ADPerf: Investigating and Testing Performance in Autonomous Driving Systems

Tri Minh-Triet Pham
O-RISA Lab
Concordia University
Montreal, Quebec, Canada
p_triet@encs.concordia.ca

Diego Elias Costa
REALISE Lab
Concordia University
Montreal, Quebec, Canada
diego.costa@concordia.ca

Weiyi Shang
SENSE Lab
University of Waterloo
Waterloo, Ontario, Canada
wshang@uwaterloo.ca

Jinqiu Yang
O-RISA Lab
Concordia University
Montreal, Quebec, Canada
jinqiu.yang@concordia.ca

***Abstract*—Obstacle detection is crucial to the operation of autonomous driving systems, which rely on multiple sensors, such as cameras and LiDARs, combined with code logic and deep learning models to detect obstacles for time-sensitive decisions. Consequently, obstacle detection latency is critical to the safety and effectiveness of autonomous driving systems. However, the latency of the obstacle detection module and its resilience to various changes in the LiDAR point cloud data are not yet fully understood. In this work, we present the first comprehensive investigation on measuring and modeling the performance of the obstacle detection modules in two industry-grade autonomous driving systems, i.e., Apollo and Autoware. Learning from this investigation, we introduce *ADPerf*, a tool that aims to generate realistic point cloud data test cases that can expose increased detection latency. Increasing latency decreases the availability of the detected obstacles and stresses the capabilities of subsequent modules in autonomous driving systems, i.e., the modules may be negatively impacted by the increased latency in obstacle detection.**

We applied *ADPerf* to stress-test the performance of widely used 3D obstacle detection modules in autonomous driving systems, as well as the propagation of such tests on trajectory prediction modules. Our evaluation highlights the need to conduct performance testing of obstacle detection components, especially 3D obstacle detection, as they can be a major bottleneck to increased latency of the autonomous driving system. Such an adverse outcome will also further propagate to other modules, reducing the overall reliability of autonomous driving systems.

***Index Terms*—Autonomous vehicles, Lidar, Software testing, System testing, Simulation, Performance evaluation**

I. Introduction

Autonomous driving offers the potential for safer, more reliable, and more efficient transportation by reducing human error and enabling rapid, complex decision-making. At the core of any autonomous driving system (ADS) is the perception module, which is safety-critical. Leveraging algorithms and deep learning (DL) models, this module provides the ADS with real-time interpretation of its environment, allowing it to navigate safely using data from a variety of sensors, including LiDAR and cameras. As part of perception, obstacle detection performs the crucial task of identifying and localizing surrounding vehicles, pedestrians, etc. As a result, the accuracy of these detectors is critical, as any failure to correctly identify an obstacle could lead to critical safety issues, such as collisions. As such, a large body of research has been devoted to testing these components [1]. This includes testing detectors' robustness towards noises and weather corruptions [2], [3], adversarial perturbations [4]–[9], etc. on the sensor data. However, their focus has been on accuracy and robustness while the performance is understudied [1], [10].

The performance of obstacle detection is a critical yet underexplored area, directly influencing safety and reliability. Often referred to as availability or efficiency, this performance reflects the system's detection speed. High latency can impair decision-making, and delayed obstacle detection is as hazardous as an undetected one, leaving the ADS with insufficient time to respond. Addressing this challenge is vital for two reasons. First, the accuracy and efficiency of detection are highly sensitive to complex, unpredictable real-world workloads, which differ from the controlled datasets typically used in evaluations. Current workloads often fail to reveal performance issues, leaving potential vulnerabilities that could lead to unforeseen failures during deployment. However, existing performance testing research has primarily concentrated on traditional software systems, often overlooking the evaluation of complex AI system performance [11]. While recent works have analyzed the impact of adversarial perturbations on 3D [12] and 2D [13]–[16] obstacle detection, the performance of obstacle detection in particular and perception in general under normal circumstances is not yet fully understood. Furthermore, there remains a gap in availability-based robustness testing of obstacle detection.

To address these concerns, in this work, we present the first performance study analyzing and modeling two representative AD systems, i.e., Apollo [17] and Autoware [18]. First, we model the entire MSF obstacle detection module, which includes 2D and 3D obstacle detection and MSF, in the two AD systems using Queuing Networks to construct a Queuing Petri-Net (QPN) in QPME. Then, we exercise the obstacle detection modules using diverse driving scenarios (nuScenes [19] for Apollo and AWSIM [20] for Autoware) to compute the service rate of the components in the QPN. Afterwards, we use the measurement to configure the built performance models. Finally, we simulate them in QPME to identify throughput and bottleneck issues. Our simulation results reveal that 3D obstacle detection represents a critical bottleneck for detection, significantly hindering overall system performance.

Motivated by the modeling and simulation results, we introduce *ADPerf*, a novel tool designed to generate new performance tests by leveraging existing test scenarios (i.e., PCD) that test 3D obstacle detection. Furthermore, *ADPerf* records latency and computes data availability, and then evaluates the impacts of frame dropping on trajectory prediction. The outcome of *ADPerf* can be used to estimate frame availability, identifying which frames are dropped due to high latency. We conduct an evaluation using two widely-used LiDAR obstacle detection models in Level 4 AD systems: PointPillars in Apollo and CenterPoint in Autoware [18], using the nuScenes dataset. Our evaluation results show that *ADPerf* amplifies the latency of 3D obstacle detection, increasing the average latency by up to 11.5ms. This increase in latency leads to a 9% rise in the percentage of dropped frames. The resulting decline in 3D obstacle detection performance cascades downstream, causing significant deviations in trajectory prediction and underscoring the critical impact of latency on system reliability.

Our work makes the following contributions:

- We present the first study analyzing and modeling the performance of two Autonomous Driving Systems (Apollo and Autoware). Our analysis reveals the performance bottlenecks in the two ADSs.
- We propose a testing approach (*ADPerf*) that automatically generates latency-stressing tests for 3D obstacle detection.
- We applied *ADPerf* to examine the performance of Apollo and Autoware through three experiments. First, we evaluate the robustness of 3D obstacle detection, i.e., PointPillar and CenterPoint, and measure the change to latency and availability. Second, we evaluate the robustness of trajectory prediction, i.e., Trajectron++, to the unavailability of 3D obstacle detection outputs. Last, we experiment with the impact of the latencies of the 3D obstacle detection on the subsequent trajectory prediction module through a simulated evaluation.

Paper organization. Section II presents the background and related work. Section III introduces our experiment setup. Section IV presents the first ADS performance modeling study on both Apollo and Autoware. Section V introduces *ADPerf*, an automated performance test generation approach for ADS, followed by Section VI, a comprehensive evaluation of *ADPerf*. Section VII discusses the threats to the validity, and Section VIII concludes the paper.

II. Background and Related Work

In this section, we present the background of ADSs, with a focus on 3D obstacle detection and trajectory prediction. Then, we describe previous work that tests or attacks these modules in ADSs. Moreover, we discuss performance testing and attacking in the ADS obstacle detection context, as well as general performance testing and modeling.

At a high level, ADSs are composed of several functional modules, i.e., sensing, perception, prediction, planning, and control [1] as shown in Figure 1. The modules and their components communicate via messages/frames. The underlying middleware frameworks typically used are the Robot Operating System (ROS2) [21] (for Autoware) or CyberRT [22] (for Apollo). The communication is organized using a publisher-subscriber model for topics/channels. In Apollo, obstacle detection is a key part of perception, which consists of three components: 3D obstacle detection, 2D obstacle detection, and multi-sensor fusion (MSF). The output of MSF would be propagated to the rest of the ADS. In Apollo's MSF, LiDAR is the main sensor [23], meaning that the MSF process waits for a LiDAR scan before integrating data from other sensors [24], [25], such as cameras and radar. Consequently, the throughput of MSF is limited by the performance of 3D obstacle detection. Similarly, Autoware only performs 3D obstacle detection; hence, 3D obstacle detection throughput is obstacle detection throughput.

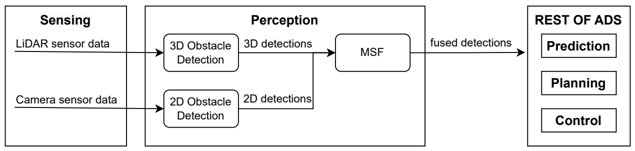


Fig. 1: The data flow between the modules (with a focus on sensing and perception components) in a general ADS. Arrows indicate data channels.

A. Detection robustness of 3D obstacle detection

ADS-mounted LiDAR scans the environment at regular intervals, producing point cloud data (PCD) where each point represents the 3D location of a reflected laser beam. 3D obstacle detection identifies obstacles in the PCD by fitting 3D bounding boxes around them.

The detection process is sensitive to various tests and attacks, such as the addition of random points outside the region of interest (ROI) in 3D obstacle detection [3], as well as weather and sensor-based corruptions to the PCD [2], [26]–[30]. Instead of modifying the sensed data, [31] fuzzed the simulated driving scenes to evaluate the robustness of the perception system in ADS. Most of these works focus on reducing the number of obstacles detected, thereby decreasing latency. In contrast, *ADPerf* is designed to generate additional detections, which has the potential to increase latency.

