Behavior-Driven Falsification of Autonomous Driving Stack Systems with LLM-Generated Scenarios

JEEVITHAN MAHENTHRAN, University of California, Santa Cruz, USA
SERGIO CASARRUBIAS, University of California, Santa Cruz, USA

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Software and its engineering** → *Software verification and validation*; • **Information systems** → Data analytics; Data mining.

Additional Key Words and Phrases: Autonomous Vehicles, Adversarial Scenario Generation, Falsification, Safety Violations, Simulation, Data Analysis, Carla, Scenic

ACM Reference Format:
Jeevithan Mahenthran and Sergio Casarrubias. 2025. Behavior-Driven Falsification of Autonomous Driving Stack Systems with LLM-Generated Scenarios. 1, 1 (March 2025), 20 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

This report presents the final phase of our project on falsification and safety analysis in autonomous vehicle (AV) systems. Due to limited time and the complexity of the toolchain, we began with low-hanging fruits — specifically, the D4 falsification benchmarks — to build familiarity with scenario modeling, the simulation environment, and VerifAI. These early experiments served primarily as an on-ramp to understanding how falsification campaigns are structured, rather than producing meaningful research results. Once this foundation was in place, we shifted to a more targeted and structured investigation using ChatScene, a large language model (LLM)-powered tool for generating adversarial scenarios directly from natural language prompts. The primary objective of this final phase was to evaluate how language-driven, LLM-generated scenarios could provoke safety violations and edge-case failures in AV decision-making systems. Unlike traditional parameterized testing that relies on manual configuration of vehicle dynamics or random variations, ChatScene enables a more expressive and human-aligned method for synthesizing realistic and safety-critical driving environments.

The study was conducted using CARLA 0.9.13, an open-source simulator for adversary vehicle development; Scenic 2.1, a domain-specific probabilistic programming language for scenario modeling; and VerifAI, a falsification and synthesis tool capable of exploring temporal logic constraints. At the core of this phase lies ChatScene, an AI-powered toolchain capable of generating Scenic code directly from natural language prompts. ChatScene enables researchers to prototype safety-critical scenarios with human-interpretable descriptions, reducing reliance on low-level scripting and enhancing the realism of adversarial test cases.

Four adversarial scenarios were generated using ChatScene and analyzed in this report:

Authors' Contact Information: Jeevithan Mahenthran, University of California, Santa Cruz, Santa Cruz, USA, jmahenth@ucsc.edu; Sergio Casarrubias, secasarr@ucsc.edu, University of California, Santa Cruz, Santa Cruz, California, USA.

Manuscript submitted to ACM

- **Dangerous Merge** (`dangerous_merge_attack.scenic`): Tests how AVs with Adaptive Cruise Control (ACC) handle high-risk highway merges initiated by an aggressive ego vehicle.
- **Traffic Light Violation** (`traffic_light_attack.scenic`): Explores whether the ego vehicle can induce the Lead Vehicle into violating traffic signals under ambiguous intersection dynamics.
- **Stop Sign Compliance** (`signal_attack.scenic`): Derived from the California Driver Handbook, this scenario investigates Lead Vehicle behavior at stop-controlled intersections.
- **Following Distance Violation** (`tailgating_attack.scenic`): Examines Lead Vehicle compliance with safe trailing distances in variable traffic conditions.

Each scenario was executed across approximately 100 simulation runs (2–4 hours per scenario), with the driving manual snippet scenarios running directly on ChatScene. Earlier experiments required CARLA 0.9.15 to track distance, speed, crash rates, and event timelines. Key metrics such as inter-vehicle distances, crash rates, and safety constraint violations were extracted, calculated, and visualized using custom analysis tools (`graphifier.py`, `crash_parse.py`).

While earlier research primarily focused on randomized attack vectors like speed modulation and inter-vehicle distance tuning, this final phase demonstrates the efficacy of language-driven, structured adversarial generation in exposing system weaknesses. All experiments were performed on a forked version of the falsification framework available at: https://github.com/Jeeevii/cse233_acc_verifai.

The remainder of this report presents an overview of relevant literature, system design enhancements, experimental setup and methodology, detailed scenario walk-throughs, metrics and results, and an in-depth analysis of the insights gained from ChatScene's integration.

2 Related work

Simulation-based testing has been widely explored as a means to evaluate the safety and reliability of autonomous vehicles. One major challenge in this domain is identifying vulnerabilities in collision avoidance and control systems, particularly under adversarial conditions. Previous research has introduced falsification methodologies as a way to systematically discover unsafe scenarios that might not be uncovered through traditional testing approaches.

A key study that directly aligns with this research is D4: Dynamic Data Driven Discovery of Adversarial Vehicle Maneuvers (Hernandez et al., 2024), which takes a data driven approach to identifying security vulnerabilities in autonomous vehicle systems. This research integrates Dynamic Data Driven Application Systems (DDDAS) with scenario based testing to optimize adversarial strategies dynamically. The study uses MTL to define safety requirements and evaluates how different attack signals persistent and intermittent, can disrupt autonomous control algorithms. A key insight from this work is that adaptive cruise control (ACC) systems may not respond quickly enough to sudden braking or acceleration, creating exploitable weaknesses. This research reinforces the importance of adversarial testing and suggests that small, calculated disruptions to driving behavior can significantly impact vehicle safety.

Another relevant research is Using Falsification to Find Vulnerabilities in Collision Avoidance Systems (Salgado et al., 2022). This paper applies falsification techniques to identify weaknesses in autonomous vehicle collision avoidance strategies. The authors use scenario based simulation combined with Metric Temporal Logic (MTL) to formally define safety requirements and identify cases where the system fails under adversarial conditions. Their approach systematically explores potential attack strategies by optimizing control parameters, such as braking time and intensity, to trigger unsafe situations. This research highlights three different collision avoidance strategies, Default Scenic Control, PD Control, and Constant Time Gap Policy, each with its own level of sensitivity to adversarial maneuvers. The study

demonstrates that by adjusting attack parameters, such as sudden braking, it is possible to induce dangerous behaviors without the adversarial vehicle itself being involved in a crash. This aligns closely with the present research, where the goal is to modify the ego car's behavior in a way that forces trailing vehicles into rule violations, potentially leading to unsafe situations.

