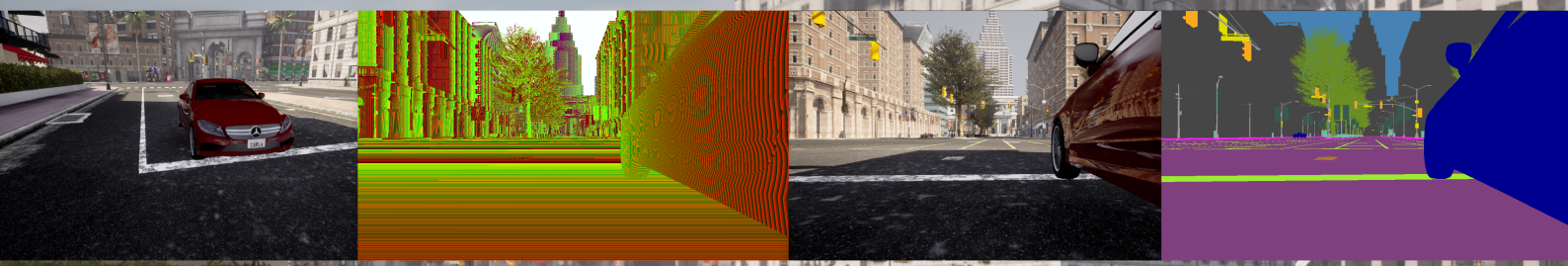
UNIVERSITY
OF OSLO



Master's thesis

Decreasing driveability in autonomous driving system simulator scenarios

LLM4DD: A tool for uncovering faults in autonomous driving systems using large language models

Oliver Ruste Jahren

Informatics: Programming and Systems architecture
60 ECTS study points

Department of Informatics
Faculty of Mathematics and Natural Sciences

Autumn 2025

Draft of November 15, 2025 (commit: 4676474)

Oliver Ruste Jahren

Decreasing driveability in autonomous driving system simulator scenarios

LLM4DD: A tool for uncovering faults in autonomous driving systems using large language models

Supervisors:
Shaukat Ali
Karoline Nylænder

Simula Research Laboratory

Abstract

Autonomous driving systems (ADSs) should be capable of tackling any challenge they may be faced with while driving. Before deploying such ADSs, it is important to be confident that they *are* capable of handling potential challenges in a fitting manner. But there is a near infinite set of potential scenarios that an ADS may be faced with, making it impossible to predict and test on all possible scenarios in advance. To this end, we propose using LLMs to decrease the driveability of our existing ADS scenarios, enabling the ADS to face a more challenging test environment. By using these more challenging test scenarios, we can either (1) cause it to fail and analyse why the ADS failed, or (2) increase our confidence in it being able to operate in challenging scenarios. We implement a tool — LLM4DD — for doing this and evaluate it with regard to the jerk metric. We compare the jerk of the ADS in the 'base' and 'enhanced' versions of the scenario, assessing if the driveability was decreased and how the ADS responded to this more challenging operating environment. We also perform a literature review to survey the extant related works and evaluate how LLM4DD fits into the state of the art. Experimental results show that using LLMs to decrease driveability is a promising strategy if the range of the changes the LLM is allowed to make is limited, whereas problems related to hallucination and simulator crashes arise if the LLM makes excessive changes to the original scenario. Based on the results, we present the research and outline several strategies for improving LLM4DD in future research.

Sammendrag

Selvkjørende biler burde være i stand til å hanskes med enhver utfordring som måtte komme deres vei mens de er ute og kjører. Før man slipper selvkjørende biler fri ut i verden er det derfor viktig å være viss på at de *er* i stand til å håndtere slike potensielle utfordringer. Men det er nært sagt et uendelig antall mulige scenarier en bil kan komme til å stå ovenfor, slik at det er umulig å forutse alle scenarier og teste på disse i forkant. Derfor foreslår vi å anvende store språkmodeller for å gjøre dagens eksisterende scenarier mindre kjørbare enn hva de allerede er, slik at bilen får bryne seg på større utfordringer i forkant av å møte på dem ute i verden. Ved å benytte disse mer komplekse testscenariene vil vi enten kunne (1) trigge den til å feile slik at vi kan analysere hva som gikk galt og lære noe nytt om bilen, eller (2) være mer trygge på at bilen er i stand til å hanskes med komplekse scenarier. På bakgrunn av dette implementerer vi et verktøy — LLM4DD — for å gjøre nettopp dette og evaluerer det med hensyn på rykk-metrikken. Vi sammenligner rykk i den selvkjørende bilen på tvers av baseformen av scenariet og dets forbedrede versjoner, og ser på om kjørbarheten har blitt forverret fra hvordan den var opprinnelig og hvordan bilen forholder seg til dette mer utfordrende scenariet. Vi har også gjennomført en litteraturgjennomgang og sett på et bredt utvalg forskning fra feltet slik at vi kan ta stilling til hvor LLM4DD passer inn i terrenget. Eksperimentene våre indikerer at dette konseptet med å bruke store språkmodeller til å senke kjørbarheten er en lovende fremgangsmåte all den tid språkmodellen begrenses i hvor *brede* endringer den får lov til å gjøre. Dersom den slippes fri uten tøyler, støter vi på problemer knyttet til hallusinering og at simulatoren krasjer. Basert på resultatene presenterer vi selve forskningen, samt en rekke videre strategier for å forbedre LLM4DD i fremtidig arbeid.

Preface

Many thanks to my lovely supervisors Shaukat Ali and Karoline Nylænder, always encouraging me to shoot for the stars and suggesting insightful ways to further the work with the research. Thanks to Simula Research Laboratory and the Department of Informatics for enabling me to do this fascinating work. Thanks to all my friends both within and outside of `ifi` for motivating me to finish the thesis.

The LaTeX sources for the project and their associated commit history is publically available on the GitHub repo https://github.com/orjahren/master, along with an overview of various issues and their progression. Hopefully all the issues are resolved by the time you read this. The code used for the thesis experiments is available on the GitHub repo https://github.com/orjahren/LLM4DD. A similar issue tracking régime has been used for the code and experiments. The project has been undertaken on Fedora Linux 42 and the thesis has undergone an accessibility check under continuous integration to assert its continued accessibility in accordance with best practice. Whenever I tried to do something I shouldn't have, Github would send out a very angry email. The thesis should therefore adhere to standard `PDF/A-2u`.

> *"If I have seen further it is by standing on the shoulders of Giants."*
>
> – **Isaac Newton**

Contents

List of Figures

Listings

Chapter 1

Introduction

> A problem well stated is a problem half solved.
>
> Charles F. Kettering

This chapter presents the motivation and problem statement of the thesis, condensing the the problem statement to a set of formalized research questions. Finally, we present the structure of the thesis with an outline of the topics of each chapter.

1.1 Motivation

Conventional cars are ubiquitous in society [45, p. 1]. Whether for freight trafficking or for humans, cars have great flexibility with their ability to go wherever without requiring tailored infrastructure such as railway tracks. They do, however, have one major weak point — the human driver [40, p. 67]. For this reason, industry and academia have put forward efforts to enhancing cars with Autonomous driving system (ADS) capabilities. By empowering humans with autonomous vehicles, it is expected that traffic safety and efficiency will increase along with comfort as well as enabling the development of several other new transportation methods [29, 37, pp. 1–2, 1].

Due to the critical safety situation of operating a car, it is essential that ADS are thoroughly tested before they are deployed so that they are verified to be sufficiently safe and capable of handling the situations in which they may typically end up [57, p. 1]. But due to the complicated nature of the typical ADS operating environment, coming up with exhaustive system test solutions is near impossible [33, p. 52]. For this reason, we want a way of testing the system that is capable of pushing the ADS to its limits such that

we can measure its performance and see if it is capable of handling critical scenarios [54].