Previous 3D obstacle detection attacks have included spoofing obstacles [32], [33], perturbing vulnerable regions in PCD [8], [9], [34]–[36], conducting occlusion-guided optimization-based black-box attacks [37], [38], leveraging multiple sensors for GAN-based attacks [6], [7], generating adversarial obstacles [5], [39]–[42], or spoofing the ADS's trajectory [43]. Among these works, some increase the number of detected obstacles through physical attacks [32] or adversarial patches [33], [41]. However, these methods typically result in only a very small increase in detection, as they are difficult to execute and require object-specific patches. In contrast, our approach does not involve attacking 3D obstacle detection. Instead, we assess performance robustness by significantly increasing the number of detections to induce higher latency.

B. Trajectory prediction tests and attacks

Trajectory prediction forecasts the future trajectories of detected obstacles over the next few seconds based on their historical positions, headings, velocities, and other factors. Previous works testing this module have involved making obstacles appear or disappear for robustness testing [44] or generating safety-critical scenarios using LiDAR for system evaluation [45]. Attacks have included adversarial perturbations to the historical trajectory [46] or detection bounding boxes [47], as well as attention-guided perturbations aimed at disrupting pedestrian-based collision avoidance [48]. In contrast to existing works that test or attack trajectory prediction through perturbations to historical data or safety-critical scenarios, we approach trajectory prediction from the perspective of data availability, specifically examining the robustness of trajectory prediction when faced with data unavailability.

C. Performance testing

One key efficiency metric of a machine learning system is the prediction or inference speed [10]. In real systems, the prediction speed can become more important than model accuracy, particularly as data and system complexity grow [49]. In this work, we refer to efficiency as detection latency. Software performance testing assesses a system's performance under different conditions to ensure it meets requirements and operates efficiently at both the system and component levels. Performance testing measures a system's performance (e.g., response time, utilization, and throughput) under a specific workload [50], [51].

Model-based Performance Analysis. Model-based performance analysis involves constructing abstract models to predict and analyze the performance of software systems [52]. These models describe the system's components, communication, and resource utilization, facilitating simulation and analysis [53], [54]. Such models can be represented in various forms, each capturing different characteristics of the software, such as queueing networks (QN) [55], [56] and queueing Petri nets (QPN) [57]–[60]. These models offer an efficient way to understand the performance of software systems and identify potential bottlenecks under varying workloads. In this work, we model and simulate an ADS, enabling its performance analysis under different workloads.

Latency Robustness of Obstacle Detection. Most previous work has focused on the obstacle detection correctness instead of the performance. Of those, two studies propose adversarial attacks that generate perturbations to maximize the number of candidate proposals (K) in the post-processing stage, exploiting the $O(K^2)$ worst-case complexity of this step to maximize the latency for 2D obstacle detection [13]–[16] and 3D obstacle detection [12]. While both ADPerf and SlowLidar [12] perturb the PCD to increase latency, ADPerf adopts a more realistic approach. Unlike SlowLidar, which relies on an adversarial attack requiring multiple optimization steps and alters 3,000 points per iteration, ADPerf is a black-box testing method that introduces fewer than 670 modifications in a single step without depending on internal post-processing stages. Making fewer changes preserves the semantic integrity of the frame and aligns better with potential real-world or physical perturbations. We aim to observe the effect of noises and realistic modifications to the PCD on the latency of 3D obstacle detection.

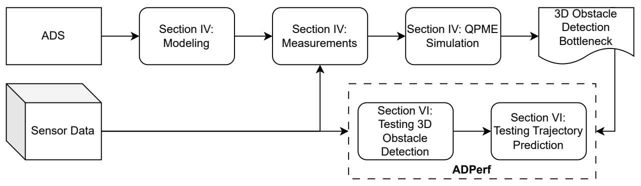

ADS
Section IV: Modeling
Section IV: Measurements
Section IV: QPME Simulation
3D Obstacle Detection Bottleneck
Sensor Data
Section VI: Testing 3D Obstacle Detection
Section VI: Testing Trajectory Prediction
ADPerf

Fig. 2: Overview of our experiment

III. Study Overview and Experimental Setup

Study Overview. The experiment consists of two stages. First, we study the performance of the MSF obstacle detection module in two ADS to identify the bottleneck, focusing on 3D obstacle detection. Second, we assess the robustness of 3D obstacle detection to latency-increasing tests and evaluate its cascading impact on the robustness of trajectory prediction. To confirm our findings, we simulate a scenario to demonstrate the effects of increased 3D obstacle detection latency on the entire ADS. We summarize the experiment in Figure 2.

Experiment Setup. Our experiments were conducted on an Ubuntu 20.04.3 LTS system with an AMD Ryzen 16-core CPU, 256 GiB of RAM, and an NVIDIA GeForce RTX 3090 GPU.

We set up two ADSs for the modeling experiment (Section IV), i.e., Autoware (RobotecAI's SS2 stable version) [61] and Apollo V7 [17], and along with them, two sources of input, i.e., V1.2.0 SS2 [20] and nuScenes [19]. While AWSIM is the best digital twin simulator for Autoware, we also chose nuScenes for our experiment because it is a popular dataset for ADS testing, which includes driving scenarios that require consecutive sensor inputs. Additionally, the tests for *ADPerf* necessitate obstacle annotations for those consecutive inputs, which datasets like KITTI [62] do not provide. nuScenes is a large-scale dataset featuring a wide range of complex driving scenarios for autonomous driving. It includes 1,000 20-second scenes collected in Boston and Singapore. The LiDAR sensor in nuScenes scans the environment at 20Hz, while the cameras capture images at 12Hz, offering a continuous and reliable stream of data for our tests.

For *ADPerf* evaluation (Section VI), we also use nuScenes for 3D obstacle detection and trajectory evaluation. For the demonstration, we reuse the above Autoware-AWSIM stack.

IV. Modeling the Performance of ADS

An ADS operates as a real-time system, where each frame must meet strict latency constraints. For instance, at a 10Hz frame rate, any frame exceeding 100ms to process is dropped. Consecutive frame drops lead to a loss of situational awareness, severely compromising driving decisions. However, the complex relationship between detection accuracy and latency under real-world workloads remains unexplored. We are interested in understanding the performance of AI-computation-heavy modules in an ADS and how the performance delays (i.e., latency) would further impact the functionality of an ADS. In this section, we describe how we employ the performance modeling technique [63], [64] to approach this problem.

First, we measure detection latency for Apollo on nuScenes and for Autoware on AWSIM with random scenarios to compute the respective service rates. Second, we model each ADS's obstacle detection component and simulate its performance under varying arrival and computed service rates. While the measurements reveal the relationship between latency and detection, they do not adequately cover detection behavior in scenarios such as frames with many detections or different sensors with varying arrival rates. In nuScenes, Apollo detects more than 50 obstacles in less than 0.3% of nearly 300,000 PCD frames, and in AWSIM random simulation, Autoware detects more than 20 obstacles in less than 0.2% of 36,000 PCD frames. This scarcity limits our ability to draw comprehensive conclusions about latency in high-density scenarios. Moreover, different ADS systems utilize diverse sensors with distinct arrival rates, even when using the same detection model. Therefore, measuring and modeling the detection module provides valuable insights into the effects of different detection configurations and scenarios, along with their potential adverse impacts.

A. Modeling methodology

In this subsection, we describe our modeling method for obstacle detection performance in ADSs. Our modeling method consists of four steps. First, we model obstacle detection using a QN [55], [56]. Second, we run a collection of experiments to measure the performance of Apollo and Autoware. Third, we determine the arrival rate (λ), the rate at which jobs enter a component, and the service rate (μ), the rate at which the component completes jobs, of each queue in the architecture model based on the collected experimental results. Finally, we transform the QNs into a QPN model for simulation.

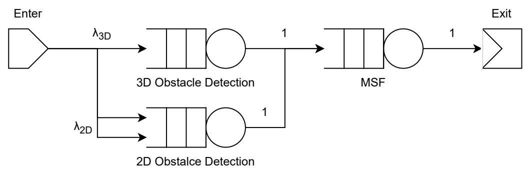
Step 1: Modeling the architecture of detection in ADS

We model the obstacle detection module in ADSs using an open QN with two types of workloads: PCDs from LiDAR sensors and images from cameras. We chose QN to model the ADSs because of their ability to capture multi-component interactions, handle real-world variability, and provide valuable insights into system performance, particularly to identify bottlenecks. Arrival rate (λ) for obstacle detection is set by the sensors' frame rate. For example, if the LiDAR scans the environment at 20Hz, the λ_{3D} for 3D obstacle detection is 20fps.

