can lead to different insights. So we compute *average* delta that weighs all the outcomes equally, *minimum* delta that considers the worst-case outcome only, and the 10*th* *percentile* that weighs the worst-case outcomes more than the best-case outcome, assuming a lower metric value is worse (e.g., lower minimum distance is less safe and hence is considered worse).

IV. SURAKSHA IMPLEMENTATION

We automate the generation of AV versions via component-level setting selection and/or recompilation, running the simulations on selected scenarios, and extracting the metrics from the logs. We encapsulate the automated components in a green dashed-line box in Fig. 1.

A. Generating AV Versions and Driving Scenarios

AV version: Suraksha investigates AV safety by changing one AV parameter at a time. For one experiment, Suraksha sets the AV parameter by modifying one or multiple of the following: AV configuration file, sensor rig file, AV software compilation flags, and hardware settings (e.g., GPU clock setting). Additionally, we experiment by changing two AV parameters at a time (e.g., FPS and model precision), where a sensitive parameter is set based on the value below which AV becomes unsafe for the considered scenarios.

Driving scenarios: According to the principles described in Section III-A, we select an urban expressway as the ODD and select four scenario categories that are realistic, encountered occasionally on-road, and can be set to create a challenging condition for the AV, i.e., the AV has to apply hard brakes after detecting a suddenly appearing obstacle. The four scenario categories, which are inspired by NCAP scenarios [26], are as follows. In the first scenario (*Vehicle following*), a vehicle traveling ahead of the AV brakes suddenly. In the second scenario (*Cut-in*), a vehicle traveling at a slower speed changes lane ahead of the AV from an adjacent lane to the lane in which the AV is driving. In the third scenario (*Cut-out*), a vehicle ahead of the AV moves out to suddenly reveal a static obstacle. In the fourth scenario (*Jaywalking*), a jaywalker crossing the road appears from behind a car stopped in an adjacent lane ahead of the AV. In all these scenarios an obstacle is revealed suddenly, and the AV needs to perceive the obstacle and apply hard braking to avoid a collision. These scenarios are summarized in Figure 2.

For each driving scenario, we initialize locations and velocities for each of the actors (e.g., $v_{AV,t1}$ and $v_{target,t1}$ will be the velocities for the AV and target obstacle at the start of the scenario, $t1$). We define t_2 as the time when the AV starts reacting to the target (e.g., traffic cone or jaywalker in scenarios (c) and (d) in Figure 2 and t_3 as the time when the AV reaches the target's speed. We define braking distance, d_{t2-t3}, as the distance travelled by the AV between t_2 and t_3. Scenarios with shorter d_{t2-t3} are more challenging for AVs, because the AV needs to apply harder braking. Higher braking results in higher deceleration, which may result in an uncomfortable drive.

The AV can apply uniform or non-uniform braking, so the braking distance can vary between AV versions based on the differences in perception and driving policies (some AV versions may stop closer to the obstacle than the others). We employ Equation 1 to calculate the average braking acceleration (a_{avg}) for a given scenario, where $v_{AV,t2}$ is the velocity of the AV at t_2 and $v_{target,t3}$ is the velocity of the target at t_3.

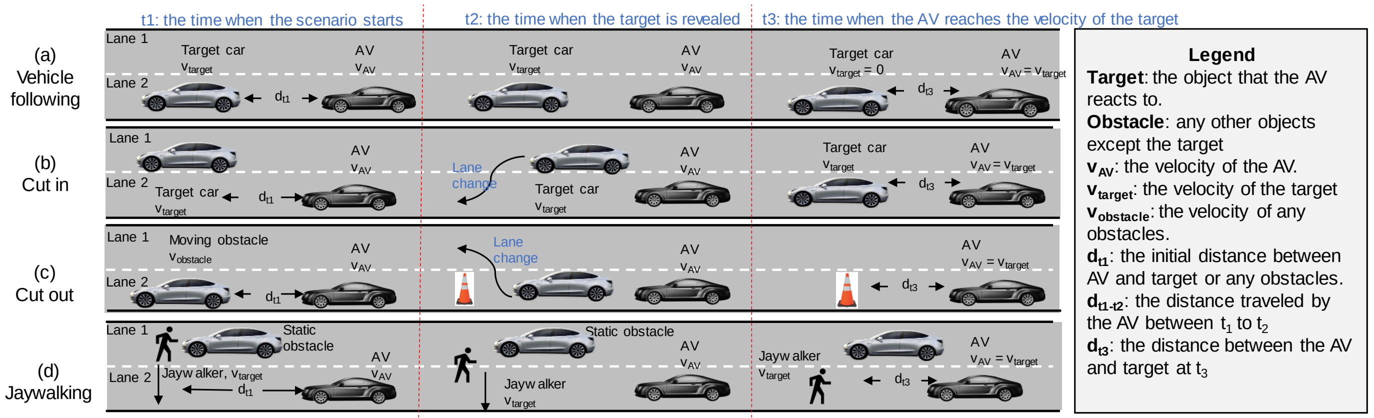
We simulate the baseline AV on a scenario (with an initial assignment) and categorize it by comparing a_{avg} with the gravitational acceleration ($g = 9.8m/s^2$). We label the scenario hard if $a_{avg} > \frac{g}{2}$, moderate if $\frac{g}{4} < a_{avg} \leq \frac{g}{2}$, and easy if $a_{avg} \leq \frac{g}{4}$. Based on the user-specified objective, Suraksha can select a desired mix of scenarios with different difficulty levels.