Existing methods for testing ADSs typically rely on driving billions of miles, but this incurs high cost, and is time-intensive [57, p. 1]. To address some of these concerns, ADS simulators have been utilised. But it's not the case that we will *know* that the ADS is safe after it has driven x kilometres on roads or y kilometres in a simulator – we will never be able to predict all possible operating situations in advance [25, p. 1]. Therefore, we need to challenge the ADS as much as possible when testing it. Having it take on a set of low-driveability test scenarios in advance of real-world operations, we can be more confident in the correctness of our ADS if it is able to complete the scenario. By making the scenario *more complex* – less driveable – for the ADS, our confidence in it will increase if it is able to complete the scenario. And if it fails at executing the more complex scenario, we will potentially have uncovered an underlying issue in the ADS that we did not know about so that we can fix it before it causes harm in the real world.

Having an existing repository of ADS simulator scenarios, we wish to improve them in such a way that they are less driveable and more challenging for the ADS. Large Language Models (LLMs) have demonstrated great capabilities of context learning and emergent abilities [10, p. 1], which begs the question of their applicability for ADS testing. We therefore ask: Can these existing test scenarios be made less driveable by applying LLM technology to them?

1.2 Problem description

Traditional techniques for obtaining ADS scenarios rely on (1) highly skilled manual labour, or (2) automated generation [54, p. 1]. The prior incurs a significant cost, and is a major limitation in obtaining a large number of good scenarios. The latter incurs a *distributional shift* from the original scenarios, which can undermine the validity of using them [54, p. 1].

Moreover, even if we were to imagine a world in which we had infinite (1) time and (2) money, we would not be able to successfully account for every possible scenario. There will always be more, unforeseen permutations of actors and actions. This is a reality we need to deal with [25, p. 1]. One possible measure of remedying with this, could be to *decrease* the driveability of our existing scenarios. Decreasing the driveability is not the same as suddenly having access to the infinite set of possible scenarios, but it is reasonable to infer that being able to *test* the ADS (in a simulator) on these enhanced low-driveability scenarios will leave it better fit for encountering other low-driveability scenarios in the wild during operation. Having access to such a set of less driveable scenarios will allow ADS operators to test their ADS

that they assume to be working, and see if it still is able to handle all these more challenging scenarios. If it is not, they will have gained a meaningful insight into the workings of their ADS and can take action to remedy the fault before it causes harm in the real world. And if the ADS *does* still work, they can be more confident in their system.

Finally, edge cases can be a major issue for ADS adaptation. The *tail problem* as it is known in the Machine learning (ML) field posits that ML tasks are faced with a long tail of unseen cases. We can map these unseen cases, to our unseen ADS scenarios. Because of this, an ADS can be at risk of encountering an unseen edge case scenario during operation – something for which it might never have been tested [54, p. 1]. Arguing that the ADS would probably crash simply due to it finding itself in an unseen scenario is not logical. But it is important to keep in mind that the end we are pursing in the broader adaption of ADS, is increased safety and efficiency on our roads. Not sufficiently testing the ADS before deploying it would not serve our goal of increasing road safety – it would be a gamble with human lives. Not all edge cases are relevant for all contexts and environments. As such, ADS system developers could employ LLMs to obtain more variants of a certain scenario that they wish to test, if they were to only be in possession of a similar form. Say for example that they wished to obtain a set of scenarios in which certain properties are met, or certain situations occur.

Based on the problem description above, these are the formal research questions of the thesis:

RQ1. Can Large Language Models be used to decrease the driveability of Autonomous driving system simulator scenarios?

RQ2. Is it feasible to employ LLMs for obtaining unseen scenarios for ADS testing without human intervention?

1.3 Thesis overview

Following this Introduction chapter, the thesis is structured as follows:

- Chapter 2 → p.5 introduces key concepts related to ADSs and LLMs.
- Chapter 3 → p.17 reviews the current state of research in the field and discusses related applied works and lessons learned.
- Chapter 4 → p.24 details our proposed solution and its technical aspects.
- Chapter 5 → p.32 describes the experimental setup used to evaluate the solution.
- Chapter 6 → p.37 presents the findings from the experiments.
- Chapter 7 → p.48 analyses and contextualizes the results.
- Chapter 8 → p.59 suggests directions for future research.
- Chapter 9 → p.65 summarizes the main contributions and findings.

Two appendices are included: Scenario file diffs → p.75 and Scenario error messages → p.94.

Chapter 2

Background

> The limits of my language mean the limits of my world.
>
> Wittgenstein

This chapter will give an introduction to ADSs, testing, ADS testing and LLMs. This will lay the foundation for understanding the motivation of the project more in-depth, such that the appeal of the later solution proposal is clearer.

2.1 Autonomous driving systems

Autonomous driving systems (ADSs) are systems that enable automotive vehicles to drive autonomously. Due to the typical operating scenarios of a car it is pivotal that the Autonomous driving system maintain a high safety standard.

2.1.1 ADS simulation

Due to the complexity involved in testing Autonomous driving systems (Section 2.3.1 → p. 10), simulators are typically used for this purpose [33]. While the same points about not being able to test *all* possible scenarios do remain true for simulator-based testing due to the sheer number of factors, using a simulator allows for far greater testing at far lower cost due to the minimal overhead of (1) generating, (2) running, and (3) evaluating the outcome of test cases.

Furthermore, simulators allow for greater flexibility in determining the test scenarios due to not being confined by the physical world that is available to the scientist that wishes to perform the testing. Using a simulator, a

Europe-based scientist can test their ADS for North American conditions, or vice-versa.

Overview of ADS simulators

Due to the appeal of running ADS simulation, several contenders exist on the market, including several Free and open source (FOSS) options.

Utilising FOSS simulators allow for reviewing their code underlying code, which can serve to better explain why the ADS does what it does. It may also allow for modifying the code that is used for experimentation, to better fit the specific needs of the scientist at work.

That said, let us review some FOSS ADS simulators:

Carla is a widely used ADS simulator [13]. It is implemented using the game engine UnrealEngine [14] and allows for running test cases under various scenarios and collecting their results. Carla is fully open source and is under active development. It has been used in projects similar projects such as KITTI-Carla [11] and AutoSceneGen [1].

LGSVL is a deprecated simulator from LG [41]. It was used in projects such as DeepScenario [33]. It allowed for running various maps with various vehicles and tracking their data. It was also capable of generating HD maps [1]. DeepScenario is a project similar to this, concerned with testing Autonomous driving systems. Further details about it in are located in Related work → p. 18.

AirSim is Microsoft's offering [43]. It has, like LGSVL, been deprecated. It is also built using UnrealEngine. Unlike the other simulators we have seen, this also focused on autonomous vehicles outside of only cars, such as drones.

Concepts of ADS simulation

Ulbrich et al. draw up an outline for the terms *scene*, *situation*, and *scenario*, that are all concepts widely used in ADS simulation testing.

scene is a term that is used in different manners in various articles [48, p. 982], but Ulbrich et al. propose standardising the definition on *a scene describing a snapshot of the environment including the scenery and dynamic elements, as well as as all actors' and observers' self-representations, and the relationships among those entities* [48, p. 983].

situation is, like *scene*, employed in various fashions. Ulbrich et al. give a background detailing its usage ranging from *"the entirety of circumstances,*

[1] https://github.com/lgsvl/simulator?tab=readme-ov-file#introduction

which are to be considered by a robot for its selection of an appropriate behaviour pattern in a particular moment'[2], in Wershofen and Graefe [52, p. 3] to Schmidt, Hofmann, and Bouzouraa introducing a distinction between *the true world* in a formal sense, and that being the ground truth upon which a situation is described [42, p. 892].