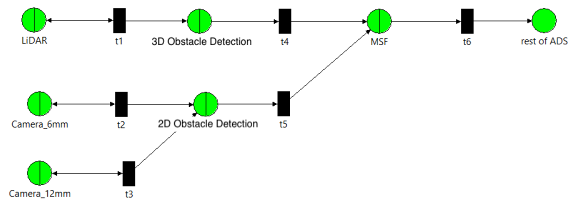
Table I presents the mapping between components from an ADS (Figure 1) to a QN model (Figure 3a), then to a QPN model (Figure 3b, presented later in step 4). We model components as queuing places and their channels as transitions. Each sensor is represented as a source node, resulting in three source nodes and one sink node for the workloads propagated to the rest of the ADS. Since the cameras share the same

TABLE I: Transformation from ADS's Detection to a QPN Model. Only immediate transitions are used.

Component	QN Element	QPN Element
Data Channel	Arc	Transition
Sensor Input	Source Node	Queuing Place, Transition
2D/3D Obs. Det., MSF	Queue	Queuing Place
Rest of ADS	Sink Node	Place
Message	Workload	Color



(a) The Queuing Network for Apollo MSF obstacle detection



(b) The QPN corresponding to the queueing network

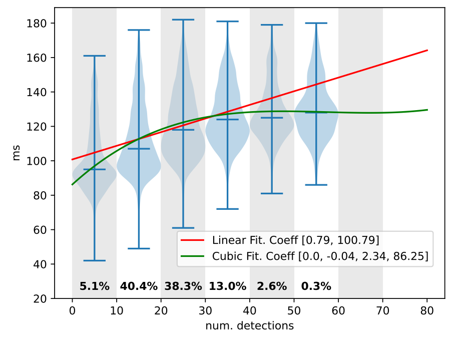
Fig. 3: Performance modeling of Apollo obstacle detection. Modeling figures for Autoware are included in the artifact.

processing queue, they also share the same type of workload. We model Apollo with three components: 2D detection, 3D detection, and MSF, and Autoware with a single component, 3D detection, as described in Section II. We exclude the radar-based detector as it is used mainly to detect very distant objects outside of the normal scope, i.e., prior works on testing MSF exclude radar-based detectors [26], [27], [41].

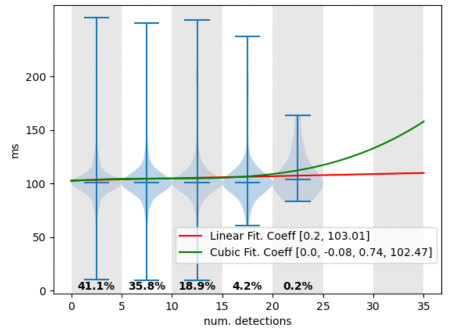
Step 2: Computing the service rate of each component

To set the architecture model parameters for performance modeling, we evaluate the live detection performance of Apollo and Autoware, measuring the relationship between the raw number of detections (regardless of IoU) and latency (ms). We run the ADS perception module with live sensor data and record the entry and exit timestamps of each message at every component. Using the native logging frameworks (ROS2 or CyberRT), which capture thousands of messages per run, ensures that the added logging overhead has a negligible impact on latency. Afterward, we calculate the average latency for each component based on the logged timestamps. For live sensor input, Apollo is evaluated using nuScenes dataset recordings (converted for playback), while Autoware is evaluated using AWSIM-generated random scenarios with varying numbers of vehicles.

Preparing nuScenes for Apollo replay. Since nuScenes format is incompatible with Apollo, we use `adataset` [65] (included in Apollo) to convert nuScenes to Apollo records. We map *LIDAR_TOP* to */apollo/sensor/lidar128/*



(a) Apollo–nuScenes



(b) Autoware–AWSIM random scenarios

Fig. 4: Measured latency of the 3D obstacle detection modules of two ADSs. Latencies are grouped into bins by the number of detections, with the percentage of frames in each bin indicated. Violin plots with marked medians illustrate the latency distribution within each bin. Regression lines reveal a positive correlation between detection count and latency, and are further used to predict latency for unseen frames.

compensator/PointCloud2 and *CAM_FRONT* to */apollo/sensor/camera/{front_6mm, front_12mm}/image*. The sensor data incompatible with Apollo's setup is dropped from the recording. When replayed in Apollo, the recorded data is fed through mapped channels, emulating real-time sensor input.

Measurement Results. We measure the latency of each available component, i.e., 2D obstacle detection, 3D obstacle detection, and MSF according to Figure 1. For example, the latency of 3D obstacle detection in Apollo measures the time elapsed between the input from */apollo/sensor/lidar128/compensator/PointCloud2* and the output to */apollo/perception/innerPrefused*. Hence, in addition to the three input channels, we monitor an additional two channels: */apollo/perception/innerPrefused* for both camera and LiDAR output and */apollo/perception/obstacles* for MSF output to compute latency. In Autoware, as it only performs 3D obstacle detection, we monitor /perception/object_recognition/detection/centerpoint/objects for the detections and lidar_centerpoint/debug/cyclic_time_ms for the latency to compute the frequency.

The latency recorded from all three components using a ten-scene sample from nuScenes (Figure included in the artifact) follows an approximately unimodal distribution, with average latencies of 8ms for 2D obstacle detection, 116ms for 3D obstacle detection, and 4ms for MSF, resulting in average service rates (μ) of 125fps for 2D obstacle detection and 250fps for MSF. This shows that obstacle detection's throughput is limited by the 3D obstacle detection service rate (described in Section II), which is expected to be 20fps based on the nuScenes arrival rate, such 2D obstacle detection and MSF service rate would bear a minimal impact on the performance of the system and is not a performance bottleneck.

On the other hand, 3D obstacle detection shows significantly higher latency, with an average service rate of 8.6fps, much lower than the PCD arrival rate of 20fps. Additionally, 52.7% of LiDAR frames were dropped during processing, indicating that the 20fps arrival rate exceeds Apollo's processing capacity. Further analysis of the 3D obstacle detection across the full dataset confirms the sample-based findings, with a service rate of 8.5fps. The delays are highly concentrated around the median, with between 70.6% and 79.4% of frames having latency within 20ms of the median. As shown by the linear and cubic regression in Figure 4a, there is a positive correlation between the number of detections and latency, suggesting that the latency increases as the number of detections grows.

For the AWSIM-Autoware setup, as Autoware ADS only performs 3D obstacle detection, we measured the average latency of the entire detection pipeline (Figure 4b) to be 128.1ms, resulting in the average service rate of 7.8fps where 6.8% of LiDAR frames are dropped.

Step 3: Configuring queues in the architecture model

To conduct a performance simulation with the architecture model that we built from step 1, we need to set the parameters of each component (e.g., queue in the architecture model), including the message processing latency, the workload arrival rate, and the queue size. We set up four different configurations reflecting different arrival and service rates.

Configuration 1 (Default setup Apollo). In (1) we configure the arrival rate λ_{2D} = 12fps for camera and λ_{3D} = 20fps for LiDAR per nuScenes capture rate, while the 2D, 3D obstacle detections and MSF have service rates of μ_{2D} = 125fps, μ_{3D} = 8.5fps, and μ_{MSF} = 250fps respectively as measured by our previous measurement step.

Configuration 2 (Low-workload setup Apollo). Since λ_{3D} = 20fps is double both Apollo and Autoware's default rate, which may result in a bottleneck due to significantly higher arrival rate, in (2), we keep the same service rates but modify the arrival rate λ_{2D} = 15fps for camera and λ_{3D} = 10fps for LiDAR according to the default rate to investigate how the model performs when the arrival rate is the default rate.

Configuration 3 (Low-latency setup Apollo). As the service rate for 3D obstacle detection, i.e., μ_{3D} = 8.5fps, is still lower than λ_{3D} = 10fps in the second setup, we are interested in a simulation with a higher service rate for 3D obstacle detection. Thus, we assume that 3D obstacle detection always detects fewer than five detections per the cubic regression shown in Figure 4a, which results in the μ_{3D} = 10.5fps, where the service rate is greater than the arrival rate.

Configuration 4 (Low-workload setup Autoware). Similar to (2), however, μ_{3D} = 7.8fps to reflect Autoware's measurements.

Since Autoware's 3D obstacle detection average service rate $\mu_{3D} = 7.8$ fps, and its average service rate for fewer than five detections $\mu_{3D} = 9.7$ fps, are both below 10 fps, we do not report results for configurations (1) and (3) on Autoware, as they would not yield additional insights.

In both Apollo and Autoware, only the newest message is processed while the rest is dropped; this effectively reduces the queue size to one. Therefore, we set the queue size to be one for all setups.

Step 4: Creating Queueing Petri Nets for simulation

We use QPME to simulate different load scenarios of detection in Apollo and Autoware, modeled with QPN to find bottlenecks in the setup. The transformation mapping is shown in Table I. The resulting QPN model for Apollo, shown in Figure 3b, consists of six queueing places and one ordinary place. Each server in the source QN model is mapped to a queueing place (FIFO) in the QPN model. The source nodes are modeled as individual queueing places with the Infinite Server scheduling strategy, initialized with one token. The service rate for each queueing place follows an exponential distribution, with rates described in step 3. The QPN model also includes an ordinary place representing the sink node, which corresponds to the remainder of the ADS. Two colors are used in the model, each representing a distinct workload from the source QN model. The QPN contains six transitions, each corresponding to an arc in the QN model, with each transition having a single mode and a firing weight of one.