$$a_{avg} = \frac{v^2_{AV,t2} - v^2_{target,t3}}{2d_{t2-t3}} \quad (1) \qquad L1 = \frac{\sum^N |q_{v_e,t} - q_{v_b,t}|}{N} \quad (2)$$

$$delta_{p,d} = \frac{m_{p,s_2,d} - m_{p,s_1,d}}{m_{p,s_1,d}} \quad (3)$$

B. Safety Quantification Metrics

For a given scenario, we collect the coordinates of all the actors (including the AV) along with the AV's actuator values from the simulator at a fixed frequency, which is 60Hz in our setup. We use this information to compute our safety metrics, which are the minimum distance between the AV and any obstacle in the scenario at any timestep, and the L1 norms computed using AV coordinates, braking values, and throttle



t1: the time when the scenario starts
t2: the time when the target is revealed
t3: the time when the AV reaches the velocity of the target
(a) Vehicle following
(b) Cut in
(c) Cut out
(d) Jaywalking
Lane 1
Lane 2
Target car
AV
Moving obstacle
Static obstacle
Jaywalker
Lane change
Legend
Target: the object that the AV reacts to.
Obstacle: any other objects except the target
v_AV: the velocity of the AV.
v_target: the velocity of the target
v_obstacle: the velocity of any obstacles.
d_t1: the initial distance between AV and target or any obstacles.
d_t1-t2: the distance traveled by the AV between t1 to t2
d_t3: the distance between the AV and target at t3

Fig. 2. Four driving scenario categories for an urban expressway ODD.

values between an AV version being evaluated and the baseline system (as described in Section III-B). For the baseline system, we use the out-of-the-box AV version with the default values assigned to the AV stack parameters (shown in Table II).

We calculate minimum distance in meters. We store the AV coordinate and actuator data from the scenario executed with the baseline AV for the L1 norm computation with different AV versions. We calculate the difference between values obtained from the two simulations (using the baseline AV and a new AV version) at each timestep in the scenario. We accumulate the differences over all the timesteps and normalize it by dividing it by the number of timesteps (N). This calculation is described in Equation 2, where $q_{v_e,t}$ and $q_{v_b,t}$ represent measured actuator or coordinate values at timestep t for the two AV versions (v_e refers to the AV version that will be evaluated and v_b refers to the baseline AV version), respectively. If a simulation ends sooner at timestep N_s, with $N_s < N$ (due to a collision, for example), we compute the L1 norm only for the N_s timesteps.

C. Safety Sensitivity Implementation

Suraksha obtains the AV parameter list and their setting ranges from the user. For each parameter and setting combination, Suraksha conducts one experiment for each of the selected driving scenarios. We compute the delta between the neighboring settings to analyze sensitivity of a parameter using Equation 3. For a parameter, two continuous settings are neighbors. For example, a camera with three FPS settings (say 1,2,3), FPS settings 1,2 and 2,3 are two sets of neighbors. In this equation, m refers to a metric value and it can be any of four metrics listed in Table I (e.g., minimum distance); d is the driving scenario from which the reading is obtained; p is the design parameter (e.g., FPS); s_1 and s_2 are two neighbor settings for the parameter p; and s_1 is closer to the default setting (e.g., $s_1 = 15$ and $s_2 = 10$ with default FPS = 30).

To analyze sensitivity of a parameter p, we compute average, minimum, and 10^{th} percentile using the computed $delta_{p,d}$ across all the driving scenarios. We refer to them as $avg(delta_p)$, $min(delta_p)$, and $10^{th} percentile(delta_p)$. The higher the value, the more sensitive the parameter is. We also compute average, minimum, or 10^{th} percentile for parameter p using $m_{p,s_1,d}$ across all the settings of p and driving scenarios. We refer to them as $avg(m_p)$, $min(m_p)$, and $10^{th} percentile(m_p)$. For metrics where higher value is worse (e.g., L1 norm) we use avg, max, and $90^{th} percentile$ functions instead. As the final step, we visualize the computed values for the desired metric (e.g., $10^{th} percentile(delta_p)$ and $10^{th} percentile(m_p)$ for minimum distance) to gain insights into the sensitivity of p.

V. Perception Quality Requirements Analysis

The general Suraksha framework can be used to analyze the safety effects of using a different sensor suite, perception stack, localization algorithm, and path planning strategy. Once the AV designers parameterize the system for the components they want to study, Suraksha can configure the AV system for an automated experimentation and analysis. In this paper, we focus on evaluating the effects of changing perception quality on AV safety. It has been demonstrated that the computational capability offered by hardware platform for complex perception stacks can be a bottleneck [17], [20], [45], [50], making it important to understand the effect of different perception parameters on AV safety to aid the development of an optimal and safer system design. We generate AV versions that degraded perception quality compared to the baseline AV version and evaluate their impact on safety. We adjust AV's perception quality by modifying AV's perception parameters and by directly modifying the perceived world model, which is the main part of the input to the planning and control logic. We summarize the AV parameters and inaccurate world model perception models we use in this study below and in Table II.

Software parameters: We select two parameters related to the object detection CNN model: model precision and model version. The model precision has two options – INT8 and FP16, each of which refers to the data type used by the optimized model during inference. The model accuracy of INT8 model is slightly lower than that of FP16. The model version has two options – v1 and v2, each of which are trained and optimized differently. Model v2 has higher precision than model v1.

Hardware parameters: We select two existing parameters to adjust the compute intensity – GPU frequency and camera FPS. We study six settings for the GPU frequency, which spans the valid GPU frequency range for the GPU used in the experiments (NVIDIA RTX 2080). We use eight settings for

TABLE II
PARAMETERS AND CORRUPTION MODELS USED TO DEGRADE PERCEPTION.

Category	Parameter/model	Settings
Software parameter	CNN model precision	INT8, FP16 (default)
	CNN model version	v1 (default), v2
Hardware parameter	Camera FPS	1, 2, 3, 5, 6, 10, 15, 30 (default)
	GPU frequency (MHz)	300, 500, 800, 1000, 1200 (default)
Inaccurate world model prediction	Random noise in obstacle distance/velocity	[-0.1,0.1], {[-0.3,-0.1),(0.1,0.3]}, {[-0.5,-0.3),(0.3,0.5]}, {[-0.7,-0.5),(0.5,0.7]}, {[-0.9,-0.7),(0.7,0.9]}
	Positive noise in obstacle distance/velocity	[0, 0.1], (0.1, 0.3], (0.3, 0.5], (0.5, 0.7], (0.7, 0.9]
	Negative noise in obstacle distance/velocity	[-0.1,0], [-0.3,0], [-0.5,0], [-0.7,0], [-0.9,0]
	Obstacle perception delay (in frames) per 100 frames	10, 30, 50, 70, 90
	World model loss (in frames) per 100 frames	10, 30, 50, 70, 90

camera FPS. The maximum (and default) camera FPS is 30. Each of selected FPS settings is a divisor of 30.