Ulbrich et al. propose to standardise on the definition of a situation being *the entirety of circumstances, which are to be considered for the selection of an appropriate behaviour pattern at a particular point of time* [48, p. 985].

scenario refers to *'the temporal development between several scenes in a sequence of scenes'*[48, p. 986]. We note how the definition a a scenario utilises that of a scene. Furthermore, Ulbrich et al. hold it to be the case that *'every scenario starts with an initial scene. Actions & events as well as goals & values may be specified to characterize this temporal development in a scenario'* [48, p. 986], clarifying the distinction between a scenario and a scene.

Lastly they posit that a scenario spans a certain amount of time, whereas a scene has no such temporal aspect to it.

When running a simulation, we refer to the autonomous vehicle that is being simulated as the *ego vehicle* [17].

ADS scenario formats

OpenSCENARIO is a standard developed by the Association for Automation and Measurement Systems (ASAM), which is dedicated to the description of dynamic scenarios [7, p. 651]. Under this format, only the *dynamic* content of the scenario is recorded in the file. The static content is kept in other formats such as OpenDRIVER and OpenCRG [7, p. 652]. The simulator Carla (outlined in Section 2.1.1 → p. 6) supports this standard [7, p. 652].

Another widely popular scenario format is *CommonRoad* [31, p. 4941], first proposed in 2017 [2]. There are tools such as those proposed by Lin, Ratzel, and Althoff that allows for converting OpenSCENARIO scenarios to the CommonRoad format [31, p. 4941].

2.1.2 The concept of driveability

Driveability is a high-level estimator of the overall driving condition of an ADS, derived from several lower-level sources [19, p. 3140]. It can be used to refer to various aspects of a scene. Guo, Kurup, and Shah discuss the concept further, using the scene definition of Ulbrich et al. as outlined in

[2] The translation from German is borrowed from Ulbrich et al., [48, p. 984]

Section 2.1.1 →p.6, they describe how driveability can refer both to (1) road conditions, and (2) human driver performance. Guo, Kurup, and Shah go on to give an overview of how driveability can be used to refer to a (3) *driveability map* which divides a map into cells indicating where the ADS expects that it will be able to go, and (4) *object driveability*, which refers to the classification of physical objects in the environment that the ADS expects that it can run over without causing damage to the ego vehicle [19, pp. 3135–3136].

The main method for assessing the driveability of a scene comes form assessing the environment of the scene. Factors such as (1) weather, (2) traffic flow, (3) road condition, and (4) obstacles all play into this. The ADS infers information from observation [19, p. 3136].

They continue to give an overview of various *driveability factors* and their associated difficulties, using a a split between *explicit* and *implicit* factors.

Explicit driveability factors will typically include factors such as *Extreme weather* such as (1) fog, (2) heavy rain, (3) snow, all serving to impair road visibility and causing increased difficulties for vision-based tasks such as road detection and object tracking [19, pp. 3136–3137]. *Illumination* also poses various challenges for typical ADS tasks as a typical ADS will be required to operate in a plethora of scenes with varying degrees of illumination depending on factors such as time of day and location (e.g. if the ADS is operation in a dimly lit tunnel) [19, p. 3137]. The authors highlight how low illumination may serve as an advantage for the ADS as this allows for using the head lights of other vehicles as a feature for detecting them, whereas it make pedestrian detection significantly more challenging [19, p. 3137]. *Road geometry* is another external factor, satisfying our natural intuition that *intersections* and *roundabouts* are more difficult to drive through than straight highways [19, p. 3137].

Implicit driveability factors consist of behaviours and intent of other road users interacting with the autonomous car [19, p. 3138]. This includes the actions of other vehicles such as their (1) overtaking, (2) lane changing, (3) rear-ending, (4) speeding, and (5) failure to obey traffic laws . Guo, Kurup, and Shah call these factors *vehicle behaviours* [19, p. 3138]. Furthermore, *pedestrian behaviours* are also taken into account, noting how pedestrians can sometimes (6) cross the road, (7) be inattentive, or (8) fail to comply with the traffic law [19, p. 3138]. They go on to describe the *driver behaviour* of other drivers pointing out how (9) distraction, and (10) drowsiness can be factors that cause accidents even for ADS-enhanced vehicles due to the other, manual, cars interfering with their operation [19, pp. 3138–3139]. Lastly *motorcyclist/bicyclist behaviours* cause their own source of implicit driveability factors: The models and methods developed for analysing the

group's behaviour are far more limited than other groups of road users [19, p. 3139]. Guo, Kurup, and Shah theorise that this comes down to the lack of available datasets that capture and label the trajectories and behaviours of motorcyclists and bicyclists [19, p. 3139], causing potential issues for any ADS that wishes to operate in a shared traffic environment with this group.

2.2 Testing

We need to establish some basic testing concepts:

Pre- and post-conditions – When running test cases, the concept of *Pre-conditions* refers to certain properties that obtain *before* running a given test case. E.g that the ADS ego vehicle is stationary.

In many ways mirroring pre-conditions, *post-conditions* refers to the properties that obtain *after* having ran a test. E.g. that the ego vehicle will be moving after having performed the test.

Test coverage refers to the what degree the entire system is being tested. The concept can be used to describe both hardware and software test coverage [35, p. 187]. Malaiya et al. posit that hardware-based test coverage is measured in terms of the number of possible faults covered, whereas software-based test coverage is measured in terms of the amount of structural or data-flow units that have been exercised [35, p. 187]. A test case that exercised every single code line of the system would by definition have perfect test coverage.

2.3 Autonomous driving system testing

This section focuses on presenting an outline of the concept of *driveability*, and then the proceeding section will delve into further detail concerning aspects relating specifically to the *testing* of ADSs.

Testing is essential for assuring ADS operative safety [20, p. 163]. Several methods for ADS testing exist, all testing various aspects of the ADS. An ADS typically exists of several modules, all working together and handling different aspect of the ADS. In the context of this project, the module we target is the *motion planner* – the part of the ADS that is responsible for 'driving' the car. A common way to test the motion planner is to use simulator based testing [33, p. 1].

Huang et al. further outline several typical architectures for ADS testing, drawing on traditional software testing traditions outlining how *software*

testing can be used alongside more specialized ADS testing techniques such as *simulation testing* and *X-in-the-loop testing* [20, pp. 163–164].

2.3.1 The complexities of ADS testing

As we have seen, ADSs can perform several tasks, in several environments. As such, there are several relevant factors for testing them. It is not feasible to test all potential variations of all potential environments in the real world, meaning that the *test coverage*[3] typically will be low.

Some of the factors that complicate ADS operations are (1) timing, (2) sequence of events, and (3) parameter settings such as the different speeds of various vehicles and other actors.

Park, Yang, and Lim posit that *the concept of complexity exists everywhere, but there is no agreement on one for driving situations* [38, p. 1182]. Therefore they introduce their own concept of Driving situation complexity (DSC), which serves to give a metric of a the complexity of a given driving situation. Their DSC is defined as the output of a mathematical formula taking into account the perplexity and standard deviation of several control variables $\mathcal{M}$ representing the surrounding vehicle's behaviour [38, p. 1182]. Their formula also takes into account the ratio of *V2X*-capable vehicles [38, p. 1182], i.e. the vehicles that are connected and capable of communicating [50, p. 1].

2.3.2 Autonomous driving system testing metrics

When evaluating ADS testing, several metrics can be used. What metric to use will depend on what the relevant test is measuring.

Building on what we have learnt about driveability (Section 2.1.2 → p. 7), we take after Guo, Kurup, and Shah and review three metrics for quantifying driveability: (1) scene driveability, (2) collision-based risk, and (3) behaviour-based risk. Finally, we also investigate the metric (4) jerk.