A recent advancement that directly aligns with the goals of this project is ChatScene: Knowledge-Enabled Safety-Critical Scenario Generation for Autonomous Vehicles (Zhang et al., 2024), which introduces a novel large language model (LLM) based framework for generating adversarial scenarios. ChatScene uses natural language prompts to generate safety-critical traffic descriptions, which are parsed and mapped to code snippets using a retrieval-based architecture. These snippets are then compiled into executable Scenic scripts for simulation within CARLA. Unlike prior approaches that rely on manual scenario design or the often brittle process of direct code generation from LLMs, ChatScene utilizes a curated database of Scenic components, including road geometry, vehicle behavior, and spawn positions, to ensure robustness and wide scenario coverage. A key contribution of this work is its iterative refinement process, which adaptively samples parameter spaces to increase the likelihood of collision-prone situations. Evaluation on the SafeBench platform shows that scenarios generated with ChatScene increase collision rates by 15% compared to state-of-the-art baselines and improve ego vehicle robustness by reducing collision rates by 9% after fine-tuning. This research supports the growing body of work that treats adversarial testing as a critical step toward safer Lead Vehicle deployment and highlights the potential of LLMs to enhance simulation-based validation pipelines.

3 System Design Overview

The system architecture for this phase of the project was centered around the integration of ChatScene into an existing falsification framework based on CARLA, Scenic, and VerifAI. The goal was to shift from parameter-randomized testing toward language-driven adversarial scenario generation, allowing for more human-interpretable, rule-based, and realistic simulations.

3.1 Architecture Overview

The system consists of the following major components:

- **ChatScene** (LLM + Scenic code generation)
- **CARLA 0.9.13** (ChatScene simulation engine) & **CARLA 0.9.15** (default simulation engine)
- **Scenic 2.1** (scenario description language)
- **VerifAI** (temporal logic falsification and parameter synthesis)
- **Custom Analysis Scripts** (`graphifier.py`, `crash_parse.py`)

These components form a modular pipeline where natural language prompts are converted into Scenic scripts via ChatScene's LLM-powered generation engine. These scripts are executed in CARLA simulations, where VerifAI systematically explores the parameter space to identify violations of safety properties.

3.2 Workflow Breakdown

Natural Language Prompting. Each adversarial test begins with a descriptive prompt, such as:

> *"An ego vehicle dangerously merges into a lane on a busy highway, causing trailing autonomous vehicles to brake abruptly."*

Scenario Generation via ChatScene. ChatScene parses the input and retrieves modular Scenic components from a curated library. These include components such as:

- Road Layouts
- Vehicle Placements
- Behavior Policies
- Dynamic Objects (e.g., traffic lights, pedestrians)

The final Scenic script is automatically validated for syntax and semantic correctness before simulation.

Simulation Execution with CARLA + VerifAI. Each generated script is executed through the CARLA simulator. VerifAI varies environmental or behavioral parameters (e.g., ego speed, start position) to expose boundary-case behaviors.

Violation and Data Logging. During each simulation run, the system logs:

- Position and velocity of all agents
- Inter-vehicle distances
- Collision events
- Temporal logic violation flags

Post-Processing & Analysis. A suite of Python scripts is used to visualize and interpret results:

- `graphifier.py`: Generates time-series plots from CSVs
- `crash_parse.py`: Filters logs to isolate successful falsifications

Design Diagram

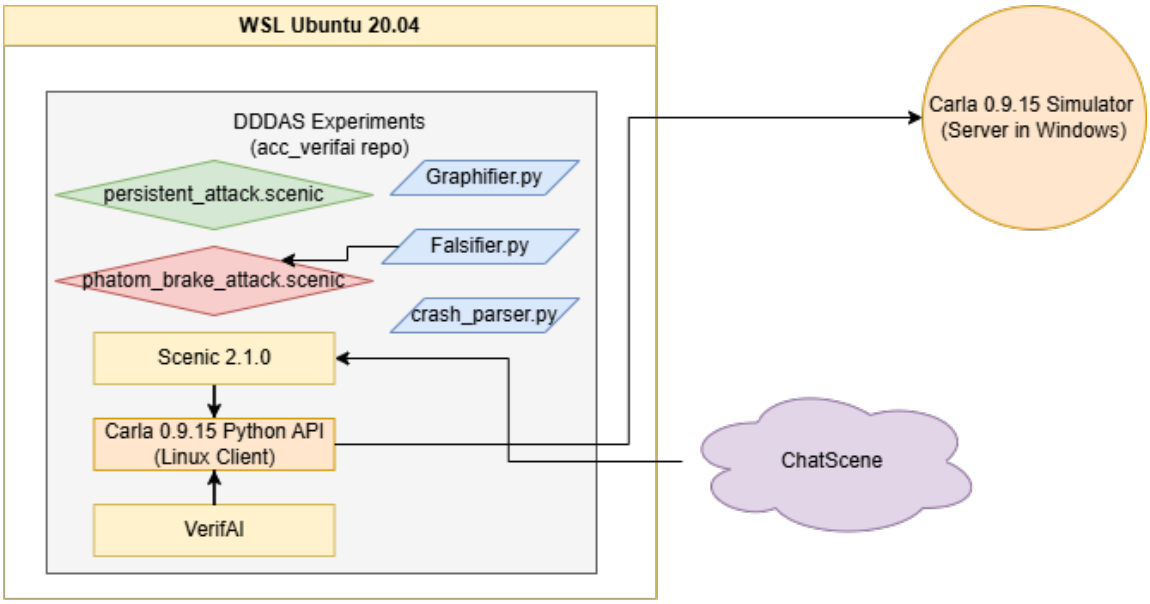


Fig. 1. System Design Diagram

4 Case Study Setup: ChatScene-Driven Scenario Generation

Each of the following four case study scenarios was generated using the ChatScene framework and tested using a consistent simulation and analysis pipeline. The goal was to evaluate how well LLM-generated scenarios could provoke violations in an autonomous vehicle (AV) system using naturalistic, regulation-driven behaviors.