B. Simulation results

During simulation, the following metrics are measured: throughput (X), mean service time (S), utilization (U), and mean token population (P). Note that since the queue size of all the queues in the architecture model is one (cf., step 3), any queue length above one would be considered an overflow.

The simulation is configured as follows: we set the *Max total run length* to 1,000,000, and the *Simulation stopping criterion* to *Fixed sample size*, meaning the simulation will run until the *Fixed sample size* is exhausted. These parameters are selected to ensure the model is tested with a sufficiently large number of inputs to evaluate throughput and identify bottlenecks. Table II shows that 2D obstacle detection and MSF are quite underutilized, and their arrival rates are well below the service rate. Table II presents the results for the configurations (described in step 3). From the results of configuration (1), we observe that 3D obstacle detection is a major bottleneck in the system. Since an ADS processes only the most recent message, any queue with more than one token causes an overflow, resulting in older frames being dropped. As shown, the average number of tokens (P) in the 3D obstacle detection queue is consistently greater than one by a significant margin. (Ideally, the simulation would capture the exact number of

TABLE II: QPME simulation results with different configurations: (1) Default setup, (2) Low-workload setup, and (3) Low-latency setup for Apollo and (4) Low-workload setup for Autoware. The following performance metrics are computed: throughput (X), mean service time (S), utilization (U), and mean token population (P). (*) shows overflows.

	Metric	Default Apollo	Low-workload Apollo	Low-latency Apollo	Low-workload Autoware
2D Obs. Det.	X	23.99	30.004	30.009	NA
	U	0.192	0.24	0.24	NA
	P	4.038	0.316	0.316	NA
	S (ms)	0.01	0.011	0.011	NA
3D Obs. Det.	X	8.497	8.49	9.997	7.803
	U	1.0	1.0	0.952	1.0
	P	574,828*	74,995*	20*	1,096,202*
	S (ms)	28,706*	7,500*	2.004*	109,651*
MSF	X	32.487	38.493	40.003	NA
	U	0.058	0.064	0.07	NA
	P	0.064	0.071	0.078	NA
	S (ms)	0.002	0.002	0.002	NA

tokens that overflow when the queue size is set to one, but this functionality is not operational.) Thus, we experimented with (2). While we observed significant improvements, the bottleneck remains. Configuration (3) demonstrates the system's performance with very few detections, specifically fewer than five. The results in the third column of Table II show that, although all measurements improve substantially, overflow still occurs. As LiDAR is the main sensor in Apollo (as described in Section II), overflown LiDAR messages result in processed but unfused detections by 2D obstacle detection, reducing the accuracy of MSF perception while wasting computation power for 2D obstacle detection.

For (4), which models Autoware, we observe similar results to (2). As Autoware only uses 3D obstacle detection, it does not represent a detection bottleneck. However, it means that the current servicing rate is not suitable for detecting obstacles at the current input rate, especially in dense scenarios.

Our experimental and simulation results show that, for 3D obstacle detection, the number of detections is positively correlated with latency. Since the input rate already exceeds the service rate for both Apollo and Autoware, 3D obstacle detection poses serious challenges in dense scenarios and can become a bottleneck, reducing the throughput of MSF ADSs such as Apollo.

V. *ADPerf*: Automated Performance Tests Generation for ADS

Our performance study reveals that bottlenecks in Apollo's obstacle detection are likely to emerge as detections become more diverse and complex, yet datasets (e.g., nuScenes) are not effectively in exposing performance bottlenecks. Performance modeling (Section IV) offers theoretical insights and guidance, but lacks realistic automated tests (i.e., driving scenarios).

To address the mentioned limitations, we propose *ADPerf*, the first automated framework for ADS performance testing. Unlike current accuracy-focused methods, *ADPerf* generates

TABLE III: *ADPerf* Tests Operators

Type	Direction	Distance (m)	% of each obstacle
Add Noises	$\pm$ y	0.1, 0.3, 0.5	5.8, 17.4, 29
Add Obstacles	$\pm$ y	3	100
Move Obstacles	$\pm$ y	0.1, 0.3, 0.5	100

simple and realistic latency-increasing test scenarios for 3D obstacle detection, the slowest component in obstacle detection. While multi-sensor synchronization is ideal for MSF-based testing, we focus on LiDAR inputs, as delaying 3D obstacle detection can stall the entire fusion pipeline and make detections from other sensors ineffective.

Overview of *ADPerf*. Figure 5 shows an overview of *ADPerf*. Leveraging existing real-world driving scenes (e.g., nuScenes) with ground-truth annotations (GT), *ADPerf* applies performance-specific mutation operators to each frame in existing scenes, producing new driving scenarios that expose performance issues when evaluated using an ADS.

ADPerf begins by extracting the point cloud representation (original PCD) for each obstacle in every frame of the dataset. It then modifies these obstacles to increase detection rates. After running 3D obstacle detection on the modified PCD frames, *ADPerf* measures latency, bounding box deviations, and the number of detected obstacles (calculated via intersection-over-union with GT). Using the latency data, *ADPerf* estimates dropped frames based on ADS frame-dropping rules and evaluates the prediction deviations caused by these drops, assessing their cascading impact on trajectory predictions.

A. Test Generation for 3D Obstacle Detection

1) Data preparation: Given an input dataset of real-world driving scenes, *ADPerf* extracts two types of obstacle data: point cloud representation in PCD and obstacle history data. First, it extracts the PCD scans and the point cloud representations for GT obstacles within the PCD to test 3D obstacle detection. In the nuScenes dataset, *ADPerf* can directly access the point cloud representation of the obstacles through the panoptic dataset to modify the points to generate test scenarios. It then uses trajectory prediction code to convert obstacle history data (visualized as consecutive frames in Figure 6) into a predictor-compatible format. In obstacle detection, each frame contains all present obstacles (schema A), while in trajectory prediction, each obstacle contains the frames in which it appears (schema B), enabling quick access to each obstacle's history.

2) Generating testing scenarios for 3D obstacle detection: In this work, we take a novel approach to robustness test generation by focusing on increasing the number of detections. To achieve this, we modify the LiDAR representations of obstacles based on the test scenario. Table III outlines the operators used to generate these testing scenarios.

- Adding Noises Outside of the Obstacles Bounding Boxes. Sensor data is noisy due to various factors such as environmental conditions, hardware limitations, or interference. In the context of obstacle detection, sensor noise can manifest as random variations and inaccuracies in the sensor readings.

In contrast to existing work, which adds noises to LiDAR representation of the obstacles (inside of the GT bounding boxes) [2], [26]–[29], we add noises right outside of the GT bounding boxes. These are simple, randomly generated points meant to increase the number of candidate objects detected by 3D obstacle detection by inflating the size of the obstacle. Since filtering these candidates is expensive (up to $O(K^2)$) as described in Section II-C, increasing the number of candidates has the most impact in increasing the latency while having an overall minimal impact on the detection results [12], [16].

We add noise outside each annotated obstacle through three steps: 1) identify noise distance, 2) calculate size, and 3) add noise. First, we determine the maximum distance (d_{noise}) from either the left or right of the obstacles to add noise. We experiment with three distances: 0.1, 0.3, and 0.5m. Then, we calculate the number of points to add $num_{noise} = num_{obs} \times \frac{d_{noise}}{width_{obs}}$ where num_{obs} represents the count of the point cloud representation of the obstacle and $width_{obs}$ represents the width of the obstacle. For example, assuming the average obstacle is similar to nuScenes' car (4.08m x 1.73m), then num_points_{noise} added at 0.1m to the obstacle in Figure 6c, is $698 * \frac{0.1m}{1.73m} = 40$ points. As d_{noise} increases, num_{noise} also increases following the formula to ensure that the added noise is not too sparse. Finally, *ADPerf* identifies a region R_{noise} of size $d_{noise} \times length_{obs}$ (where $length_{obs}$ is the length of the obstacle) immediately adjacent to either side of the obstacle. *ADPerf* upsample the existing points in the region to match num_{noise}. Gaussian noises are commonly used to simulate realistic environment noises [66]–[68]. Hence, *ADPerf* perturbs the newly added points with Gaussian noises of 0.05m in line with previous works [2], [26]. The noises added (P_{noise}) can be summarized by the formula:

$$P_{noise} = \{p + normal(0,1) \times 0.05m \mid p \in upsample(R_{noise}, num_{noise})\} \quad (1)$$

where $upsample(R_{noise}, num_{noise})$ represents the up sampled noise from R_{noise}, $normal(0,1)$ represents Gaussian noises with $\mu = 0$ and $\sigma^2 = 1$, $0.05m$ is the scale of the noise and p represents individual point from the upsampled R_{noise} region.