Scene driveability refers to how easy a scene is for an ADS to navigate, and the *scene driveability score* refers to how likely the Autonomous driving system is to fail at traversing the scene [19, p. 3140]. It is typically found through and end-to-end approach. Note how this is a metric for *scenes*, without taking into account the performance of any specific ADS.

Collision-based risk comes in two kinds - (1) binary risk indicator, and (2) probabilistic risk indicator. Guo, Kurup, and Shah posit that the prior, binary metric, indicates whether a collision will happen in the near future

[3] See section 2.2.

in a binary 'either-or' sense, whereas the latter yields a probability calculated based on current states, event, choice of hypothesis, future states and damage [19, p. 3140].

Behaviour-based risk estimation also represents a binary classification problem wherein nominal behaviours are learnt from data, and then dangerous behaviours are detected on that. This requires a definition of 'nominal behaviour', which is typically defined on on acceptable speeds, traffic roles, location semantics, weather conditions and/or the level of fatigue of the driver [19, p. 3140]. Furthermore Guo, Kurup, and Shah describe how this metric also allows more than one ADS to be labelled as 'conflicting' or 'not conflicting' [19, p. 3140], representing a ruling on their compatibility. Finally, they note how behaviour-based risk assessment typically focuses on driver behaviours, not taking into account other actors in the scene such as pedestrians or cyclists.

Furthermore, *jerk* is a metric that renders the change of vehicle acceleration with respect to time. It has been used been used as a measure of the smoothness or abruptness of a movement in many domains such as the trajectory planning of the human arm and industrial robots [16, p. 126]. Jerk has also been shown to relate to a driver's physiological feelings of ride comfort [16, p. 126], giving it a clear relation to our previously stated definition of driveability. Feng et al. go on to posit that a goal of driving should be to minimize the jerk, as it both is both (1) linked to comfort, and (2) detection of safety-critical events [16, p. 126].

2.4 Large language models (LLMs)

Large Language Models (LLMs) are Machine learning (ML)-powered statistical transformer-based language models that typically contain several hundred billion parameters and are trained on massive text data [56, p. 4]. Base language models, as the name implies, *model language*. They are statistical models and as such, their output is not deterministic.

A Large Language Model is a neural network trained on big data [56, p. 3]. They expand on the older statistical language models by training on more data. This gives rise to *emerging abilities* such as in context learning [56, p. 3] (Emergent abilities → p. 12). These older statistical models are also neural networks, but they were impractical to train on large amounts of data. It was not until the seminal paper Attention is all you need [49] that a Google team headed by Vaswani et al. showed how neural networks can be trained in parallel using their new *attention* mechanism. This allowed for using amounts of data that was not technologically practical up until that point, opening the door for later advancements such as ChatGPT [56, p. 9]

The importance of training data

As a consequence of LLMs being statistical models of a certain input data [56, p. 1], what data the model is trained on is of great importance for the capabilities of the model [56, p. 6]. Zhao et al. give an overview of various LLMs and what kinds of corpora[4] they have been trained on [56, pp. 11–14].

The training data will provide the model with its base understanding of the world, and as such it will dictate (1) what it 'knows', and (2) how we should interact with it. E.g., if we want to solve problems related to software code, we should employ a model that has been *trained* on software code related topics so that the probability of it predicting correct tokens will be higher. If it has not seen any code during its training it would not have any base 'knowledge' for solving our problem, and its output would be bad. The LLM would however have no way of knowing if its output would be right or wrong, and we could say that it would have *hallucinated*. See General challenges with LLMs→p.15 for further information about hallucination.

2.4.1 Emergent abilities

Wei et al. outline how *emergent abilities* appear when scaling up language models [51, p. 1]. They define *emergent ability* to refer to abilities that are not present in smaller models, but present in the larger ones[51, p. 1], building on physicist Anderson stating that *Emergence is when quantitative changes in a system result in qualitative changes in behaviour.* [51, p. 2].

Furthermore, they discuss how *few-shot prompting* typically can achieve far superior results for harvesting LLM emergent abilities, whereas one-shot prompting can perform worse than randomized results [51, pp. 3–4].

They continue outlining several approaches for achieving augmented prompting strategies, underlining how (1) multi-step reasoning (2) instruction following (3) program execution, and (4) model calibration all serve as possible ways of increasing LLM performance [51, p. 5].

2.4.2 Intelligence in LLMs

There are three theories on machine intelligence, each serving to explain how they '*think*': (1) stochastic parrot (2) Sapir-Whorf hypothesis, and (3) conceptual blending.

[4] A corpus (pl. corpora) refers to a document collection.

Stochastic parrot

Bender et al. outline how LLMs can *fool* humans as they are trained on ever larger amounts of parameters and data, appearing to be in possession of an intelligence [4, pp. 610–611].

This anticipates the phenomenon of hallucination (Section 2.4.4 → p. 15).

Sapir-Whorf hypothesis

The Sapir-Whorf hypothesis posits that *The structure of anyone's native language strongly influences or fully determines the world-view he will acquire as he learns the language.* [5, p. 128].

We note how this maps to our LLMs, indicating that they will only ever be able to 'know' the data on which they have come into contact with.

Or: *Language* defines the possible room for *thought*.

Conceptual blending

Conceptual blending is a theory on intelligence. It refers to the basic mental operation that leads to new meaning or insight that occurs when one identifies a match between to input mental spaces, to project selectively from those inputs into a new 'blended' mental space [15, pp. 57–58].

This phenomenon explains how we are able to imagine phenomena that logically should not exist such as *land yacht* (Land yacht conceptual blend → p. 14)

We note how this is how LLMs operate when processing vectorized linguistic data.

2.4.3 Utilising LLMs - Prompt engineering

A typical way of interacting with LLMs is *prompting* [56, p. 44]. You prompt the model to solve various tasks. As we saw in Emergent abilities → p. 12, the level of performance you are able to extract from your Large Language Model can depend a great deal on how you interact with it. The process of manually creating a suitable prompt is called *prompt engineering* [56, p. 44]. Zhao et al. outline three principal prompting approaches:

In-context learning (ICL) is a representative prompting method that formulates the task description and/or demonstrations in natural language text [56, p. 44]. It is based on *tuning-free prompting* and it, as the name implies, never tunes the parameters of the LLM [32, p. 15]. One the one hand, this allows

[5] Diagram borrowed from Fauconnier and Turner, [15, p. 67].

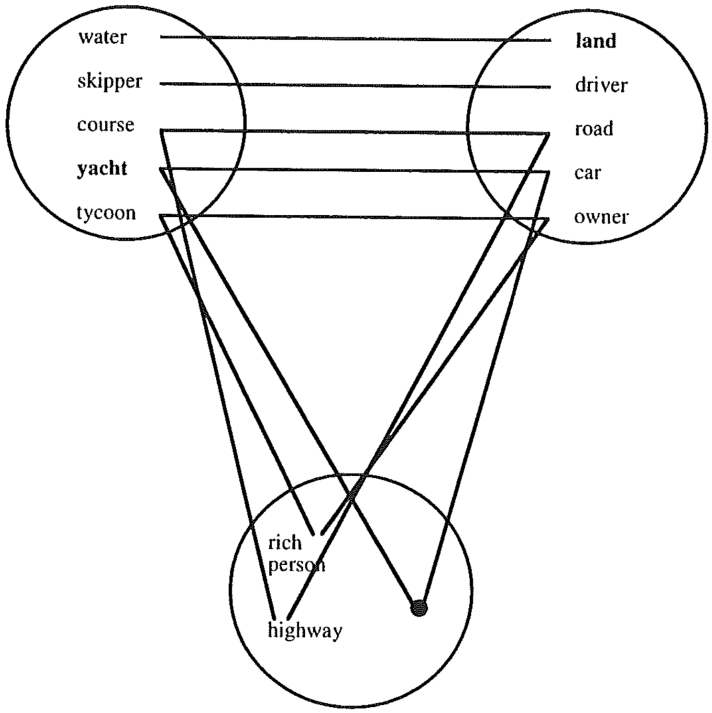


Figure 2.1: The conceptual blend of a *land yacht*[5]

for efficiency, but on the other hand, heavy engineering is typically required to achieve high accuracy, meaning you must provide the LLM with several answered prompts [32, p. 16]. In layman's terms, ICL entails including examples of the process you want the model to perform when prompting it.