For each case study, the following steps were performed:

(1) A natural language prompt was written to describe an adversarial or safety-critical situation.
(2) The prompt was input into ChatScene, which used its retrieval-based generation engine to produce valid Scenic scripts by combining pre-defined road geometries, vehicle behaviors, and positioning templates.
(3) The generated script was automatically validated to ensure it could be compiled and simulated without errors.
(4) Once validated, the scripts were executed in CARLA 0.9.13 through ChatScene.
(5) Approximately 100 simulations were run for each scenario, generating logs, crash data, and time-series metrics for analysis.

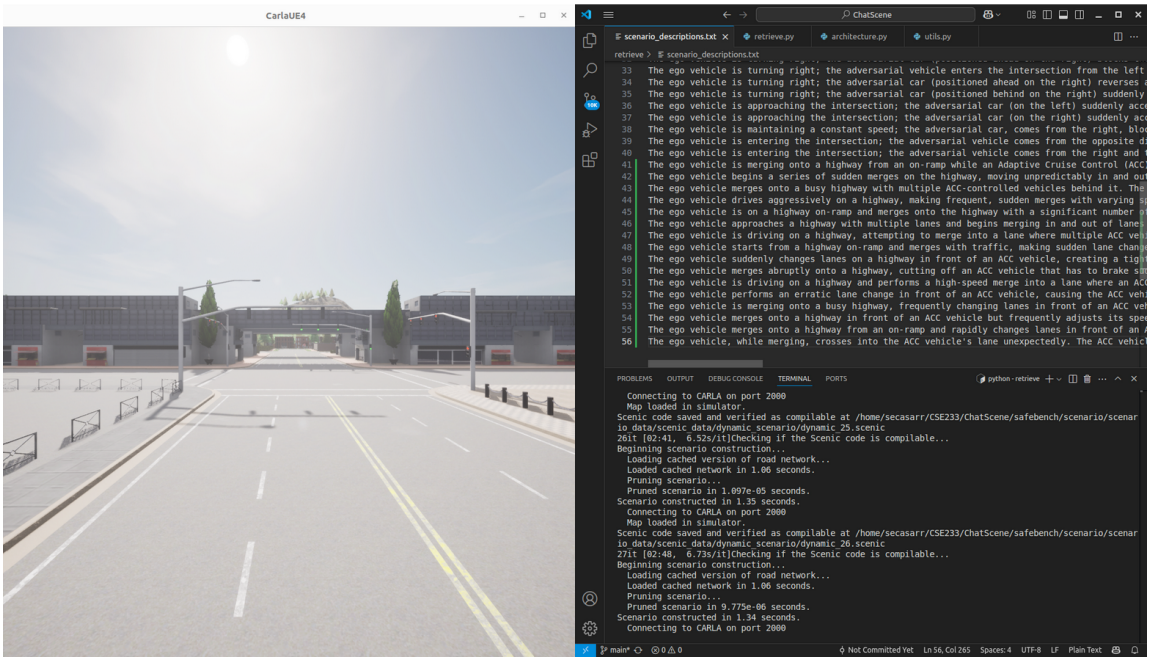


Fig. 2. ChatScene Set Up with Sample Prompts

Figure 2 illustrates ChatScene's prompt processing logic. Each line in the input file represents a complete natural language prompt. Before running the prompt generation, a custom version of CARLA must be launched in server mode. Once the server is active, executing the `retrieve.py` script reads the prompts line by line, generating corresponding Scenic code for each scenario.

The tool then attempts to verify whether the generated code is compatible and executable. If the scenario is valid, it is saved as an executable Scenic file. If it is not, the code is still saved, but stored in a plain text file to indicate that it requires manual adjustments before it can be run successfully.

4.1 Case Study Scenario 1 – Dangerous Merge

Prompt Used.

> *"The ego vehicle begins a series of sudden merges on the highway, moving unpredictably in and out of lanes, forcing an ACC vehicle to react by adjusting its speed and maintaining a safe distance. The goal is to evaluate the ACC system's ability to handle sudden lane changes and maintain safe following distances on a busy highway."*

Generated Behavior: `dangerous_merge.scenic`. ChatScene parsed the above prompt into a Scenic script simulating an aggressive highway merge. The ego vehicle initiates a sharp merge maneuver into the path of an ACC-enabled vehicle already in the right lane. The scenario includes multi-agent behavior and variable speed adjustments from the ego vehicle.

```
# Adversarial vehicle follows lane, then merges in front of ego
behavior AdvBehavior():
    while (distance to self) > 60:
        wait
    do FollowLaneBehavior(target_speed=globalParameters.OPT_ADV_SPEED) until (distance to self) < globalParameters.OPT_ADV_DISTANCE
    do LaneChangeBehavior(laneSectionToSwitch=network.laneSectionAt(ego), target_speed=globalParameters.OPT_ADV_SPEED)

# Follower behavior (simple ACC approximation)
behavior FollowerBehavior(target_speed):
    do FollowLaneBehavior(target_speed=target_speed)

```

Fig. 3. ChatScene Generated Behavior Snippet of `dangerous_merge.scenic`

Figure 3 illustrates the Scenic code generated from the dangerous merge prompt. This setup creates dynamic interactions where the Lead Vehicle must decide whether to brake, adjust speed, or change lanes in response to the ego vehicle's sudden merge.