Inaccurate world model prediction: Varying individual perception parameters (or a group of them) can inform the parameters critical to safety, but does not disclose the perception quality limits needed for safe operation, which can help design a better perception system. To better understand the perception needs, we introduce inaccuracies directly in the world model. We study three types of inaccuracies – random noise, obstacle perception delay, and world model loss. We model inaccurate obstacle perception with more noise using the random noise model; the perception delay model models stress on compute resource (e.g., hardware over-subscription or power throttling due to operating conditions) that can introduce intermittent delay; and the world model loss aims to model intermittent perception failures due to hardware or software issues. For a world object, we change the perceived location and velocity.

For the random noise model, we introduce noise in the target obstacle's location and velocity at *every timestep* during the scenario. We further consider three options – inject random noise including both positive and negative noise; inject positive noise such that the object is perceived farther and faster than its actual distance and speed, respectively; inject negative noise such that the object is perceived closer and slower than its actual distance and speed, respectively. For each of these options, we select five settings. A setting specifies the error range for the noise injection. Equation 4 shows our noise injection model. $W_{perceived,t}$ and $W_{inaccurate,t}$ in the equation refer to the perceived world model at timestep t before and after the noise injection, respectively; e_t refers to the random noise value at time t selected based on the range setting; and l_1, l_2, l_3, l_4 define the range of the noise as listed in Table II. For example, the second setting for random noise injection model in Table II sets $l_1 = -0.3, l_2 = -0.1, l_3 = 0.1, l_4 = 0.3$ and e_t is randomly generated from the ranges [-0.3,-0.1) and (0.1,0.3].

$$W_{inaccurate,t} = W_{perceived,t} \times (1 + e_t) \qquad \begin{matrix} l_1 \leq e_t \leq l_2 \\ l_3 \leq e_t \leq l_4 \end{matrix} \tag{4}$$

$$W_{inaccurate,f1} = W_{perceived,f2} \qquad \begin{matrix} f2 = \left\lfloor \frac{f1}{F} \right\rfloor, f1 \bmod F = 1..D \\ f2 = f1, \quad f1 \bmod F = D..F \end{matrix} \tag{5}$$

$$\begin{matrix} W_{inaccurate,f} = W_{perceived,f} & f \bmod F = D..F \\ W_{inaccurate,f} = 0 \ (Blank) & f \bmod F = 1..D \end{matrix} \tag{6}$$

For the perception delay model, we consider five settings (shown in Table II) that specify the number of frames (D) for which the perception output will be delayed every F frames ($D < F$). Equation 5 explains our delay model.

To study world model loss, we consider five settings (shown in Table II) that specify the number of frames (D) for which the target's information is lost (or removed) from the world model every F frames ($D < F$). Equation 6 explains this model.

As described earlier, the goal of these models is to understand the limits in perception inaccuracies introduced by the AV stack that are tolerated and do not result in AV safety issues. Models that capture the manifestations of random hardware errors (e.g., transient and permanent errors) or sensor errors (e.g., mud on camera or dead pixels in a camera) at the world model level can also be incorporated in Suraksha, to expand the scope of analysis, which is part of our future work.

VI. EVALUATION

A. Experimental Methodology

Simulation methodology and AV stack: We use NVIDIA DRIVE Sim to simulate the environment, driving scenarios, and AV response [32]. It provides a virtual sensor set (e.g., cameras, radars, LiDARs) to configure AV sensor rigs and an interface to forward the sensor data to the AV stack for perception. It also provides a simulation state monitoring interface to collect AV actuator values, and location and velocity of all the actors, which we use to calculate safety metrics defined in Section III-B. We use NVIDIA's state-of-the-art Level-2 AV in this paper [30]. Level 2 autonomy provides partial automation of driving functions. One of its key function is Adaptive Cruise Control (ACC) [22], which is responsible for maintaining a safe distance while following a vehicle in the front. In our implementations, the AV stack employs one front-facing camera and one radar to perceive the obstacles. While the sensor configuration is scalable, we use this setup to analyze a common use-case. Simulations can be non-deterministic. We run the experiments twice to analyze the effects. We show the data from one campaign in the following sections, which show representative trends. For simulators that are repeatable, a single experiment campaign is sufficient.

Hardware: We execute the simulator and the AV stack on a PC with Intel(R) Core(TM) i7-5820K CPU with 32GB system memory and two discrete GPU cards. We run the simulator on an NVIDIA TITAN V (Volta) GPU [29] and the AV stack on an NVIDIA GeForce RTX 2080 (Turing) GPU [28]. Changing GPU frequency for RTX 2080 from 1200MHz to 300MHz, reduces computational throughput and increases the latency by 4× each. We do not change the memory frequency.

TABLE III
DRIVING SCENARIO ANALYSIS. WE SELECT THE SCENARIOS IN BOLD FONT FOR FURTHER ANALYSIS.