Chain-of-Thought (CoT) prompting is proposed to enhance In-context learning by involving a series of intermediate reasoning steps in prompts [56, pp. 44, 52]. The basic concept of CoT prompting, is including an actual Chain-of-Thought inside the prompt that shows the way form the input to the output [56, p. 52]. Zhao et al. note that the same effect can be achieved by including simple instructions like '*Let's think step by step*' and other similar 'magic prompts' in the prompt to the LLM, making CoT prompting easy to use [56, p. 52].

Planning is proposed for solving complex tasks, which first breaks them down into smaller sub-tasks and then generates a plan of action to solve the sub-tasks one by one [56, pp. 44, 54]. The plans are being generated by the LLM itself upon prompting it, and there is a distinction between text-based and code-based approaches. Text-based approaches utilise natural language, whereas code-based approaches utilise executable computer code [56, pp. 54–

55].

2.4.4 General challenges with LLMs

We have seen that LLMs demonstrate promising abilities (Emergent abilities →p.12) But they have nevertheless certain issues attached to them that we need to be aware of.

Hallucination

As we saw in Section 2.4.2 →p.13, LLMs are prone to *bullshitting*. They have no intuition of, or concern with *the truth*. They only ever yield whatever response is the most probable under their BEAM SEARCH algorithm being applied on their training data.

Environmental concerns

A University of Rhode Island study on the environmental impact of LLMs have shown that they require wast amount of energy and water [22]. They also found that the different LLMs may differ greatly in their energy consumption, highlighting that that certain LLMs may consume more than 70 times more energy than others [22].

Another study by Tomlinson et al. focusing specifically on *carbon emissions* did however find that these emissions significantly lower for LLMs than humans for specific tasks such as text and image generation, ranging from 130 to 2900 times less Co2 emitted depending on the task [47, p. 1].

Li et al. surveyed the water consumption of LLMs, finding that training the LLM GPT-3 could evaporate as much as 700 000 litres of clean freshwater [28, p. 1]. Furthermore they review the trends of current AI adoption and project that the water consumption of AI could reach levels as high as 4.2 - 6.6 billion cubic metres by 2027, which is comparable to 4 - 6 Denmark's, or half of the United Kingdom [28, p. 1]. Recent research indicates that *serving* LLMs currently account for more emissions than training them [12, p. 37].

Efforts to achieve greener LLMs have been proposed by Li et al., while recognizing the trade-off between ecological sustainability and high-quality outputs [27, p. 21799].

2.4.5 The different kinds of LLMs

There are several available LLMs, some of which are open source, and some proprietary. Open source LLMs afford greater insight into their composition and underlying training data, whereas proprietary models appear more like

black boxes. Some popular model families include the GPTs, Gemini, Llama, Claude, Mistral, and DeepSeek.

The LLMs differ primarily in their (1) parameters, and (2) training data. As we saw above, all typical LLMs utilise a transformer-based neural network. But due to their various different properties, different models can behave differently for different tasks regardless of their similar architecture.

What they all share is their ability to perform *inference*, meaning that they predict output tokens given some input tokens (see Section 2.4.2 → p. 13).

2.5 Existing LLM applications for ADS

Cui et al. give a broad overview of some of the ways LLMs have been applied for ADSs, highlighting some of the opportunities and potential weaknesses of LLM applications for ADS purposes. One of the ways LLMs can be applied, is for adjusting the driving mode, or aiding in the decision-making process [10, p. 1]. Cui et al. delve further into these aspects in their other work "Drive As You Speak: Enabling Human-Like Interaction With Large Language Models in Autonomous Vehicles", providing a framework for integrating Large Language Model's (1) natural language capabilities, (2) contextual understanding, (3) specialized tool usage, (4) synergizing reasoning, and (5) acting with various modules of the ADS [9, p. 1].

This will be reviewed in greater detail, along with other such projects in Related work and literature review → p. 17.

Chapter 3

Related work and literature review

> Learn from the mistakes of others. You can't live long enough to make them all yourself.
>
> Eleanor Roosevelt

This chapter surveys several related works and does a literature review. It contains a selection of works that are typically related to applying LLMs specifically or ML more generally to ADS simulator scenarios.

3.1 Literature review

This section surveys the current state of the research field with a theoretical perspective. Applied pieces of work are saved for later, to be surveyed in the latter part of the chapter.

3.1.1 Survey of LLM applications in scenario-based ADS testing

Zhao et al. give an extensive overview of some of the various ways that LLMs have been applied to scenario based testing of Autonomous driving systems. The authors classify the various research efforts based on (1) how they have employed the LLM, and (2) to what end [57]. Their survey is continually updated, the last update having been made 2 months before the time of writing[1]. This entails a certain overlap with some of the works we review in Related work →p.18.

[1] I.e. as of September 17th 2025, the last update to their Github repo was on July 23rd, 2025. The paper on Arxiv was last updated May 22nd 2025.

Not deterred by this, let us delve into the survey: They start by highlighting the trend between the number of LLM surveys, and ADS surveys – while the trend was increasing from 2020-23, there was an explosion in 2024, with about 200 works concerning applying LLMs for Autonomous driving system purposes being published [57, p. 1, figure (b)]. Furthermore, the number of ADS studies has remained steady over the last 4 years, whereas the number of LLM studies has exploded in popularity [57, p. 1, figure (a)]. This indicates that a significant amount of the scientific effort around ADSs the last year, has been concerned with utilising LLMs.

The article summarizes the field, pulling together various surveys of the related subfields. Those being (1) LLM surveys, (2) surveys of scenario-based testing, (3) general cases of LLMs for ADSs, and finally (4) a broader review of surveys of LLMs being applied for *miscellaneous domains* , for each highlighting their specialized foci [57, p. 2].

3.1.2 LLM4AD

LLM4AD is a paper that gives a broad overview of LLMs for Autonomous driving system. It touches on several of the various ADS applications where LLMs are relevant such as (1) language interaction, (2) contextual understanding, (3) zero-shot and few shot planning allowing LLMs to perform tasks they weren't trained on, helping with handling edge cases (4) continuous learning and personalization, and finally (5) interpretability and trust [10, p. 2]. Furthermore, the authors also propose a comprehensive benchmark for evaluating the instruction-following abilities of an LLM based system in ADS simulation [10, p. 1].

3.2 Related work

Having obtained an overview of the current state of the literature, we proceed to surveying several pieces of applied works. Here, they are categorized broadly with regard to what they do and how the do it.

There is some overlap between some of the works and several of the categories. A work gets allocated to the category in which that fits the best with regard to the *focus* of the contribution of the work.

3.2.1 ADS scenario generation

The following works relate to generating ADS test scenarios using traditional ML techniques.

Dataset and toolset – DeepScenario

DeepScenario is both a dataset and a toolset aimed at Autonomous driving system testing [33]. The principal value proposition of this work lies in recognizing the fact that (1) there are an infinite number of possible driving scenarios, and (2) generating critical driving scenarios is very costly with regard to time costs and computational resources [33, p. 52]. The authors therefore propose an open driving scenario of more than 30 000 driving scenarios focusing on ADS testing [33, p. 52]. The project utilises traditional machine learning methodologies, having been performed prior to the broad adaptation of LLMs.