The primary safety property tested was:

> *"The Lead Vehicle shall maintain a safe following distance and avoid collision during lane merges."*

Case Study Scenario 1 Results. The "Dangerous Merge" case study demonstrates ChatScene's strength in modeling nuanced traffic scenarios with multi-agent consequences. The merging behavior triggered latent vulnerabilities in the Lead Car's Adaptive Cruise Control (ACC) logic. Specifically:

Violations Detected: 14 / 100

ChatScene's structured behaviors, `AdvBehavior()`, enabled realistic ego maneuvers difficult to model using parameter randomization alone.

This scenario validated that LLM-generated prompts can simulate traffic weaving, risky merges, and evasive braking, capturing adversarial situations with a high degree of realism and impact.

Figure 4. CARLA 0.9.15 In-Simulation View of the Dangerous Merge Scenario. A screenshot during execution shows the ego vehicle initiating a sharp merge into the lead vehicle's lane.

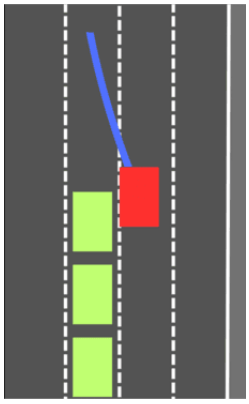


Figure 5. Top-Down View of the Dangerous Merge Scenario from ChatScene. Visual representation of lane geometry and vehicle placement before and during the merging event.

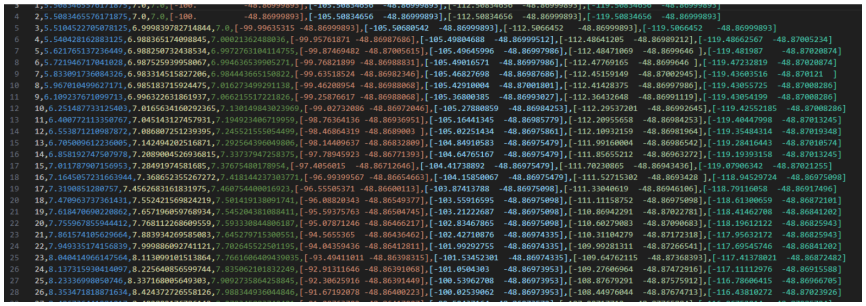


Figure 6. Falsifier Output CSV Snapshot for Dangerous Merge Scenario. Sample data showing vehicle states, parameter variations, and timestamps of safety violations.

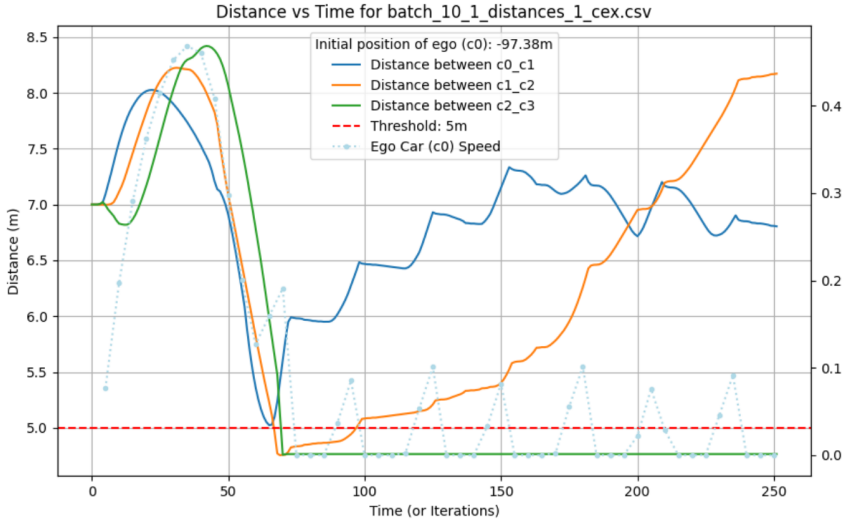


Figure 7. Graphifier Time-Series Plot for Dangerous Merge Scenario. Visualization of inter-vehicle distances over time, highlighting moments of unsafe following during merge attempts.

4.2 Case Study Scenario 2 – Traffic Light Violation

Prompt Used.

> *"The ego vehicle approaches a signalized intersection and exhibits abnormal behavior such as abruptly stopping, reversing or accelerating late through a yellow light. This challenges the ACC enabled autonomous vehicle following behind. The goal is to test whether the Lead Vehicle can obey traffic signal rules while responding safely to the unpredictable behavior of an AV. The Lead Vehicle should not enter an intersection during a red light and must maintain a safe following distance. The scenario is inspired by California driving rules which require stopping at yellow lights when it is safe and never proceeding on red."*

Generated Behavior: `traffic_light_attack.scenic`. ChatScene translated this prompt into a Scenic script featuring an ego vehicle navigating a signal-controlled intersection. The ego vehicle is programmed to decelerate abruptly, reverse

slightly, or hesitate during a yellow signal phase. This creates a conflict for the trailing AV, which is forced to either proceed through the intersection unsafely or stop abruptly at a late stage.

```
# Ego stops or reverses unpredictably at the intersection
behavior TrafficLightAttacker(id, dt, ego_speed, lane):
    long_control = AccControl(id, dt, ego_speed, True)
    lat_control = LateralControl(dt)
    acted = False
    while True:
        cars = [ego] + trailing
        b, t = long_control.compute_control(cars)
        s = lat_control.compute_control(self, lane)
        if not acted and self.position.distanceTo(intersection.position) < globalParameters.STOP_BEHAVIOR_DISTANCE:
            if globalParameters.REVERSAL_ENABLED:
                take SetThrottleAction(-0.5), SetBrakeAction(0.0)
            else:
                take SetThrottleAction(0.0), SetBrakeAction(1.0)
            acted = True
        else:
            take SetThrottleAction(t), SetBrakeAction(b), SetSteerAction(s)
```

Figure 8. ChatScene Generated Behavior Snippet of `traffic_light_attack.scenic`. The code encodes behaviors such as hesitation, reverse motion, and late crossing during yellow lights, targeting safe-following and traffic law compliance.