Category	#	d_{t1} (m)	$v_{AV,t2}$ (m/s)	$v_{target,t3}$ (m/s)	d_{t2-t3} (m)	a_{avg}	Difficulty
Vehicle following	**1**	**100**	**25.9**	**0**	**65.8**	**5.10**	**Hard**
	2	100	24.67	0	60.6	5.02	Hard
	3	**50**	**6.55**	**0**	**9.9**	**2.17**	**Easy**
Cut-in	**1**	**400**	**31.8**	**8.9**	**68.2**	**3.84**	**Moderate**
	2	330	25.9	8.9	51.3	2.82	Moderate
	3	**265**	**22**	**8.9**	**42.9**	**2**	**Easy**
Cut-out	**1**	**30**	**27.7**	**0**	**63.1**	**6.08**	**Hard**
	2	40	24.8	0	53	5.80	Hard
	3	**90**	**13.3**	**0**	**22.3**	**3.97**	**Moderate**
Jaywalking	**1**	**180**	**16.5**	**0**	**28.7**	**4.74**	**Moderate**
	2	230	15.2	0	26.4	4.38	Moderate
	3	**280**	**13.4**	**0**	**43.8**	**2.05**	**Easy**

Safety quantification metrics: We use the minimum distance threshold as a uniform metric across scenarios that do not have collisions but are unsafe. The exact minimum distance for a collision depends on several factors – center of the vehicle, which is used to measure the distance between two vehicles; size of the car, and orientation of the car. We account for the size of the AV and the other actor, but not the orientation and shape. So we consider the AV has a collision when the minimum distance is less than 3m to account for the uncertainties in vehicle orientations and shapes. This threshold can be adjusted based on the relative speeds. We consider the AV to be safe when the minimum distance is more than the length of the AV (we use Ford Fusion as the AV, which is about 5m long), and it is close to obstacle when the minimum distance is between 3m and 5m. For AV trajectory change, we only show the L1 norm of the AV's y coordinate values because the AV drives along y-axis orientation in all four scenarios.

B. Driving Scenarios

As described in Sections III-A and IV-A, we select four categories of driving scenarios and generate scenarios for each category with different levels of difficulties. We used scenarios that are challenging (need high deceleration), but can be safely navigated by an attentive driver. We created tens of variations per scenario category with different initial conditions. In Table III, we show 12 driving scenarios, three scenarios per category to show the effect of the initial conditions on a_{avg}, which we use to assign the difficulty level. The higher the AV velocity when the target is revealed ($v_{AV,t2}$), the harder the scenario is, because the AV must perform harder braking to avoid a collision. A harder scenario implies quick and accurate perception is required to brake in time and avoid the collision. We also find that the shorter the initial distance between the AV and the target (d_{t1}), the harder the scenario, for similar reasons. For analysis, we select one relatively harder and one relatively easier scenarios for each category, which are shown in bold font in Table III.

C. Degrading Perception with SW and HW Parameters

Figure 3 shows the results by adjusting four perception parameters – model type, model precision, camera FPS, and GPU frequency. Minimum distance is plotted on the left y-axis, and whenever the value goes below the red dotted horizontal line (3m), we assume the AV is in a collision. L1 norms of the brake and throttle values compared to the baseline AV version are plotted on the secondary (right) y-axis. The range for brake and throttle values is [0,1]. L1 norm for the AV's y-coordinate values, which quantifies the change in the AV's trajectory, is also plotted on the secondary y-axis. It is measured in meters but we scale the values (to kilometers) to plot it on the same secondary y-axis. The L1 norm will be small if small changes are observed in many timesteps or large differences are observed in a few timesteps.

Model precision: The use of INT8 models results in slightly higher L1 norms for braking, throttle, and y coordinate. The INT8 model used for perception is slightly less accurate than FP16, which means there might be some noise in obstacle perception. If an obstacle is perceived closer in some timesteps, the planning module may take a conservative approach and brake harder to avoid a potential collision, increasing the minimum distance. We observe a slight increase in the minimum distance in most scenarios we tested with INT8. The minimum distance may decrease if the error introduced by the less-accurate model always predicts the obstacle farther than its actual location, which may be the case with the easier cut-in scenario. While we observe some differences, no safety concerns are observed.

Model version: Results show that safety with v2 is similar to v1 at FP16 precision. At INT8 precision, v2 is closer to the default (FP16+v1) than INT8+v1, which implies that the enhancements made in v2 improve accuracy of the INT8 model.

FPS: AVs can tolerate lower camera FPS in easier scenarios as there is enough time to perceive the obstacle (even with the delay) and apply sufficient braking to avoid the collision. The results for the vehicle following scenario show that FPS of 3 is tolerable for the easier scenario but at least 5 FPS is required to remain safe for the harder scenario. Among the harder scenarios, cut-out and jaywalking require quicker response based on the scenario settings.

GPU frequency: For the baseline AV, the GPU processes 30 frames per second. This computation demand is satisfied by the default GPU settings as well as with 300MHz (the lowest setting we tested). The results can be different with a smaller GPU or when more perception algorithms are run per camera or with more cameras, testing which is part of our future work.

Based on these results, we find the following:

- High L1 norm of braking, throttle, and AV's y-coordinate do not always imply a safety concern. The planning algorithm changes how the AV's react to obstacles between the tested AV versions, which can affect the L1 norms but not the minimum distance. Simulation non-determinism (small change in many timesteps) can also make the L1 norms non-negligible. Minimum distance, however, may not change much even when the L1 norms change. So, we use the minimum distance as the primary metric.
- Small inaccuracies introduced by optimized models (e.g., INT8 versus FP16 model) show minimal effects on AV safety. Trading off small per-frame accuracy for significantly