Its scenarios are intended for the simulator SVL by LG (Section 2.1.1 → p. 6).

Test case specification language – RTCM

RTCM is a ADS testing framework that allows the user to utilise natural language for synthesizing test cases. The authors propose a domain-specific language — called RTCM, after Restricted Test Case Modelling — for specifying test cases. It is based on natural language and composed of (1) an easy-to-use template, (2) a set of restriction rules, and (3) keywords [55, p. 397]. Furthermore, they also propose a tool to take this RTCM source code as input and generating either (1) manual, or (2) automatically executable test cases [55, p. 397]. The proposed tools were evaluated in experiments with industry partners, successfully generating executable test cases [55, p. 397].

Generating crash scenarios – DeepCollision

Lu et al. utilise Reinforcement learning (RL) for ADS testing, with the goal of getting the ADS to *collide*. They used *collision probability* for the loss function of the Reinforcement learning algorithm [34, p. 384]. Their experiments included training 4 DeepCollision models, then using (1) random, and (2) greedy models for generating a baseline to compare their models with. The results showed that DeepCollision demonstrated significantly better effectiveness in obtaining collisions than the baselines. While not specifically focused on *testing*, we recognize that their work is thematically similar to our project.

3.2.2 Utilising LLMs on ADS scenarios

The remaining works all relate to utilising LLMs for various purposes related to ADS scenarios.

AutoSceneGen

AutoSceneGen is a framework for ADS testing using LLMs, focusing on the motion planning of Autonomous driving system [1, p. 14539]. Aiersilan highlights how LLMs provide opportunities for efficiently evaluating ADS in a cost-effective manner [1, pp. 14539–14540]. They generate a substantial set of synthetic scenarios and experiment with using (1) only synthetic data, (2) only real-world data, and (3) a combination of the 2 as training data. They find that motion planners trained with their synthetic data significantly outperforms those trained solely on real-world data [1, p. 14539].

LLM-Driven testing of ADS

Petrovic et al. worked on using LLMs to for automated test generation based on free-form textual descriptions in the area of automotive [39, p. 173]. They propose a prototype for this purpose and evaluate their proposal for ADS driving feature scenarios in Carla. They used the LLMs GPT-4 and Llama3, finding GPT-4 to outperform Llama3 for the stated purpose. Their findings include this LLM-powered test methodology to be more than 10 times faster than traditional methodologies while reducing cognitive load [39, p. 173].

Requirements All You Need?

Lebioda et al. provide an overview of LLMs for ADS in their recent preprint *Are requirements really all you need? A case study of LLM-driven configuration code generation for automotive simulations*[2], focusing on LLM's abilities for translating abstract requirements extracted from automotive standards and documents into configuration for Carla (Section 2.1.1 $^{\rightarrow p.6}$) simulations [26]. Their experiments include employing the *autonomous emergency braking* system and the sensors of the ADS. Furthermore, they split the requirements into 3 categories: (1) vehicle descriptions, (2) test case pre-conditions, and (3) test case post-conditions [26]. The preconditions they used included (1) agent placement, (2) desired agent behaviour, and (3) weather conditions amongst others, whereas their postconditions reflected the desired outcomes of the tests, primarily related to the vehicle's telemetry [26].

Language Conditioned Traffic Generation

Tan et al. look into using LLMs to generate specific traffic scenarios. They identify the importance of being able to use simulators to test ADSs, and highlight how test scenarios are expensive to obtain [44, p. 1]. To this end, they propose a tool – LTCGen which employs the strengths of LLMs to match a natural language query with a fitting underlying map[3], and populates

[2]This was submitted to Arxiv on 2025-05-19.

[3]Map as in a *world* in which a scenario can take place.

this with a (1) initial traffic distribution, and (2) the dynamics of all the vehicles involved in the scene. Something to note is that they generate their scenarios, without initially taking the *ego vehicle* into account. The ego vehicle of the scene is simply determined as the vehicle that is in the *centre* of the first *frame* [44, p. 3].

Scenario engineer GPT

Li et al. outline a framework for utilising the LLM-backed ChatGPT in order to generate scenarios. They propose SeGPT – a scenario generation framework that they found to yield *significant progress in the domain of scenario generation* [30, p. 4422]. They posit that their prompt engineering ensures that the generated scenarios are authentically diverse and challenging [30, p. 4423]. The focus is primarily on *trajectory scenarios* [30, pp. 4422–4423].

Note how they explicitly mention scenario *generation*. Our approach for this project has a different angle, with the focus being on modifying *existing* scenarios. More on this in LLM4DD implementation → p. 24. The difference between generating a 'brand new' scenario with a model trained on existing scenarios, and modifying an existing scenario seems like a matter of granularity. These are very similar concepts, only that the enhanced scenario will have more common DNA whereas the other 'new' scenario will consist of a broader range of DNA from its various underlying scenario corpora.

LLM driven scenario generation

Chang et al. also look into using Large Language Models to generate ADS scenarios. They recognize several of the challenges we outline in Section 1.2 → p. 2. In their 2024 paper, they propose LLMScenario, which is an LLM-backed framework for both (1) scenario generation, and (2) evaluation feedback tuning [6, p. 6581].

They analyse scenarios in order to provide the LLM with a minimum baseline scenario description, and propose score functions based on both (1) reality and (2) rarity. Their prompting is based on Chain-of-Thought (CoT) and a posteriori empirical experience. Lastly, they tested several Large Language Models for their experiments. Their results were positive, indicating effectiveness for scenario engineering in Industry 5.0 [6, p. 6581].

Chat2Scenario

Zhao et al. propose a method for utilising LLMs to retrieve ADS scenarios given a natural language query. Their framework synthesizes scenarios from naturalistic[4] driving datasets, based on observation real world human driv-

[4] Their term. The intended meaning of *naturalistic* is not all clear to me.

ing [58, p. 55], that it then uses as a database for retrieving the scenario that best matches the user's natural language query. Furthermore, they employ traditional techniques for asserting the relevance of the retrieved scenarios, allowing the user to specify a set of *criticality metrics*, of which a certain threshold must be reached amongst the scenarios that are initially retried by the LLM, pruning false positives. As a measure to increase the usability of their framework, they also provide a web-app with an intuitive GUI for both (1) operating the tool, and (2) visualizing the scenarios [58, p. 560].

In order to allow the LLM to determine whether a scenario is relevant under the provided query, they put forward a method for classifying the various scenarios using traditional ML techniques. This classification focuses primarily on highway scenarios and the activities of other actors in relation to the ego vehicle [58, pp. 561–562].

The project's prompts are 'informed' by the 6 OpenAI guidelines from their prompt engineering guide[5], ending up with a structured prompt of 5 segments. These segments serve to guide the LLM, delineating its role as an 'advanced AI tool for scenario analysis, specifically tasked with interpreting driving scenario following a pre-established classification model' [58, p. 562]. They then input the user-provided description of the scenario they wish to retrieve. Following this, a third segment declares the format for the LLM response, followed by a prime example of In-context learning, demonstrating what a satisfactory fulfilment of the desired format could look like. Lastly they instruct the LLM to *Remember to analyse carefully and provide the classification as per the structure given above* [58, p. 563].

Predicting driving comfort in autonomous vehicles using road information and multi- head attention models

The 2025 article of Chen et al. [8], delves into the various aspects related to predicting driving comfort in autonomous vehicles based on (1) available road information, and (2) multi-head attention models. Their principal focus is on driving *comfort*. To this end, they evaluate ADSs in light of the **jerk** metric in various situations. Furthermore, they highlight how a high complexity in the scenarios can increase the probability of emergency breaking occurring, which is naturally antithetical to comfort for the ADS operator and their passengers.