This scenario specifically targets red light compliance and reaction timing under ambiguous traffic signal behavior. The Lead Vehicle behind the ego vehicle must decide whether to proceed or stop when the ego's behavior disrupts expected motion patterns.

Primary Safety Property Tested:

> *"The Lead Vehicle shall not enter an intersection when the light is red, and must maintain a safe stopping distance behind a leading vehicle."*

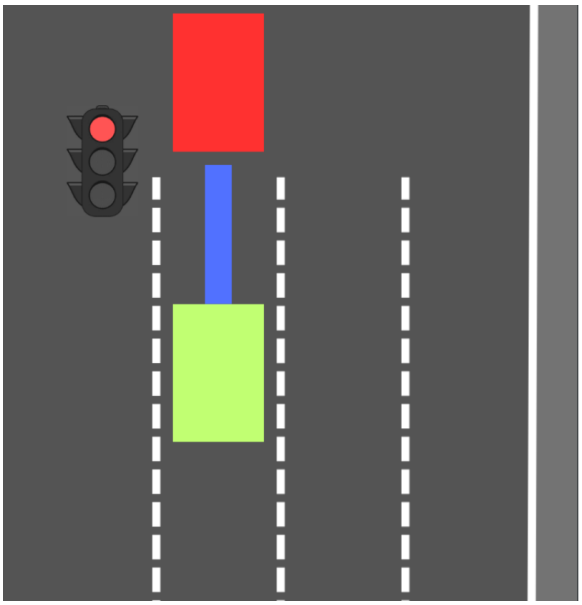


Figure 9. Top-Down View of the Traffic Light Attack Case Study from ChatScene. The scene illustrates the ego vehicle's position near a signalized intersection, along with lane geometry and the trailing autonomous vehicle.

Case Study Scenario 2 Results. The "Traffic Light Violation" case study demonstrates how ChatScene-generated behaviors like hesitation, reverse motion, and delayed intersection entry can reveal weaknesses in an AV's intersection-handling logic.

Violations Detected: 8 / 100

ChatScene's `TrafficLightAttacker()` behavior consistently introduced ambiguous timing — something that's difficult to replicate using randomized parameters.

Notably, in some runs, the Lead Vehicle chose to stop prematurely — demonstrating overly conservative logic. This suggests ChatScene prompts may help simulate not just unsafe violations, but also corner cases of overly cautious Lead Vehicle decision-making.

4.3 Case Study Scenario 3 – California Stop Sign Rule Violation

4.3a. Prompt Used.

SECTION 7. *Laws and Rules of the Road*

TRAFFIC CONTROL

When at or approaching traffic signals or signs, yield to pedestrians, bicyclists, and other nearby vehicles that may have the right-of-way. See Right of Way Rules: Who Goes First, in this section.

Traffic Signals

Solid Red Light

A red traffic signal light means STOP. You can turn right at a red light, if:

- There is not a NO TURN ON RED sign posted.
- You stop at the stop or limit line, yield for pedestrians, and turn when it is safe.

Red Arrow

A red arrow means STOP. Do not turn at a red arrow. Remain stopped until a green traffic signal light or green arrow appears.

Flashing Red Light

A flashing red signal light means STOP. After stopping, you may go when it is safe.

Solid Yellow Light

A yellow traffic signal light means CAUTION. The light is about to turn red. When you see a yellow traffic signal light, stop, if you can do so safely. If you cannot stop safely, cautiously cross the intersection.

Figure 10. Driving Manual of California Laws and Rules of the Road Snippet Fed into ChatScene. The DMV reference was used to ground the generated behavior in real-world traffic rules.

4.3b. Generated Behavior: `california_manual_stop_rule.scenic`. ChatScene generated a Scenic script based on the prompt that simulates a 2-way stop intersection with an ego vehicle that does not perform a complete stop or slightly overshoots the stop line. The trailing autonomous vehicle must interpret this behavior and respond according to stop

sign rules in California's DMV handbook — specifically the requirement to stop at the line or before entering the intersection.

```
behavior SignalFollower(id, dt, speed, lane):
    long_control = AccControl(id, dt, speed, False)
    lat_control = LateralControl(dt)
    while True:
        b, t = long_control.compute_control([ego])
        s = lat_control.compute_control(self, lane)
        if self.position.distanceTo(intersection.position) < globalParameters.SIGNAL_DISTANCE:
            take SetThrottleAction(0.0), SetBrakeAction(1.0), SetSteerAction(s)
        else:
            take SetThrottleAction(t), SetBrakeAction(b), SetSteerAction(s)
```

Figure 11. ChatScene Generated Behavior Snippet of `california_manual_stop_rule.scenic`. The Scenic code models incomplete stops and slight rollovers through a stop sign at a 2-way intersection.

This case aims to evaluate whether the Lead Vehicle respects the stop rule when the lead car exhibits borderline behavior, potentially triggering a domino effect where multiple vehicles fail to stop properly due to ambiguous lead behavior.