In nuScenes, we add fewer than 670 points, far fewer than state-of-the-art latency-inducing adversarial methods [12].

- Adding Obstacles. The next test involves adding new obstacles to the PCD, motivated by the findings in Section IV. As the number of obstacles increases, we aim to observe a corresponding rise in latency, confirming their relationship. These obstacles are selected from the existing obstacles in each frame to ensure realism. The new obstacles are placed two car widths away from the existing ones, for two reasons: First, to minimize the impact of different capture angles on obstacles at varying distances, we avoid placing new obstacles too far away. Second, the new obstacle must account for both the original and its own width while maintaining a small gap.

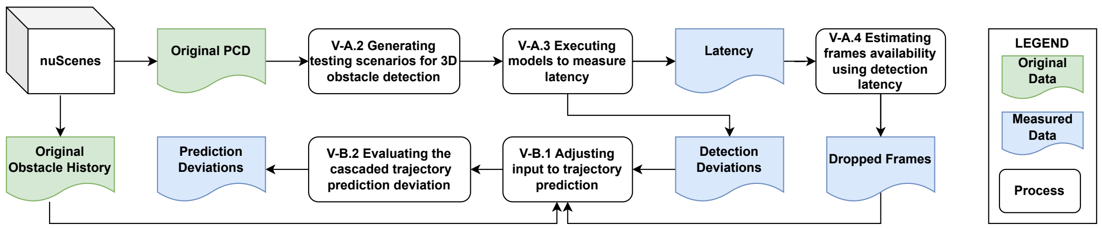
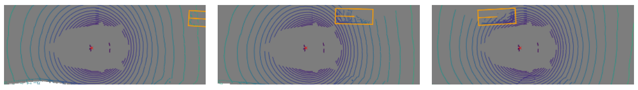

nuScenes
Original PCD
V-A.2 Generating testing scenarios for 3D obstacle detection
V-A.3 Executing models to measure latency
Latency
V-A.4 Estimating frames availability using detection latency
LEGEND
Original Data
Measured Data
Process
Original Obstacle History
Prediction Deviations
V-B.2 Evaluating the cascaded trajectory prediction deviation
V-B.1 Adjusting input to trajectory prediction
Detection Deviations
Dropped Frames

Fig. 5: Overview of the *ADPerf*, showing how the data flows from the beginning (nuScenes).



(a) Frame 4 (b) Frame 5 (c) Frame 6

Fig. 6: Visualization of three PCD frames showing a moving obstacle car with the orange GT bounding box. The car moves towards the ego car. The size (in point cloud) at each frame is 56, 518, and 698, respectively.

To simplify the calculations, we assume that the cars have the same width as those in the nuScenes dataset, i.e., 1.73m. As a result, the new obstacles are placed approximately three meters away from the original ones. However, the obstacle can be placed at a different location if needed. The selection and checking process is repeated each frame to maintain plausibility over the entire scene.

- Moving Obstacles. The final test involves moving existing obstacles in the current frame. We explore moving the obstacles closer together. The objective is to challenge the detection process by causing the obstacles' point cloud representations to be close but not overlap, thereby confusing the system about the number of obstacles. To do this, we first compute the center of mass using the LiDAR representation of all obstacles in each frame, then shift them toward the y-center to assess the impact of clustered obstacles on detection latency.

3) Executing models to measure latency: We run the 3D obstacle detection models on both the unaltered and modified PCD frames for comparison. The models are executed with a single worker, and the data frames are not shuffled to maintain the original ordering. The latency between the model's input and output is measured.

4) Estimating frames availability using detection latency: Since the 3D obstacle detection models are run via OpenPCDet and not within an active ADS (e.g., Apollo), frames are not automatically dropped. As a result, we estimate data availability (i.e., which frames are dropped) based on detection latency. Our research into how existing ADSs handle latency reveals two approaches to managing high latency. In Apollo, incoming messages are stored in a FIFO queue. After the current message is processed, the newest message is selected for processing, while any intermediate messages are considered outdated and dropped [69]. In Autoware, there is no general rule, but the point cloud preprocessor documentation mentions a timer to manage high latency [70]. Since this timer is highly specific and requires tuning for each frame rate, we adopt Apollo's approach, as it is more widely applicable.

Using the sensor message arrival rate (λ_{sensor}), *ADPerf* can compute the expected processing time for each component. For instance, if the sensor message arrival rate is 20Hz (as in nuScenes), the expected processing time for the corresponding component is $\frac{1}{20\,\text{Hz}} = 50\,\text{ms}$. The delay ($delay_{frame_i}$) for each processed message is computed as the latency exceeding the expected processing time. Since the frames arrive at a fixed rate, *ADPerf* only considers positive delays.

$$delay_{frame_i} = maximum(0, latency_{frame_i} - \frac{1}{\lambda_{sensor}}) \quad (2)$$

Using this formula, *ADPerf* computes the delays for all frames ($delay\_by\_frame$). Then, the availability for each frame can be estimated based on the accumulated delay (Algorithm 1).

Algorithm 1: *ADPerf*'s Frame Drop Computation

```
Data: $delay\_by\_frame$, $theshold$
Result: indices of points dropped
1  $accm\_delay \leftarrow 0$  $dropped\_frames \leftarrow Empty\ list$;
2  for each $frame\_index, delay$ in $delay\_by\_frame$ do
3      if $frame\_index$ is start of new scene then
           /* restart for each scene */
4          $accm\_delay \leftarrow 0$;
5      end
6      if $accm\_delay \geq threshold$ then
           /* adjust acc. delay and save alarm */
7          $accm\_delay \leftarrow Max(0, accm\_delay - threshold)$;
8          Append $frame\_index$ to $dropped\_frames$;
9      end
10     else
11         $accm\_delay \leftarrow accm\_delay + delay$;
12     end
13 end
```

B. Test Generation for Trajectory Prediction

1) Adjusting input to trajectory prediction: We aim to assess the impact of dropped frames from 3D obstacle detection on trajectory prediction. Before doing so, we must adjust the GT to reflect the PCD modifications-induced changes. Without this adjustment, two issues arise: first, the frame drop may not occur without the PCD modification, and second, deviations may be inaccurate if obstacles are moved but not accounted for. For instance, if the deviation is reported as 2m, but we moved the obstacle by 2m, there would be no actual deviation.

Calculating the deviations from obstacle detection and matching them to the correct GT obstacles. For each modified frame, we record latency, detections, and positions to compute the new IoUs and GT matches. The detected obstacles are matched to the trajectory prediction input in two steps. First, for each frame, *ADPerf* adjusts the GT obstacles based on the generated test. For adding noise, the original GT remains unchanged. For moving obstacles, the GT is shifted according to the corresponding changes in the PCD. Added obstacles are ignored, as they do not exist in the GT. This enables *ADPerf* to match the detected obstacles from the modified PCD to the modified GT obstacles using IoU, allowing the computation and linkage of detection deviations to the corresponding GT. We consider two types of deviations, i.e., status and displacement. For status deviations, we only consider when GT obstacles are **not** detected. When GT obstacles are detected, we consider their displacement. These deviations are used to adjust the trajectory prediction input.

Adjusting the input of trajectory prediction using the calculated deviations. To transfer the observed deviations from the previous step, we first match the obstacles in the two different data schemas using GT and frame ID. For each deviation type described, we handle the obstacles differently. If the histories are too short or fragmented, trajectory calculations would either be impossible or highly inaccurate. Repeating the status strikes a balance between maintaining calculable and reasonably accurate obstacle trajectories. For displacements, *ADPerf* shifts the obstacles in the trajectory prediction input according to the displacement values.

2) Evaluating the cascaded trajectory prediction deviation: After adjusting the previous step, we can proceed with frame dropping and estimate the impact. Since we use datasets containing scenes with consecutive data frames, we can drop the exact frames where data is expected to be dropped based on obstacle detection latency. Following Apollo's rule, we calculate the frames to drop as outlined in Subsection V-A4. To drop these frames from the trajectory prediction input, we iterate over each obstacle and each frame. If the index of the frame matches a dropped frame, we set its values to those of its previous frame. If all frames of an obstacle are dropped, we set its values in all frames to be those of the first frame, as if the obstacle is stationary throughout the scene. Once we run trajectory prediction with the corresponding frames dropped from the calculation, we obtain a list of obstacles and their predicted trajectory. Comparing the newly generated trajectory with that when no frames are dropped, we can obtain the deviation to analyze the impact of increased latency.

VI. Evaluating ADS Performance Using *ADPerf*

This evaluation contains three experiments. First, we evaluated the robustness of two 3D obstacle detection models, i.e., PointPillar and CenterPoint, against the latency-stressing scenarios by *ADPerf*. Second, we investigate how the high-latency obstacle detection may affect trajectory prediction. Finally, we demonstrated in simulation that increased 3D obstacle detection latency cascades through the ADS, leading to adverse outcomes such as indefinite stops.