[5] https://platform.openai.com/docs/guides/prompt-engineering (URL from the paper.)

In order to measure this comfort, they rely metrics calculated from data-points from the ADS system – jerk and acceleration. This, they use in conjunction with manual human driving evaluation scores, to compose a new metric, the 'driving comfort evaluation score' (DCES) [8, p. 10].

Moreover, they use this information to propose a model – the Autonomous driving comfort prediction (ADCP) model – for *predicting* driving comfort from road information [8, p. 2].

Chapter 4

LLM4DD implementation

> The only difference between a problem and a solution is that people understand the solution.
>
> Charles F. Kettering

This chapter will describe our proposed solution for the problem presented in the Introduction →p.1. This implementation will serve as a testbed for the experiments for obtaining our experiment Results →p.37 that we later analyse and evaluate (Chapter 7 →p.48).

Recall that have seen in the Introduction →p.1 and Background →p.5 that there are several complexities involved with ADS testing, such that we typically use simulator-based testing to aim at verifying the safety of the ADS before it gets deployed to the real world. But then we saw further that simulator-based testing can lead to a false sense of safety due to the ADS passing all our test scenarios, and then there can be a hole in what they test for, so that the ADS may actually have undiscovered faults that we haven't caught on to.

Furthermore, we saw that LLMs have capabilities for modifying textual data (such as e.g. scenario definitions) by natural language prompt engineering. In Related work and literature review →p.17 we surveyed several projects that aim to enhance scenario-based testing of ADSs, and we saw that some of them employed LLMs. But no extant research has specifically used LLMs to *decrease driveability* of ADSs scenarios[1].

This research gap is addressed by this thesis – we propose a novel LLM-powered methodology for decreasing the driveability of ADS scenarios so

[1]The most similar work, to the author's knowledge, is [54] – *AGENTS-LLM: Augmentative GENeration of Challenging Traffic Scenarios with an Agentic LLM Framework*.

that when the ADS is tested with the less driveable test scenarios, we can either (1) trigger it to fail and analyse what went wrong, or (2) be more confident in the ADS being able to operate in complex scenarios due to passing them.

To this end, we propose LLM4DD[2] – a tool for (1) running a base ADS test case, (2) enhancing the test case using LLMs, (3) running the enhanced test case, and (4) comparing the results of the two runs. We summarize the motivation behind using the LLM4DD tool in the following user story:

(1) I have a set of ADS test scenarios. I provide this set to LLM4DD. It will run the entire set, and generate a baseline of my ADS performance.

(2) LLM4DD will then improve my test scenarios using LLMs to make them less driveable, and run the enhanced versions.

(3) Lastly LLM4DD will report how the results differ from running the base and enhanced version of a test case.

(4) This will give me insight into my ADS by reviewing what scenarios it failed to complete. Then I can look into the cause of the error state and uncover underlying faults in the ADS that i did not know about beforehand. If the ADS *is* able to complete the less driveable scenarios, I can be more confident in it and be more assured that it will work properly during real-world operation.

The tool follows a natural pipeline structure. We have some base test scenarios that need to be run in order to get a baseline for the results, we then have to enhance these, and run the improved versions and compare them to their original versions. Experimenting with this tool will allow us to learn the extent to which LLMs can be applied for decreasing driveability in ADS test scenarios, satisfying the Problem description → p. 2. The tool is implemented in an LLM- and scenario-agnostic fashion so that it is scalable to other combinations of LLM and scenario formats than those experimented with in this specific work to verify the feasibility of the tool.

Figure 4.1 → p. 26 renders an example of how it can appear when executing a test scenario on the Carla simulator. Runs like this will later be presented in Results → p. 37 and then analysed in Discussion → p. 48 to evaluate the value of the LLM4DD tool.

[2] 'LLMs for decreased driveability'.

Figure 4.1: A screenshot from executing a Carla scenario.

4.1 Architectural overview

In order to decrease the driveability of the ADS scenarios, we need 3 separate components: (1) something to handle the LLM interfacing, (2) something that can integrate with the ADS simulator, and finally (3) some kind of human-facing interface to administrate the process. By compartmentalizing what component has responsibility for what task, we reduce complexity and increase the possibility of repurposing the modules for other possible tasks in the future. UNIX philosophy!

For the sake of making their roles more clear, we christen the components as follows: (1) Odin will be the module for interfacing with LLMs and performing the enhancement. (2) Thor will take ADS scenarios, run them on a simulator and report the results. Finally, (3) Loki will interface with the human and start the process by determining what LLM is to be used with what prompts with what scenarios. These components and their relationship is rendered in Figure 4.2 → p. 27.

All the prompts exists as a part of the Odin module. Loki will request available prompts from Odin on behalf of the user, and the user will select which one they want to use. Similarly, all the scenarios exist within the Thor module and are handled by it. As for the prompt, the user requests a list of available scenarios from the Thor module and makes a decision on which they want to use.

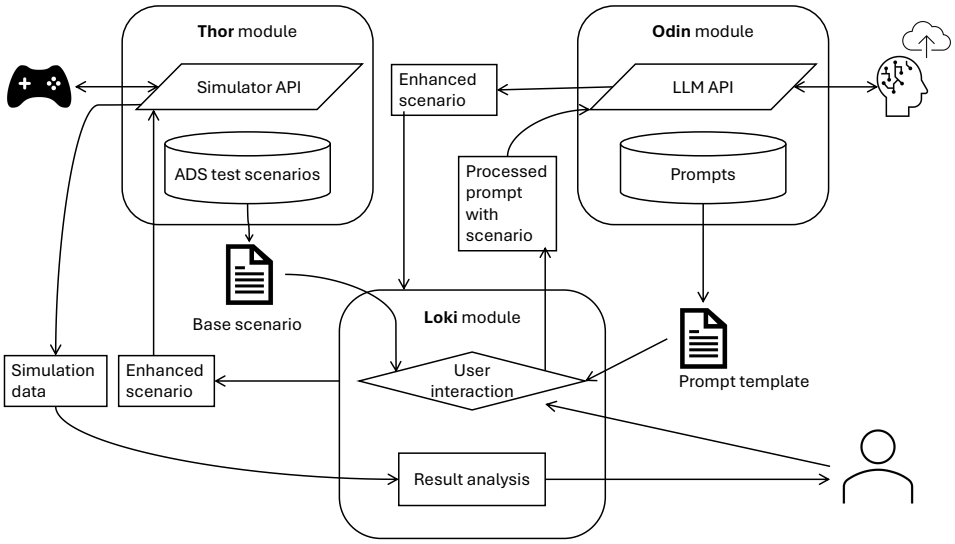


Figure 4.2: LLM4DD pipeline architecture

The programming language PYTHON is widely used for ADS simulation. It is a high level language, allowing the user great flexibility and developer experience. For this reason, it is the de facto goto-language for ADS simulation purposes and LLM4DD is also implemented using that language.

All code is available on the GitHub repo LLM4DD.

4.2 Component details

All the components are intended to be ran as separate containerised processes on Docker, allowing for scaling them up to industrial scale in a Kubernetes (K8s) setting. Docker is a piece of software that allows a piece of software to run in a reproducible *pod* which allows for easy dependency management and scaling. The motivation for using K8s technology in this project would lay in spinning up several Odin and Thor pods, allowing for processing several scenarios in parallel even if there is only 1 Loki client.

Let us now focus on the individual components and analyse how they solve their given tasks and how they all fit together to make up the total LLM4DD pipeline tool.