Primary Safety Property Tested:

> *"The Lead Vehicle shall come to a complete stop at a stop-controlled intersection and maintain a safe following distance."*

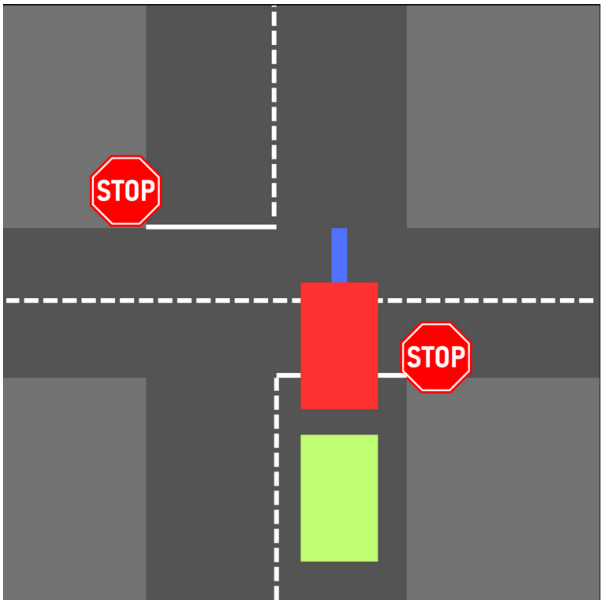


Figure 12. Top-Down View of the California Manual Stop Rule Case Study. Visualization of the stop intersection layout, ego vehicle behavior, and trailing autonomous vehicle positioning.

4.3c. Case Study Scenario 3 Results. This scenario exposed rule-based corner cases in Lead Vehicle behavior — particularly around interpreting lead vehicle decisions at low speeds. The ego's behavior often caused the trailing Lead Vehicle to either:

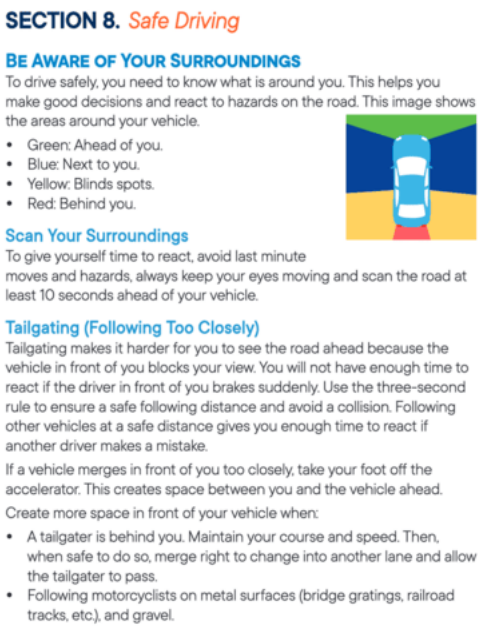
- Stop too late, entering the intersection improperly
- Fail to stop fully, mimicking the rolling behavior

Violations Detected: 9 / 100

ChatScene's `SignalFollower()` behavior introduced subtle infractions that traditional falsifiers wouldn't easily catch. The Lead Vehicle system occasionally mirrored the ego vehicle's behavior, showcasing a form of policy imitation that could cascade into compounding infractions — a particularly relevant issue for multi-agent traffic environments.

This scenario effectively demonstrates how regulatory compliance violations (as opposed to crash-based violations) can be simulated using natural-language-generated behaviors, giving broader coverage to corner cases found in real-world DMV rulebooks.

4.4 Case Study 4 – California Following Distance Rule Violation



SECTION 8. *Safe Driving*

BE AWARE OF YOUR SURROUNDINGS

To drive safely, you need to know what is around you. This helps you make good decisions and react to hazards on the road. This image shows the areas around your vehicle.

- Green: Ahead of you.
- Blue: Next to you.
- Yellow: Blinds spots.
- Red: Behind you.

Scan Your Surroundings

To give yourself time to react, avoid last minute moves and hazards, always keep your eyes moving and scan the road at least 10 seconds ahead of your vehicle.

Tailgating (Following Too Closely)

Tailgating makes it harder for you to see the road ahead because the vehicle in front of you blocks your view. You will not have enough time to react if the driver in front of you brakes suddenly. Use the three-second rule to ensure a safe following distance and avoid a collision. Following other vehicles at a safe distance gives you enough time to react if another driver makes a mistake.

If a vehicle merges in front of you too closely, take your foot off the accelerator. This creates space between you and the vehicle ahead.

Create more space in front of your vehicle when:

- A tailgater is behind you. Maintain your course and speed. Then, when safe to do so, merge right to change into another lane and allow the tailgater to pass.
- Following motorcyclists on metal surfaces (bridge gratings, railroad tracks, etc.), and gravel.

Fig. 4. Driving Manual of California Safe Driving Snippet Fed into ChatScene

Prompt Used.

Generated Behavior: `california_manual_following_distance.scenic`. ChatScene generated a Scenic script based on this prompt, simulating a two-vehicle highway scenario where the ego vehicle performs unpredictable short-term deceleration and acceleration, pressuring the trailing vehicle's following policy.

The `Follower()` behavior was composed from ChatScene's behavior library, producing minor but rapid braking pulses interleaved with forward acceleration—mimicking aggressive or inattentive human driving. This tested the AV's constant time-gap following logic and how it handled dynamic spacing.

```
# ACC behavior
behavior Follower(id, dt, speed, lane):
    long_control = AccControl(id, dt, speed, False)
    lat_control = LateralControl(dt)
    while True:
        b, t = long_control.compute_control([ego])
        s = lat_control.compute_control(self, lane)
        take SetThrottleAction(t), SetBrakeAction(b), SetSteerAction(s)
```

Fig. 5. ChatScene Generated Behavior Snippet of `california_manual_following_distance.scenic`

Primary Safety Property Tested:

"The Lead Vehicle shall maintain a safe following distance, adjusting its speed based on the behavior of the lead vehicle to avoid tailgating or collisions."

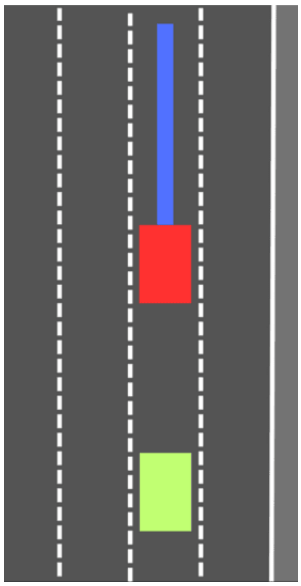


Fig. 6. ChatScene Top Down View of the California Manual Following Distance Case Study

Case Study 4 Results. This scenario uncovered multiple cases where adaptive following policies failed to react quickly enough to subtle but rapid braking. ChatScene's generated behavior provided non-catastrophic but realistic adversarial driving, making it an ideal candidate for policy tuning and stress-testing.