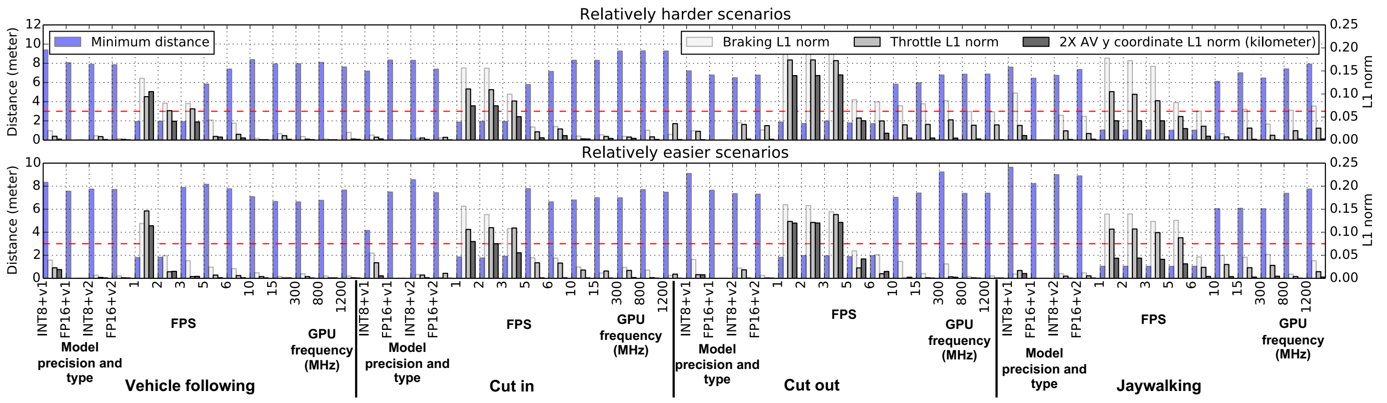


Fig. 3. Effects on AV safety due to perception degradation with software and hardware parameters. For each scenario and parameter settings (x-axis), primary and secondary y-axes show the minimum distance and L1 norms, respectively. A blue bar (minimum distance) below the red dotted line indicates a collision.

higher throughput will allow the designers to consider adding diversity from additional sensors to avoid possibility of consistent mispredictions and make the system safer.

- Camera FPS is the most safety sensitive parameter among the studied perception parameters, but shows significant tolerance even in the harder scenarios.
- Perception requirements for safe AV driving depend heavily on driving scenarios, i.e., the speed of AV, obstacles, and their distances. Based on these factors, the AV designers may predict the existence of some tolerance and consider optimizations to free up some resources for other tasks (e.g., diverse perception or a more complex planning algorithm).

Finding 1: AV system's tolerance to small per-frame inaccuracies and reduced FPS cameras even in the tested harder scenarios suggest the default system can tolerate even harder scenarios. It also suggests at the existence of opportunities for system optimization and introduction of more diversity to make the system safer.

D. Inaccurate World Model Prediction

To improve our understanding of the perception quality requirements, we introduce inaccuracies in the world model prediction using the models described in Section V. Our results are shown in Figure 4, which uses the format similar to that of Figure 3 except that it does not show throttle L1 norm. We only show the results for the relatively harder scenarios here.
Positive and negative noise: Negative noise places the obstacle closer than its real location, which makes the AV act conservatively and brake harder (braking and y-coordinate L1 norms are higher). This phenomenon increases the minimum distance and results in no safety issues. For the positive noise, the results show that the AV can tolerate up to 50% for vehicle following and cut-in scenarios, 30% for jaywalking, and 10% for cut-out.
Random noise: Random noise introduces randomly generated positive or negative error per frame. Based on the frequency with which the negative noise is introduced, the planner can take a conservative action by braking harder. While the results show that the minimum distance increases as the injected noise range increases (similar to negative noise), there are few exceptions in the jaywalking experiments. In these experiments, we hypothesis that the random noise introduces positive and negative error at different rates, which results in slight variations in the minimum distance trend. We do not observe any safety issues in the harder scenarios with this noise model.
Perception delay: The amount of tolerable delay depends heavily on the scenario. In the harder scenarios for vehicle following, cut-in, cut-out, and jaywalking, the AV tolerates significant delay of 50, 30, 10, and 30 frames, respectively, every 100 frames. We determine the tolerance based on the minimum distance metric (>3m is tolerable). For the vehicle following scenario, we interestingly find that 3 FPS is not tolerated, but a delay of 50 frames every 100 frames while running at 30 FPS is tolerated. We hypothesise that the AV was able to receive fast updated about the target's location after the delay, which allows the AV to react in time. If the speeds were higher, the tolerable delay would have been lower (as is the case in cut-out).
World model loss: This model results in more serious safety issues compared to the above models. The harder vehicle following, cut-in, cut-out, and jaywalking scenarios tolerate 30, 30, 10, and 10 frames of lost information, respectively, every 100 frames.
Based on these results, we find the following:

- As long as the planner is designed to take conservative actions based on the recent obstacle distance predictions, frequent negative noise introduced by the perception stack will not affect safety.
- While the tolerance to inaccurate world model depends heavily on the scenario, results from all the studied harder scenarios show tolerance to 10% positive noise or 10 frames of delay or 10 frames of the lost world model.
- The inaccurate world model results provide an efficient way to search the space to find optimal perception settings for safe operations. For example, the perception delay results showed low tolerance in the harder cut-out scenario, implying that the delay introduced by slower processing units (e.g., limited GPU frequency) may not be tolerated.

Finding 2: The inaccurate world model prediction models provide an efficient way to test the tolerable perception limits to find optimal perception settings for a safe yet efficient AV. We observe that the tested AV has a tolerance to 10% noise and occasional 10 frames of delay or lost information in all

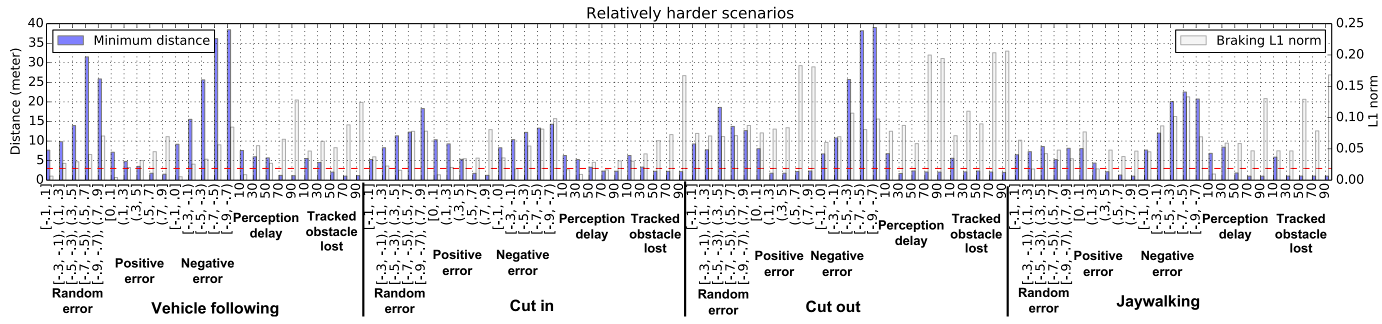


Fig. 4. Effects on AV safety due to inaccuracies in the perceived world model.