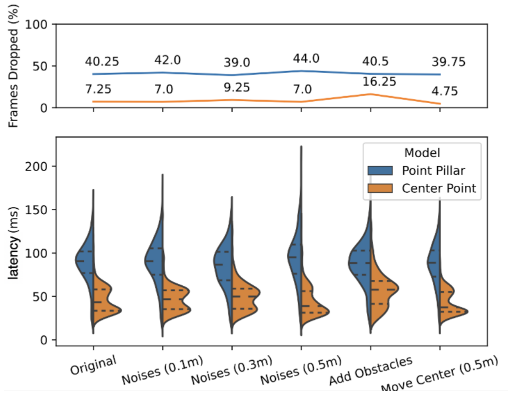


Fig. 7: Effects of different tests on model latency distributions. The violin plots show quartiles (dashed lines), and the linear plot indicates the percentage of frames dropped.

A. Testing 3D obstacle detection availability robustness to latency increasing scenarios

Motivation. From the previous experiment, we identify 3D obstacle detection as a performance bottleneck in ADS, with latency increasing as the number of obstacles grows. This motivates an investigation into its resilience to noise and variations, as its robustness directly impacts system throughput and real-world reliability.

Method. We assess the robustness of 3D obstacle detection in ADSs to noise and changes while measuring the sensitivity of latency through three simple and realistic tests: adding noise, adding obstacles, and moving obstacles in the PCD from nuScenes. These mutations are carried out following the procedures outlined in subsections V-A1, V-A2, V-A3, and V-A4. For the 3D obstacle detection, we select two models from the OpenPCDet model zoo [71], both trained on nuScenes data: PointPillar-MultiHead (PointPillar) and CenterPoint-PointPillar (CenterPoint). These models were chosen for their role in well-known ADSs: PointPillar in Apollo and CenterPoint in Autoware.

To evaluate whether the observed increased latencies with *ADPerf*'s tests are statistically significant, we apply non-parametric methods that account for the distribution of the data. We applied both Cliff's delta (δ) [72] and Wilcoxon signed-rank test [73] to compare the two sets of latency values, i.e., the original latency values and the ones under each perturbation type.

Results. Overall, PointPillar generates 84 fewer raw detections on average than CenterPoint for the same number of detected obstacles. Despite this, PointPillar has significantly higher latency, with the average median latency 41ms higher and the maximum latency 43ms higher, as shown in Figure 7. The linear graph also shows that obstacle detection using PointPillar causes considerably more frame drops, where the median latency is 88-95ms. Additionally, PointPillar shows a smaller increase in detections from the tests. However, the increase in detections has a more significant impact on latency.

Both models exhibit latency sensitivity to the generated testing scenarios, albeit in different ways. PointPillar is more sensitive to added noise (mean latency increases up to 6.2ms and the frame drop rate increases up to 3.75%), while it remains relatively resistant to added obstacles (latency decreases by 0.4ms and frame drop rate increases 0.25%). In contrast, added noises increase the average latency up to 3.9ms and the number of frames dropped up to 2% in CenterPoint, while added obstacles increase the mean latency by 11.5ms and the percentage of dropped frames to 16.25%. Both models exhibit strong resistance to moving obstacles, with the frame drop rate decreasing by 4.25% for PointPillar and 3.5% for CenterPoint.

Statistical Analysis. For PointPillars, two types "noise (0.3m)" and "noise (0.5m)" show statistically significant increased latencies ($p-value$ is 0.02 and < 0.001 respectively) with small effect size ($r = 0.12, 0.20$ respectively). Cliff's delta confirmed the small effect for noise (0.5m) ($|\delta| = 0.15$), while indicating negligible effects for the remaining PointPillars tests.

For CenterPoint, based on the Wilcoxon test, three types, i.e., "noise (0.3m)", "noise (0.5m)", and "moving center (0.5m)" led to significant latency increases ($p < 0.001$, $p = 0.03$, and $p < 0.001$, respectively), with a small effect size ($r = 0.19$, $r = 0.12$, $r = 0.15$). "adding obstacles" produced a highly significant increase ($p < 0.001$) with a medium effect size ($r = 0.48$). Only "noise (0.1m)" failed to produce a significant difference. This is also supported by Cliff's delta: except for noise at (0.1m), which showed negligible effects, noise (0.3m), noise (0.5m), and moving center (0.5m) yielded small effects ($|\delta| = 0.16$, $|\delta| = 0.15$, $|\delta| = 0.16$), while adding obstacles produced a medium effect ($|\delta| = 0.35$).

This experiment reveals contrasting model behaviors under different scenarios. PointPillar is more sensitive to 2 of 3 noise types but handles crowded scenes better, while CenterPoint is less affected by noise but more sensitive to added obstacles.

Each obstacle detector has its inherent latency, which *ADPerf*'s testing scenarios can further amplify. *ADPerf* triggered opposite responses from the detectors, with each being sensitive to different scenarios. Depending on the specific scenarios the system is often used in, the appropriate detector should be selected for optimal performance.

B. Testing trajectory prediction robustness to unavailable detection output

Motivation. Having learned that certain modifications can impact latency, we are eager to explore how these changes affect trajectory prediction, which relies on obstacle detection as its input. Understanding this relationship is crucial, as the unavailability of detection output can significantly influence the accuracy and reliability of trajectory predictions, ultimately shaping the effectiveness of the entire ADS.

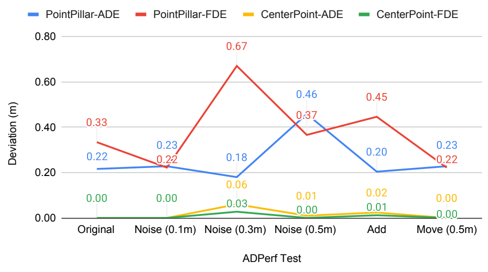


Fig. 8: Average ADE and FDE due to data unavailability caused by *ADPerf*'s tests (Figure 7). Only ADE and FDE of moving obstacles are considered.

Method. The changes observed in perception are two-fold: altered detections and their associated latency can influence the prediction results. As a result, we adjust the input as described in Subsection V-B1. We use Trajectron++ [74], a well-established trajectory prediction, which is commonly evaluated in prior works [46], [75], [76]. We measure trajectory deviation using Average Displacement Error (ADE) and Final Displacement Error (FDE), commonly used in prior works [46], [74]. ADE reflects overall accuracy, while FDE measures the precision of the final position. Last, we applied statistical tests to highlight statistical significances when comparing the two metric values between the original and under perturbations by *ADPerf*.

Results. Figure 8 shows that despite high frame drop rates, trajectory deviations remain small, indicating Trajectron++'s resilience to missing data, though some trends still emerge.

Overall, the trajectory prediction deviations using CenterPoint's detection output are significantly smaller than those with PointPillar. Added noise contributes to the largest increase in ADE (0.06m) and FDE (0.03m). With fewer frames dropped from CenterPoint detection (indicating higher availability), trajectory prediction using its output is more resistant to deviations. On the other hand, the trajectory deviations resulting from PointPillar's detection are significantly larger. Adding noises increases the ADE up to 0.24m and FDE up to 0.34m. Additionally, adding obstacles leads to a 0.12m increase in FDE. The limited effect on trajectory prediction can be partly attributed to the availability distribution of detection frames. Even under high drop rates, at most two consecutive frames are lost. At 10fps, sufficient temporal information remains available for the prediction module, minimizing the overall impact on trajectory estimation.

Our statistical tests show that the prediction trajectory deviations caused by *CenterPoint*'s increased latencies are not statistically significant. Differently, for *PointPillar*, under all the perturbation types, except for "noise (0.5m)", ADE and FDE are statistically significant ($p < 0.02$) with large effect sizes ($0.83 < r < 0.87$). Such statistical significances are also supported by Cliff's test.

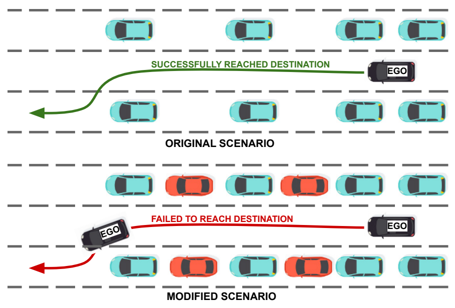


Fig. 9: An example scenario in the system demonstration, i.e., exercising Autoware in AWSIM using one latency-stressing scenario by *ADPerf*. *ADPerf* generates a new scenario by adding the red vehicles.

Deviations to predicted trajectories can result in unsafe trajectory plans or misleading control decisions, causing path deviation. Such system failures are ultimately linked to real-world collisions as documented in public reports [1].

> Trajectory prediction demonstrates resilience to low data availability. However, the availability of data from obstacle detection significantly impacts deviations. Therefore, trajectory prediction should prioritize detectors with higher data availability.