Violations Detected: 10 / 100

The Lead Vehicle occasionally attempted to recover from a late response by swerving or braking sharply—both undesirable in real-world driving. Notably, some falsification cases were non-collision but comfort violations, such as aggressive deceleration that would be unsafe for passengers or nearby vehicles.

This scenario highlights the strength of LLM-driven behavior modeling—generating adversarial patterns that do not rely on extreme inputs but instead reflect the subtle stressors of real-world traffic.

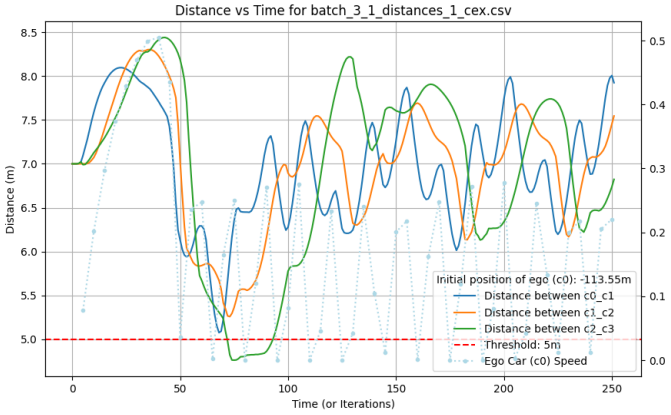


Fig. 7. Graph generated by Graphifier for results from Scenario 1, Batch 3 of 200

5 Case Study Summary

Scenario Type	Total Runs	Crashes	Chat Scene Generation Success Rate (%)	Notable Observations
Persistent Attack	1500	120	N/A	Ego mostly causes crashes when the speed is greater than 20 m/s
Dangerous Merge	100	14	48%	Driving Stack with ACC controller was easily influenced when Ego made merges at rapid speeds
Traffic Light Violation	100	8	52%	AV decision latency was slow
Stop Sign Rule Violation	100	9	39%	Lead Vehicle mirrored ego's rolling stop mostly
Following Distance Violation	100	10	41%	Crash risk due to late Lead Vehicle reaction

Fig. 8. Case Study Summary Table Diagram

For each scenario shown in Figure 8, 20–30 natural language prompts were created with varied parameters, such as vehicle types, entry points, and behaviors. These prompts were fed into ChatScene, which generated corresponding Scenic scripts. Only those that passed internal compilation and scenario checks—ensuring no empty scenes, invalid behavior trees, or simulation load errors—were retained for use. The validated scripts were then executed within the VerifAI falsification framework, producing datasets for analysis.

This filtering process ensured that each selected case study represented a realistic and high-quality adversarial test, balancing both behavioral complexity and execution stability.

Across all case studies, the most effective attacks were those that created dynamic, human-like driving behaviors, leading to hesitation, confusion, or overreaction in the trailing autonomous vehicle (AV). These adversarial patterns caused the AV to misinterpret traffic context or respond too slowly under stress.

- **Dangerous Merge** scenarios triggered the highest falsification rate (14%), leveraging abrupt lane shifts and speed modulation to exploit gaps in the AV's Adaptive Cruise Control (ACC) logic. The ego vehicle's aggressive maneuvers often forced late braking or unsafe following decisions by the AV.
- **Following Distance Violation** (10%) also proved effective, with pulse braking behaviors that subtly tested the AV's time-gap response without requiring overt collisions.
- By contrast, **Traffic Light Violation** (8%) and **Stop Sign Violation** (9%) exposed regulatory policy failures rather than physical collisions. These scenarios illuminated weaknesses in rule enforcement, such as roll-throughs at stop signs or entering intersections during late yellow phases.

- Together, these results suggest that merge-based and distance-modulation scenarios are more likely to provoke unsafe reactions under tight timing or spatial constraints. Meanwhile, ChatScene-generated regulatory violations show promise for revealing subtle policy misalignments or inconsistency in AV interpretation of ambiguous situations.

6 Challenges

The integration of ChatScene into our testing pipeline posed some of the most technically complex and persistent challenges throughout the project. The biggest initial obstacle stemmed from running CARLA through the Windows Subsystem for Linux (WSL), which triggered a `CommonUnixCrashHandler: Signal=11` error. This segmentation fault consistently caused CARLA to crash when attempting to run ChatScene-generated scenarios. Drawing from prior success resolving socket communication issues between CARLA and Scenic, we attempted similar debugging strategies, including testing different CARLA versions and dependency configurations. However, we were ultimately unable to resolve the issue within the constraints of WSL.

To work around this limitation, we explored multiple experimental setups. These included running ChatScene's custom CARLA 0.9.13 build alongside SafeBench under WSL, attempting a native Windows installation of CARLA with cross-platform communication to the ChatScene client running in WSL Ubuntu 22.04, and importing key files from the ChatScene repository into our Scenic-focused testbed. Despite considerable effort, none of these configurations succeeded due to persistent instability and inter-process communication failures between Windows and WSL.

Eventually, we opted for a more drastic but effective solution by reinstalling the host operating system and setting up a dual-boot configuration with native Ubuntu. This allowed for a stable Linux environment in which CARLA and ChatScene could run without WSL-related crashes. However, this introduced a new set of challenges related to hardware compatibility. Our primary development machines were equipped with AMD GPUs, which are not natively supported by the CUDA toolkit required by several components of ChatScene, particularly those involving deep learning inference using PyTorch. ChatScene's `utils.py`, extttarchitecture.py, and `retrieve.py` modules were all configured to use CUDA by default. After researching AMD compatibility, we attempted to use ROCm (Radeon Open Compute), which theoretically supports the `cuda:0` device string in PyTorch. Despite multiple installation attempts and environment tweaks, ROCm proved incompatible with our hardware and consistently produced low-level execution errors.