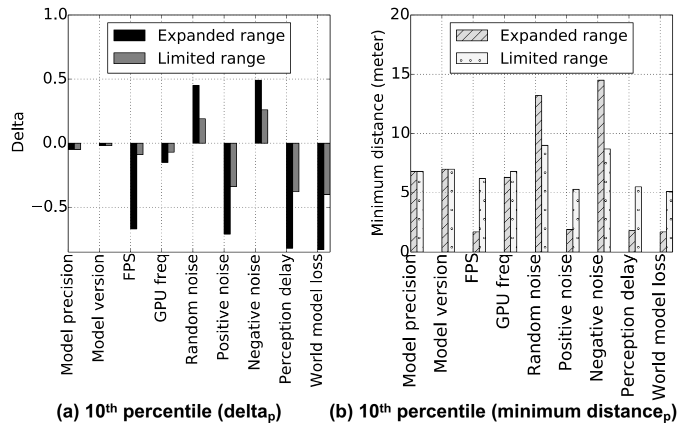


Fig. 5. Sensitivity analysis.

the studied scenarios, which affirm Finding 1.

E. Sensitivity Analysis

The goal of this analysis is to quantify safety sensitivity of each of the design parameters. We use the data from Figures 3 and 4 to conduct the sensitivity study and show the results in Figure 5. We quantify the change in minimum distance metric value when a perception parameter's setting changes by presenting $10^{th} percentile(delta_p)$ in Figure 5(a). The delta values can be positive or negative. A negative value implies that the minimum distance has reduced (the system has become less safe) when a parameter is changed from a setting that is closer to default to a setting that is further (e.g., 30 FPS to 15 FPS for the camera). A positive value implies that the system has become safer. We show the 10^{th} percentile of maximum distance across all the scenarios (10^{th} $percentile(minimum\ distance_p)$) in Figure 5(b) for comparison. While we show $10^{th} percentile$ data here, similar graphs for *avg* and *min* can also be obtained. For both of the figures, the higher the value, the safer the AV. The setting range of a parameter significantly affects the sensitivity analysis. To better understand these effects, we analyze two parameter ranges – expanded and limited.

Expanded parameter ranges: We use these ranges – (1) camera FPS: 5, 6, 10, 15, 30; (2) GPU frequency(MHz): 300, 500, 800, 1000, 1200; (3) random, positive, and negative noise: up to 50% (or -50%); and (4) perception delay and world model loss: 10, 30, 50 frames. The delta values (left figure) and minimum distance values (right figure) suggest that the most safety sensitive parameter among the four SW/HW parameters is FPS. For inaccurate world model prediction, world model loss, perception delay, and positive noise worsen the safety compared to random noise and negative noise. These findings match the results shown in above two sections. These visualizations simplify the identification of the sensitive parameters.

Limited parameter ranges: We use these ranges – (1) camera FPS: 15, 30; (2) GPU frequency (MHz): 800, 1000, 1200; (3) random, positive, and negative noise: up to 10% (or -10%); and (4) perception delay and world model loss: 10 frames. Both delta and minimum distance-based results show no obvious safety issues among the hardware and software parameters, which implies that these parameters have similar negligible safety effects with the limited ranges. The inaccurate world model prediction results also show less safety sensitivity.

Finding 3: Sensitivity analysis presents AV designers with a simpler method to gain insights into the sensitive parameters for a quicker design space analysis.

F. Identifying Optimal but Safe Perception Settings

When designing an optimized AV system, the system designers may want to tune multiple perception parameters. While different strategies can be used to explore the space, we study a greedy approach because the simulations are time-consuming. We evaluate one strategy to demonstrate how Suraksha can be used and plan to expand to other common search techniques in the future. In the greedy approach, we prioritize the sensitive parameters and search by limiting the most-sensitive parameter setting. We demonstrate this approach by analyzing AV safety when the camera FPS, the most sensitive parameter, is set to a critical setting. A critical setting here refers to a value that makes the system unsafe when the value is reduced further. Due to space limitations, we show the results for two representative scenarios: vehicle following (critical FPS=5) and cut-out (critical FPS=10) in Figure 6. It uses the same format as that of Figure 4.

Model: INT8 model precision + model v2 results in a collision for the cut-out scenario (minimum distance is below the red dashed line). Unlike the results in Section VI-C, model precision + version combination has now become sensitive.

GPU frequency: Frequencies lower than 1200MHz result in collisions for the cut-out scenario, which demonstrates that the tolerance to higher latency inference has now diminished.

Noise: Similar to the findings in Section VI-D, negative noise does not introduce any safety concerns. Random error also does not introduce safety concerns, unless the noise moves the

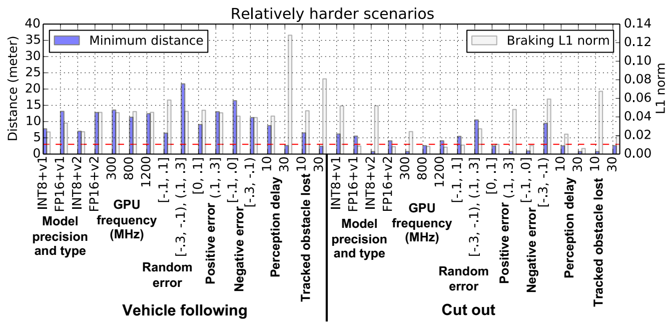


Fig. 6. Fixed FPS analysis.

perceived object further from its real location between $t_2 - t_3$. Results show reduced tolerance to positive error. For the vehicle following scenario, up to 30% positive noise is tolerated, but no positive noise in the cut-out scenario is tolerated.

Perception delay: We observe that up to 10 frames of perception delay is tolerated, but no delay in the cut-out scenario is tolerated. This result corroborates with the intolerance observed when the GPU frequency was reduced.