C. System demonstration: applying ADPerf to test ADSs in simulated environments

Motivation. Prior studies [77], [78] illustrate that non-trivial gaps exist between testing individual modules alone and testing the whole ADS (e.g., through a simulator). Hence, we set off to demonstrate the impact of increased latency of 3D obstacle detection latency detected by *ADPerf* on an ADS in a simulated environment.

Method. We conduct system demonstrations on Autoware SS2 [61] using a compatible simulator (i.e., AWSIM V1.2.0 [20]). We cannot experiment on Apollo due to the lack of a compatible simulator. AWSIM supports a very limited driving scenario compared to nuScenes, which was used in the other evaluations of this work. We experimented with different scenarios (e.g., mutating obstacles) but found similar outcomes. In the paper, we focus on one representative demo as an example: In the original scenario (Figure 9), the ego car needs to travel 60m, along the lane to its destination on the adjacent lane, which means the ego car needs to plan a route for lane changing at a proper time. Eight vehicles are on the adjacent lanes moving in the same direction. The ego car successfully arrived at the destination in 30 seconds.

We then applied *ADPerf* to generate new scenarios that stress the latency of 3D obstacle detection, and further impact the whole ADS. Limited by space, we will describe one type of perturbation by *ADPerf*, which is "add obstacle".

Results. Under the *ADPerf*'s generated scenarios (i.e., with four non-overlapping vehicles on the adjacent lane), the ego car stops indefinitely and fails to change the lane when it is safe (the bottom figure in Figure 9).

On the surface, we found that an increased number of high-latency (i.e., over 100ms) single-frame processing is observed: from 13.8% to 23% of all the frames. High-latency frames are dropped by a real-time AD system. Hence, this translates to a sharp increase in frame dropping. Even though the accuracy of the obstacle detection is consistent between the original and *ADPerf*'s generated scenarios. The *ADPerf*'s generated scenario stresses the latency of the obstacle detection, promotes more frame dropping, and causes the ego car to "freeze."

Root cause analysis. We further examine the full story behind the ego car's erroneous behavior. We found that due to the increased frame dropping, some obstacles of the dropped frames are deemed as *disappeared* by the *tracking* module. An obstacle must be continuously (i.e., over one second) not detected to be deemed as "disappeared". That exact obstacle in the next frame will have a brand-new status for trajectory prediction, i.e., no history locations, which leads to a poor predicted trajectory. Significant discrepancy between a trajectory and the detected location will lead to frequent replanning, which is computationally expensive. Therefore, the ego car will "freeze" once it begins frequent replanning.

VII. Threats to Validity

Experiment reproducibility. As live performance measuring can be fickled due to randomization of the algorithms preprocessing data for the detection models, for each PCD modification, we evaluate it with the model five times to record the combined latency and enhance reproducibility.

Scenes contain a large percentage of stopped cars. The scenes used to test trajectory prediction contain a low percentage of moving cars (18.5%), which skews the results by reducing the number of deviations and limiting the variability in the computed deviations. We plan to address this limitation in future studies.

VIII. Conclusion

In this paper, we present a study to investigate the performance of two ADSs that identify the bottleneck in 3D obstacle detection that can be further tested for robustness. We introduce *ADPerf*, a tool that generates testing scenarios to assess the robustness of 3D obstacle detection and its cascading effects on other components in the ADS, such as trajectory prediction. Experimental results show that the scenarios generated by *ADPerf* can cause significant delays in 3D obstacle detection, leading to non-trivial deviations in the entire ADS. The generated QPME model and code for ADPerf are available at https://github.com/anonfolders/adperf.

References

[1] S. Tang, Z. Zhang, Y. Zhang, J. Zhou, Y. ling Guo, S. Liu, S. Guo, Y. Li, L. Ma, Y. Xue, and Y. Liu, "A survey on automated driving system testing: Landscapes and trends," *ACM Transactions on Software Engineering and Methodology*, vol. 32, pp. 1 – 62, 2022.

[2] Y. Dong, C. Kang, J. Zhang, Z. Zhu, Y. Wang, X. Yang, H. Su, X. Wei, and J. Zhu, "Benchmarking robustness of 3d object detection to common corruptions in autonomous driving," *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1022–1032, 2023.

[3] Z. Q. Zhou and L. Sun, "Metamorphic testing of driverless cars," *Commun. ACM*, vol. 62, no. 3, p. 61–67, feb 2019.

[4] S.-T. Chen, C. Cornelius, J. Martin, and D. H. P. Chau, "Shapeshifter: Robust physical adversarial attack on faster r-cnn object detector," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2018, p. 52–68.

[5] Y. Cao, N. Wang, C. Xiao, D. Yang, J. Fang, R. Yang, Q. A. Chen, M. Liu, and B. Li, "Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks," in *2021 IEEE Symposium on Security and Privacy (SP)*, May 2021, pp. 176–194.

[6] Z. Xiong, H. Xu, W. Li, and Z. Cai, "Multi-source adversarial sample attack on autonomous vehicles," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 3, pp. 2822–2835, 2021.

[7] B. Liu, Y. Guo, J. Jiang, J. Tang, and W. Deng, "Multi-view correlation based black-box adversarial attack for 3d object detection," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1036–1044.

[8] X. Xu, J. Zhang, Y. Li, Y. Wang, Y. Yang, and H. T. Shen, "Adversarial attack against urban scene segmentation for autonomous vehicles," *IEEE Transactions on Industrial Informatics*, vol. 17, pp. 4117–4126, 2021.

[9] Z. Hau, K. T. Co, S. Demetriou, and E. C. Lupu, "Object removal attacks on lidar-based 3d object detectors," *ArXiv*, vol. abs/2102.03722, 2021.

[10] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2022.

[11] M. Jangali, Y. Tang, N. Alexandersson, P. Leitner, J. Yang, and W. Shang, "Automated generation and evaluation of jmh microbenchmark suites from unit tests," *IEEE Transactions on Software Engineering*, vol. 49, pp. 1704–1725, 2023.

[12] H. Liu, Y. Wu, Z. Yu, Y. Vorobeychik, and N. Zhang, "Slowlidar: Increasing the latency of lidar-based detection using adversarial examples," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 5146–5155.

[13] A. Shapira, A. Zolfi, L. Demetrio, B. Biggio, and A. Shabtai, "Phantom sponges: Exploiting non-maximum suppression to attack deep object detectors," *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 4560–4569, 2022.

[14] E.-C. Chen, P.-Y. Chen, I.-H. Chung, and C.-R. Lee, "Overload: Latency attacks on object detection for edge devices," *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 24 716–24 725, 2023.

[15] C. Ma, N. Wang, Q. A. Chen, and C. Shen, "Slowtrack: Increasing the latency of camera-based perception in autonomous driving using adversarial examples," *ArXiv*, vol. abs/2312.09520, 2023.

[16] Y. Xiao, J. Ma, P. Yi, and X. Chen, "Sponge backdoor attack: Increasing the latency of object detection exploiting non-maximum suppression," in *2024 International Joint Conference on Neural Networks (IJCNN)*, 2024, pp. 1–8.

[17] Baidu. (2025) Apollo. [Online]. Available: https://github.com/ApolloAuto/apollo/

[18] T. A. Foundation. (2025) Autoware. [Online]. Available: https://autoware.org/

[19] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *CVPR*, 2020.

[20] T. IV. (2023) Awsim. [Online]. Available: https://github.com/RobotecAI/autoware-1/tree/awsim-ss2-stable

[21] D. Thomas, W. Woodall, and E. Fernandez, "Next-generation ROS: Building on DDS," in *ROSCon Chicago 2014*. Mountain View, CA: Open Robotics, sep 2014.

[22] Baidu. (2025) Apollo cyber rt. [Online]. Available: https://github.com/ApolloAuto/apollo/tree/master/cyber

[23] ——. (2025) Apollo. [Online]. Available: https://github.com/ApolloAuto/apollo/blob/master/modules/calibration/\\data/mkz_lgsvl_321/sensor_meta.pb.txt

[24] ——. (2025) Apollo. [Online]. Available: https://github.com/ApolloAuto/apollo/blob/master/modules/perception/\\multi_sensor_fusion/multi_sensor_fusion_component.cc

[25] ——. (2025) Apollo. [Online]. Available: https://github.com/ApolloAuto/apollo/blob/master/modules/perception/\\multi_sensor_fusion/fusion/fusion_system/probabilistic_fusion/probabi\\listic_fusion.cc

[26] K. Yu, T. Tao, H. Xie, Z. Lin, T. Liang, B. Wang, P. Chen, D. Hao, Y. Wang, and X. Liang, "Benchmarking the robustness of lidar-camera fusion for 3d object detection," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2023, pp. 3188–3198.

[27] X. Gao, Z. Wang, Y. Feng, L. Ma, Z. Chen, and B. Xu, "Benchmarking robustness of ai-enabled multi-sensor fusion systems: Challenges and opportunities," *ArXiv*, vol. abs/2306.03454, 2023.