In response, we manually modified the codebase to enforce CPU-based execution. This involved removing or bypassing CUDA-specific function calls and ensuring that all modules explicitly used the CPU device. We also adjusted the `setGPU()` utility to prevent GPU allocation. Although this allowed us to run the code in a limited capacity, performance was significantly impacted due to the absence of hardware acceleration. More importantly, this limited our ability to scale or parallelize scenario generation and falsification testing.

In the later stages of the project, we encountered a `ModuleNotFoundError: No module named 'torch._custom_ops'` error when attempting to generate new scenarios and run existing ones. This issue prevented us from extracting screenshots and simulation visuals for additional case studies. Despite reinstalling PyTorch, using the `requirements.txt` file, and cloning a fresh copy of the ChatScene repository, the error persisted across systems. This effectively halted our efforts to expand the number of experimental results. Although we had originally planned to present more generated scenarios, we focused on the four completed case studies and worked with the data we had.

These cumulative challenges highlight both the potential and the practical limitations of integrating LLM-based tools like ChatScene into an autonomous driving simulation pipeline. ChatScene is a powerful and innovative framework, but it requires a carefully controlled and well-supported development environment. For future iterations of this project,

improving our infrastructure by using systems with NVIDIA GPUs or containerized environments with GPU support will be critical to unlocking the full capabilities of the tool.

7 Contributions and Conclusion

This project was a collaborative effort between Jeevithan Mahenthran and Sergio Casarrubias, with responsibilities divided to ensure balanced involvement across all phases of the work.

Individual Contributions

Jeevithan Mahenthran:

- I developed and debugged the ChatScene generated script to work with 0.9.15 in our forked version of the `acc_verfiAI` repo
- I conducted all initial experiments on D4-style scenarios and integrated distance and crash monitoring into the simulation backend.
- I created custom analysis scripts including `graphifier.py` and `crash_parse.py` to evaluate simulation outcomes.
- I managed the adaptation of earlier attack scenarios to the ChatScene framework and tested compatibility with CARLA 0.9.15.

Sergio Casarrubias:

- I handled the integration and setup of ChatScene, including deep learning environment configuration and codebase modifications for CPU compatibility.
- I designed and authored the natural language prompts for all case studies based on real-world traffic rules and driving manuals.
- I curated and validated ChatScene-generated Scenic scripts, ensuring scenario realism and filtering out invalid cases.
- I conducted final scenario runs and data logging for all four adversarial test cases.

Conclusion

Despite various technical and logistical barriers, this project successfully demonstrated the potential of using large language models to generate meaningful adversarial test scenarios for AV falsification. The use of ChatScene enabled high-level, interpretable scenario generation that moved beyond manual parameter tweaking, revealing subtle weaknesses in AV control policies. While the number of scenarios tested was limited, the results showed that LLM-generated behaviors can meaningfully stress-test adaptive cruise control and traffic rule compliance logic in AV systems.

8 Future Work and Timeline Limitations

Several factors limited the overall scope and experimental depth of this research:

- **Computational Resources:** The lack of access to machines with NVIDIA GPUs significantly hindered our ability to use CUDA-dependent components in ChatScene. As a workaround, we enforced CPU-based execution, which slowed down simulation and scenario generation.
- **OS Compatibility:** Many core components of ChatScene and CARLA were unstable under WSL. Ultimately, we had to configure a dual-boot Linux system to ensure a reliable runtime environment.

- **Limited Timeline:** The project timeline constrained the number of case studies we could execute and analyze. Each high-fidelity scenario took hours to simulate and log across 100+ runs.
- **Knowledge Ramp-Up:** At the beginning of the project, significant time was spent learning the intricacies of Scenic, VerifAI, and ChatScene. This limited the time available for advanced falsification techniques or scenario diversity.

For future iterations, we recommend setting up containerized environments with GPU support early in the project timeline, expanding the number of natural language prompts and scenario variations, and developing a falsification-specific monitor to automatically label safety violations. These improvements would scale scenario generation, reduce debugging time, and increase the experimental reach of LLM-driven AV testing.

References

[1] Salgado, I. F., Quijano, N., Fremont, D. J., and Cardenas, A. A. (2022). Fuzzing malicious driving behavior to find vulnerabilities in collision avoidance systems. In *Proceedings of the 2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 368–375. IEEE.

[2] Hernandez, C., Ortiz Barbosa, D., Lei, Z., Burbano, L., Park, Y., Ukkusuri, S., and Cardenas, A. A. D4: Dynamic Data-Driven Discovery of Adversarial Vehicle Maneuvers.

[3] Zhang, J., Xu, C., and Li, B. (2024). ChatScene: Knowledge-Enabled Safety-Critical Scenario Generation for Autonomous Vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[4] Scenic Development Team. Scenic: A Probabilistic Programming Language for Scenario Generation. https://scenic-lang.org.

[5] VerifAI Development Team. VerifAI: Verification and Testing of AI-Driven Systems. https://verifai.readthedocs.io/en/latest/.

[6] CARLA Team. CARLA: Open-Source Autonomous Driving Simulator. https://carla.org.

[7] DDDAS Repository. Dynamic Data-Driven Discovery of Adversarial Vehicle Maneuvers Repository. https://github.com/lburbano/acc_verifai.

[8] ChatScene Repository. ChatScene: AI-Driven Scenario Generation for Autonomous Vehicles. https://github.com/javyduck/ChatScene.

[9] ACM Publications. ACM Journals Primary Article Template. https://www.overleaf.com/latex/templates/acm-journals-primary-article-template/cpkjqttwbshg.