World model loss: Similar to the above results, we observe 10 frames of the lost world model was tolerated in the vehicle following scenario, but no tolerance in the cut-out scenario.

Finding 4: Limiting a perception parameter to its critical setting (e.g., FPS=10 for cut-out) can reduce the resource demand (by 3$\times$ for cut-out), but removes tolerance in other perception parameters in the AV. Developing optimization strategies that tunes multiple design parameters to design an optimal and safer AV is a promising future direction.

G. Portability

Suraksha is a general AV safety evaluation framework. To demonstrate it's portability to a different software stack, we implemented it using LGSVL simulator [38] with Baidu's Apollo AV software stack [3]. The Apollo stack uses LiDAR as the primary perception module whereas the NVIDIA stack evaluated above uses camera as the primary sensor. We conducted limited experiments by varying LiDAR sensor parameters and injecting random noise into the world model before it is sent to the planner. We show results from the random noise injections for the vehicle following scenario in Table IV. We ran ten simulations per random noise range and show the average and standard deviation (SD) for minimum distance. As the noise range increased, the average distance decreased (still in the safe range) and the SD increased. It shows that the response to the increased noise is different in the Apollo AV stack compared to that of the NVIDIA's stack.

VII. Related Work

Various AV simulation tools are available for AV developers, e.g., LGSVL [38] and CARLA [7]. These simulators can be connected to AV stacks such as Apollo [3] and Autoware [2]. Siemens' Simcenter [40] provides a leading physics-based simulation platform for robust testing of AV safety, reliability, and functionality. This infrastructure is similar to NVIDIA DRIVE Constellation, which is a dedicated data center platform

TABLE IV
Results from random noise injection in the Apollo AV stack.

Random noise	[-0.1,0.1]	[-0.3,-0.1), (0.1,0.3]	[-0.5,-0.3), (0.3,0.5]	[-0.7,-0.5), (0.5,0.7]	[-0.9,-0.7), (0.7,0.9]
Average	11.76	11.01	11.61	10.96	10.29
SD	2.19	2.24	2.31	3.35	3.75

for AV hardware-in-the-loop simulation at scale [31]. These platforms allow users to test, develop, and optimize AV systems in a cloud-based setting. Suraksha's concepts are general enough to be implemented on top of any of these platforms.

Safety models such as Responsibility-Sensitive Safety (RSS) [39] and Safety Force Field (SFF) [27] cross-check the AV's behavior with a separate model to avoid collisions and keep safe distance from potential hazards. Prior safety-aware AV system designs also proposed perception algorithms to improve safety [1], [16]. As machine learning (ML) models are extensively used in AVs, Sina et al. [23] reviewed safety challenges in the ML techniques, and propose methods to enhance safety of the ML applications. Researchers also proposed testing solutions to detect erroneous behaviors caused by DNNs that may result in safety hazards [35], [37], [41], [43], [49]. To test end-to-end safety of the AVs, prior work studied generating driving scenario based tests [8], [25]. However, these papers do not vary the component-level parameters to find insights to build a safer and efficient system. Two studies explore safety assessment for connected AVs [34], [47], and posit that safety can be improved if cars on the road can share information. In a recent study, Lin et al. [20] analyzed the effects of accelerating an AV system using GPUs, FPGAs, and ASICs and compared to CPU-only systems. This study analyzed how the latency changes with the use of these accelerators and the change in input images sizes. However, it lacked the capabilities to perform AV safety evaluation and comparison between different driving conditions.

Jahangirova et al. [12] performed a literature survey and collected 126 metrics to assess the driving quality of autonomous vehicles, and analyzed correlation between the metrics and the human perception of driving quality when AVs are driving. Codevilla et al. [6] investigated the relation between offline prediction accuracy and noted a weak correlation between the two. Haq et al. [9] performed an empirical study comparing offline and online testing of DNNs, and the results showed that (1) simulator-generated datasets are able to yield DNN prediction errors that are similar to real-life datasets; and (2) offline testing is more optimistic than online testing.

VIII. Conclusion

AV designers can benefit from analyzing the effects of varying component-level parameters on safety. In this paper, we propose an automated safety analysis framework, Suraksha, to explore an AV's component-level design choices using different driving scenarios generated based on a user-specified difficulty level. We employ Suraksha to analyze the safety implications of perception parameters that degrade the quality and present interesting insights that provide new information for AV designers to better optimize and design a safer AV.

References

[1] R. Aufrère, J. Gowdy, C. Mertz, C. Thorpe, C.-C. Wang, and T. Yata, "Perception for Collision Avoidance and Autonomous Driving," *Mechatronics*, vol. 13, no. 10, pp. 1149–1161, 2003.

[2] https://www.autoware.org/.

[3] "Baidu Apollo Autonomous Driving Platform," https://apollo.auto/developer.html.

[4] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on Over a Million Miles of Field Data," in *International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 586–597.

[5] T. Brell, R. Philipsen, and M. Ziefle, "sCARy! Risk Perceptions in Autonomous Driving: The Influence of Experience on Perceived Benefits and Barriers," *Risk Analysis*, vol. 39, no. 2, pp. 342–357, 2019.

[6] F. Codevilla, A. M. Lopez, V. Koltun, and A. Dosovitskiy, "On offline evaluation of vision-based driving models," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 236–251.

[7] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[8] A. Gambi, M. Mueller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 318–328.

[9] F. U. Haq, D. Shin, S. Nejati, and L. C. Briand, "Comparing offline and online testing of deep neural networks: An autonomous car case study," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 85–95.

[10] International Organization for Standardization, ""ISO 26262-1:2018 road vehicles — functional safety"," 2018.

[11] "ISO/DIS 21448," https://www.iso.org/standard/77490.html.

[12] G. Jahangirova, A. Stocco, and P. Tonella, "Quality metrics and oracles for autonomous vehicles testing," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2021, pp. 194–204.

[13] N. Kalra and S. M. Paddock, "Driving to Safety: How Many Miles of Driving Would it Take to Demonstrate Autonomous Vehicle Reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[14] P. Koopman and F. Fratrik, "How many operational design domains, objects, and events?" *SafeAI@ AAAI*, vol. 4, 2019.