[28] S. Li, Z. Wang, F. Juefei-Xu, Q. Guo, X. Li, and L. Ma, "Common corruption robustness of point cloud detectors: Benchmark and enhancement," *IEEE Transactions on Multimedia*, pp. 1–12, 2023.

[29] A. Guo, Y. Feng, and Z. Chen, "Lirtest: Augmenting lidar point clouds for automated testing of autonomous driving systems," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 480–492.

[30] T. M. T. Pham, B. Yang, and J. Yang, "Evaluating the robustness of lidar-based 3d obstacles detection and its impacts on autonomous driving systems," 2024. [Online]. Available: https://arxiv.org/abs/2408.13653

[31] ——, "Perception-guided fuzzing for simulated scenario-based testing of autonomous driving systems," 2024. [Online]. Available: https://arxiv.org/abs/2408.13686

[32] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, and Z. M. Mao, "Adversarial sensor attack on LiDAR-based perception in autonomous driving," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, nov 2019.

[33] H. Shin, D. Kim, Y. Kwon, and Y. Kim, "Illusion and dazzle: Adversarial optical channel exploits against lidars for automotive applications," in *Cryptographic Hardware and Embedded Systems – CHES 2017*, W. Fischer and N. Homma, Eds. Cham: Springer International Publishing, 2017, pp. 445–467.

[34] S. Zheng, W. Liu, S. Shen, Y. Zang, C. Wen, M. Cheng, and C. Wang, "Adaptive local adversarial attacks on 3d point clouds," *Pattern Recognition*, vol. 144, p. 109825, 2023.

[35] Y. Zhu, C. Miao, T. Zheng, F. Hajiaghajani, L. Su, and C. Qiao, "Can we use arbitrary objects to attack lidar perception in autonomous driving?" in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1945–1960.

[36] Y. Zhu, C. Miao, F. Hajiaghajani, M. Huai, L. Su, and C. Qiao, "Adversarial attacks against lidar semantic segmentation in autonomous driving," *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021.

[37] J. Sun, Y. Cao, Q. A. Chen, and Z. M. Mao, "Towards robust lidar-based perception in autonomous driving: General black-box adversarial sensor attack and countermeasures," in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20. USA: USENIX Association, 2020.

[38] X. Wang, M. Cai, F. Sohel, N. Sang, and Z. Chang, "Adversarial point cloud perturbations against 3d object detection in autonomous driving systems," *Neurocomputing*, vol. 466, pp. 27–36, 2021.

[39] K. Yang, T. Tsai, H. Yu, M. Panoff, T.-Y. Ho, and Y. Jin, "Robust roadside physical adversarial attack against deep learning in lidar perception modules," *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021.

[40] M. Abdelfattah, K. Yuan, Z. J. Wang, and R. Ward, "Towards universal physical attacks on cascaded camera-lidar 3d object detection models," in *2021 IEEE International Conference on Image Processing (ICIP)*, 2021, pp. 3592–3596.

[41] 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," in *Proceedings of the 5th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Faust, D. Hsu, and G. Neumann, Eds., vol. 164. PMLR, 08–11 Nov 2022, pp. 1013–1024.

[42] 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 (CVPR)*, June 2020.

[43] Y. Li, C. Wen, F. Juefei-Xu, and C. Feng, "Fooling lidar perception via adversarial trajectory perturbation," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7878–7887, 2021.

[44] J. C. Han and Z. Q. Zhou, "Metamorphic fuzz testing of autonomous vehicles," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ser. ICSEW'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 380–385.

[45] 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 *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 9904–9913.

[46] Q. Zhang, S. Hu, J. Sun, Q. A. Chen, and Z. M. Mao, "On adversarial robustness of trajectory prediction for autonomous vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 15 159–15 168.

[47] Y. Jia, Y. Lu, J. Shen, Q. A. Chen, Z. Zhong, and T. Wei, "Fooling detection alone is not enough: First adversarial attack against multiple object tracking," 2019.

[48] S. Saadatnejad, M. Bahari, P. Khorsandi, M. Saneian, S.-M. Moosavi-Dezfooli, and A. Alahi, "Are socially-aware trajectory prediction models really socially-aware?" *Transportation Research Part C: Emerging Technologies*, vol. 141, p. 103705, 2022.

[49] R. A. Baeza-Yates and Z. Liaghat, "Quality-efficiency trade-offs in machine learning for text processing," *2017 IEEE International Conference on Big Data (Big Data)*, pp. 897–904, 2017.

[50] Z. M. Jiang and A. E. Hassan, "A survey on load testing of large-scale software systems," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1091–1118, 2015.

[51] M. Jangali, Y. Tang, N. Alexandersson, P. Leitner, J. Yang, and W. Shang, "Automated generation and evaluation of jmh microbenchmark suites from unit tests," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1704–1725, 2023.

[52] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: a survey," *IEEE Transactions on Software Engineering*, vol. 30, pp. 295–310, 2004.

[53] S. Becker, H. Koziolek, and R. H. Reussner, "Model-based performance prediction with the palladio component model," in *Workshop on Software and Performance*, 2007.

[54] ——, "The palladio component model for model-driven performance prediction," *J. Syst. Softw.*, vol. 82, pp. 3–22, 2009.

[55] L. Kleinrock, *Queueing Systems, Volume I*, ser. A Wiley-Interscience publication. Wiley, 1974, no. v. 1.

[56] K. Kant and M. Srinivasan, *Introduction to Computer System Performance Evaluation*, ser. McGraw-Hill computer science series. McGraw-Hill, 1992.

[57] F. Bause, ""qn + pn = qpn" - combining queueing networks and petri nets," 1993.

[58] ——, "Queueing petri nets-a formalism for the combined qualitative and quantitative analysis of systems," *Proceedings of 5th International Workshop on Petri Nets and Performance Models*, pp. 14–23, 1993.

[59] F. Baccelli, G. Balbo, R. J. Boucherie, J. Campos, and G. Chiola, "Annotated bibliography on stochastic petri nets," 1994.

[60] F. Bause, P. Buchholz, and P. Kemper, "Integrating software and hardware performance models using hierarchical queeing petri nets," in *Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen*, 1997.

[61] RobotecAI. (2023) Autoware. [Online]. Available: https://github.com/RobotecAI/autoware-1/tree/awsim-ss2-stable

[62] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[63] S. Kounev, "Performance modeling and evaluation of distributed component-based systems using queueing petri nets," *IEEE Transactions on Software Engineering*, vol. 32, no. 7, pp. 486–502, 2006.

[64] S. Kounev, S. Spinner, and P. Meier, "Introduction to queueing petri nets: modeling formalism, tool support and case studies," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 9–18. [Online]. Available: https://doi.org/10.1145/2188286.2188290

[65] Baidu. (2025) Apollo cyber rt. [Online]. Available: https://github.com/ApolloAuto/apollo/tree/master/modules/tools/adataset

[66] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on Robot Learning*, 2017.

[67] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, M. Hutter and R. Siegwart, Eds. Cham: Springer International Publishing, 2018, pp. 621–635.

[68] W. Jansen, N. Huebel, and J. Steckel, "Physical lidar simulation in real-time engine," in *2022 IEEE Sensors*, 2022, pp. 1–4.

[69] Baidu. (2025) Apollo. [Online]. Available: https://github.com/ApolloAuto/apollo/blob/master/cyber/data/channel_buffer.h

[70] T. A. Foundation. (2025) Autoware timer. [Online]. Available: https://github.com/autowarefoundation/autoware.universe/blob/main/sensing/\\autoware_pointcloud_preprocessor/docs/concatenate-data.md

[71] O. D. Team, "Openpcdet: An open-source toolbox for 3d object detection from point clouds," https://github.com/open-mmlab/OpenPCDet, 2020.

[72] J. Romano, J. D. Kromrey, J. Coraggio, J. Skowronek, and L. Devine, *Appropriate Statistics for Ordinal Level Data: Cliff's Delta and the Ordinal Dominance Statistic*. Tallahassee, FL: Florida Association of Institutional Research, 2006, paper presented at the 2006 Annual Meeting of the Florida Association of Institutional Research.

[73] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics*, vol. 1, pp. 196–202, 1945.

[74] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data," in *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 683–700. [Online]. Available: https://doi.org/10.1007/978-3-030-58523-5_40

[75] B. Ivanovic, J. Harrison, and M. Pavone, "Expanding the deployment envelope of behavior prediction via adaptive meta-learning," *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7786–7793, 2022.

[76] T. Gu, G. Chen, J. Li, C. Lin, Y. Rao, J. Zhou, and J. Lu, "Stochastic trajectory prediction via motion indeterminacy diffusion," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17 092–17 101, 2022.

[77] F. U. Haq, D. Shin, S. Nejati, and L. C. Briand, "Can offline testing of deep neural networks replace their online testing?" *Empirical Software Engineering*, vol. 26, 2021.

[78] A. Stocco, B. Pulfer, and P. Tonella, "Model vs system level testing of autonomous driving systems: a replication and extension study," vol. 28, no. 3, May 2023.