[15] P. Koopman and M. Wagner, "Autonomous Vehicle Safety: An Interdisciplinary Challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90–96, 2017.

[16] F. Kunz, D. Nuss, J. Wiest, H. Deusch, S. Reuter, F. Gritschneder, A. Scheel, M. Stübler, M. Bach, P. Hatzelmann, C. Wild, and K. Dietmayer, "Autonomous Driving at Ulm University: A Modular, Robust, and Sensor-independent Fusion Approach," in *Intelligent Vehicles Symposium (IV)*, 2015, pp. 666–673.

[17] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams, "A Perception-driven Autonomous Urban Vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[18] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards Fully Autonomous Driving: Systems and Algorithms," in *Intelligent Vehicles Symposium (IV)*, 2011, pp. 163–168.

[19] G. Li, Y. Li, S. Jha, T. Tsai, M. Sullivan, S. K. S. Hari, Z. Kalbarczyk, and R. Iyer, "AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems," in *International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 25–36.

[20] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The Architectural Implications of Autonomous Driving: Constraints and Acceleration," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018, pp. 751–766.

[21] S. Liu, L. Li, J. Tang, S. Wu, and J.-L. Gaudiot, "Creating Autonomous Vehicle Systems," *Synthesis Lectures on Computer Science*, vol. 6, no. 1, pp. i–186, 2017.

[22] V. Milanés, S. E. Shladover, J. Spring, C. Nowakowski, H. Kawazoe, and M. Nakamura, "Cooperative Adaptive Cruise Control in Real Traffic Situations," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 1, pp. 296–305, 2013.

[23] S. Mohseni, M. Pitale, V. Singh, and Z. Wang, "Practical Solutions for Machine Learning Safety in Autonomous Vehicles," *arXiv preprint arXiv:1912.09630*, 2019.

[24] J. Moody, N. Bailey, and J. Zhao, "Public Perceptions of Autonomous Vehicle Safety: An International Comparison," *Safety Science*, vol. 121, pp. 634–650, 2020.

[25] G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, and S. K. Gupta, "Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles," *Journal of Systems and Software*, vol. 137, pp. 197–215, 2018.

[26] "NCAP Roadmap 2016-2020," https://www.euroncap.com/en/about-euro-ncap/timeline/roadmap-2016-2020.

[27] D. Nistér, H.-L. Lee, J. Ng, and Y. Wang, "The Safety Force Field," NVIDIA White Paper, 2019.

[28] "GeForce RTX 2080 Graphics Card — NVIDIA," 2020. [Online]. Available: https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2080/

[29] "The World's Most Powerful Graphics Card — NVIDIA TITAN V," 2020. [Online]. Available: https://www.nvidia.com/en-us/titan/titan-v/

[30] "NVIDIA DRIVE - Software — NVIDIA Developer," 2021. [Online]. Available: https://developer.nvidia.com/drive/drive-software

[31] "NVIDIA DRIVE Constellation — NVIDIA Developer," 2021. [Online]. Available: https://developer.nvidia.com/drive/drive-constellation

[32] "NVIDIA DRIVE Sim — NVIDIA Developer," 2021. [Online]. Available: https://developer.nvidia.com/drive/drive-sim

[33] On-Road Automated Driving (ORAD) committee, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, apr 2021. [Online]. Available: https://doi.org/10.4271/J3016_202104

[34] A. Papadoulis, M. Quddus, and M. Imprialou, "Evaluating the Safety Impact of Connected and Autonomous Vehicles on Motorways," *Accident Analysis & Prevention*, vol. 124, pp. 12–22, 2019.

[35] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[36] A. Reschka, "Safety Concept for Autonomous Vehicles," in *Autonomous Driving*. Springer, 2016, pp. 473–496.

[37] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 876–888.

[38] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta *et al.*, "LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving," *arXiv preprint arXiv:2005.03778*, 2020.

[39] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a Formal Model of Safe and Scalable Self-driving Cars," *arXiv preprint arXiv:1708.06374*, 2017.

[40] "Simcenter: ADAS & AV System Simulation," https://www.plm.automation.siemens.com/global/en/products/simulation-test/active-safety-system-simulation.html.

[41] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 359–371.

[42] G. Strang, *Introduction to Linear Algebra*. Wellesley-Cambridge Press Wellesley, MA, 1993, vol. 3.

[43] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[44] "UL 4600: Standard for Safety for the Evaluation of Autonomous Products," https://edge-case-research.com/ul4600/.

[45] J. Van Brummelen, M. O'Brien, D. Gruyer, and H. Najjaran, "Autonomous Vehicle Perception: The Technology of Today and Tomorrow," *Transportation Research Part C: Emerging Technologies*, vol. 89, pp. 384–406, 2018.

[46] J. Z. Varghese and R. G. Boone, "Overview of Autonomous Vehicle Sensors and Systems," in *International Conference on Operations Excellence and Service Engineering*, 2015, pp. 178–191.

[47] L. Ye and T. Yamamoto, "Evaluating the Impact of Connected and Autonomous Vehicles on Traffic Safety," *Physica A: Statistical Mechanics and its Applications*, vol. 526, p. 121009, 2019.

[48] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," *arXiv preprint arXiv:1906.05113*, 2019.

[49] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 132–142.

[50] H. Zhao, Y. Zhang, P. Meng, H. Shi, L. E. Li, T. Lou, and J. Zhao, "Towards Safety-Aware Computing System Design in Autonomous Vehicles," *arXiv preprint arXiv:1905.08453*, 2019.

[51] H. Zhao, Y. Zhang, P. Meng, H. Shi, L. E. Li, T. Lou, and J. Zhao, "Driving scenario perception-aware computing system design in autonomous vehicles," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 88–95.

[52] H. Zhao, Y. Zhang, P. Meng, H. Shi, L. E. Li, T. Lou, and J. Zhao, "Safety score: A quantitative approach to guiding safety-aware autonomous vehicle computing system design," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 1479–1485.