4.2.1 LLM interface and prompt applications – Odin

The Odin module handles all things LLM. It provides a unified API for applying various prompts to scenarios and returning the enhanced output resulting from having applied the prompt. We have implemented support for

the LLMs that are available on Ollama, and Gemini. This allows for testing with LLMs such as `Mistral 7.2B`, and `gemini-2.5-flash`.

Note that we don't employ any traditional Natural language processing (NLP) techniques related to e.g. tokenization or input processing – we leave this up to the internal mechanisms of the LLMs.

In order to facilitate testing various LLMs, the module is written to be as general as possible, exposing a simple API that can be used with all various LLMs with minimal modification necessary. This allows for experimenting with several LLMs that may all have different internal Application programming interfaces (APIs) without having to modify the major underlying code for specific APIs. This saves time, and prevents repetitive manual work.

Therefore, interfacing with LLMs is the principal task of the Odin module. It offers a simple API to Loki, and then Odin itself will handle all the complexities of making the LLM work with the provided prompt etc. As mentioned above, the current version of the LLM4DD tool has support for Ollama and Gemini APIs. Let us review how this has been achieved:

Gemini integration

The Gemini integration is quite straightforward, relying on Google's own `genai` Python module. The one piece of complexity to not is that it requires that the user provides their own Gemini API key and has this set as an environment variable with the proper name. Without this being as it should, the script has been written to hard crash, as it would not possible for it to complete the desired LLM enhancement regardless as long as the API key is not present.

Ollama integration

The Ollama integration is a bit more cumbersome. This mostly comes down to it not using any existing library modules for this specific purpose, instead relying on using the `json` and `requests` modules to implement the desired functionality from scratch, making it so that we need to handle network IO and marshalling the LLM response into a fitting return buffer.

Its complexity arises principally from 2 major factors – (1) the already mentioned manual networking, and (2) having to parse the streamed response Furthermore, this code expects that the user already *has* an Ollama installation running on their host machine. The code provides no means of setup for this – that is an entirely external endeavour that is left up to the end user.

Similarly to how the Gemini implementation does it, this will crash if Ollama

is not functioning properly as it would not possible for it to complete the desired LLM enhancement regardless if Ollama is unreachable.

Prompts and their associated code

In this project, the prompts are the instruction to the LLM for applying the enhancement to the scenario. Quite possibly the most critical piece of code related to the experiments, outside of the base LLM integration that allows for running prompts altogether. They need to take the base scenario as an input and integrate it into the LLM context, such that it knows what it shall use as its base to apply enhancements that will decrease the driveability. For this reason, the Odin module also provides certain scenario utilities, for facilitating prompt operations. As mentioned, the prompts need to include the scenarios *in* them, so that they are accessible to the LLM. The most interesting aspect about how we do this, is how the prompts are stored in the system as lambda functions. This makes it so that they can take an argument that represents the scenario and simply *execute* the function to insert the scenario into the prompt. This is then inserted into the output prompt. This is a very intuitive concept that might best be *seen* in order to grasp it.

Listing 4.1 renders an example how the prompts are stored in the code base. As mentioned, this is used in a such a manner that the ADS scenario is provided as an argument and is then inserted on that which is line 4 in this example listing. This generalization of inserting the scenarios into template strings allows for using the same prompts for all scenarios without having to modify the underlying code.

```
1    lambda python_carla_scenario_raw: f"""
2        1 - Context: We are working with a driving simulation environment for the Carla
      simulator.
3        2 - Task: Decrease the driveability of the scenario by enhancing it with more details
      and complexity.
4        3 - Input: {python_carla_scenario_raw}
5        4 - Output: An enhanced version of the scenario description with additional
6        details and complexity, still in Python carla scenario format.
7    """,
```

Listing 4.1: An example prompt.

4.2.2 Carla interface and scenario utilities – Thor

The Thor module is responsible for all things related to the Carla ADS simulator. It provides the client with several scenario-related utilities, and is capable of executing the desired scenarios.

For the reasons we have seen in Section 2.1.1 → p. 6, we want to run our scenarios on Carla. It is the best offering as it is open source, under active development and has a feature rich Python API with a first party 'scenario runner' module.

Executing ADS scenarios on Carla is quite straightforward when using Carla's existing Scenario Runner module's functionality. We have made our own fork of this in order to make it behave in a way that makes the most sense for the LLM4DD situation[3].

We record a plethora of datapoints when executing scenarios on the simulator[4]. The way the Carla simulator works, one simulator run can be analysed several times post factum. The entire scenario execution is stored in a Carla-specific binary format. This binary file can then later be analysed, extracting various metrics from one run. This saves time not having to run the simulator more than necessary, and allows for reproducing the metric calculations from the original underlying binary log file.

Certain of the Thor utilities are simple tools for asserting the liveness of Carla, such as the `get_carla_is_up` function. This function will use the Carla standard Python library and attempt to connect to the server on its default port, i.e. 2000, as per the Carla documentation. Note that we refer to the host as simply `carla` – this is possible due to the entire project running containerised with Docker Compose. Instead of referring to the specific IP address of the Carla server (typically localhost, if not running it externally), the Docker system will facilitate this name translation for us.

The Thor healthcheck is used both to assert the general liveness of the pipeline of the LLM4DD pipeline, and to verify that the simulator is available before performing experiments. By 'liveness' we mean to check that all components are running and ready to process work. Essentially checking that they haven't crashed. It is better to detect this illegal crash state *before* running experiments rather than during their execution as that would cause (1) cumbersome debugging, and (2) wasting time.

As described in the Introduction → p. 1, we use the *jerk* metric as a proxy for ride quality and safety. It is implemented as an extension to the Carla Scenario runner software suite. Using this, we can compare the results from running the baseline unmodified test case and comparing it with the results from running the LLM-enhanced version and returning to the user with regard to driveability. Jerk is to be calculated *after* having executed the scenario, utilising the Carla binary file described above.

[3] This fork is available on https://github.com/orjahren/scenario_runner-LLM4DD. It was mostly used for debugging purposes and the LLM4DD tool should work just fine with the stock version as well.

[4] This is again provided by the Carla software suite. A complete overview of data points is provided in the documentation, see e.g. https://carla.readthedocs.io/en/0.9.15/adv_recorder/

4.2.3 Execution tool / user oriented frontend – Loki

The final module of the LLM4DD pipeline is Loki – it is simply a tool intended to be used by the user for operating the process. It (1) says what scenarios are available to it (i.e. those that are eligible for being enhanced), and (2) allows the user to select a prompt and (3) execute that prompt to the scenario of their choosing.

The Loki module relies on the Odin and Thor modules for all essential functionality, which is in line with what is to be expected as this is simply a frontend client to *reach* them. For these reasons, it's a quite simple Python script that relies on RPC to the APIs of the other components of the pipeline.

Chapter 5

Experiment methodology

> The torment of precautions often exceeds the dangers to be avoided. It is sometimes better to abandon one's self to destiny.
>
> Napoléon

This chapter will describe the experimentation that has been done with the implemented solution proposal (Chapter 4 →p.24), describing the methodological process that was undertaken in order to obtain the Results →p.37. This chapter lays the groundwork for presenting them succinctly with their relevant context in Chapter 6 →p.37 and then analysing them in Chapter 7 →p.48.

5.1 ADS-related aspects

This section surveys ADS-related aspects from the experimentation, detailing the experimentation related to scenarios and metrics.

5.1.1 Scenarios

We do naturally have to walk before we can run. For this reason, the tool will initially be tested on simple test scenarios provided by people behind the Carla simulator. When we have verified that the project is sufficiently working for its stated purpose, we can scale up the activities to other datasets. Several are presented in Related work →p.18. The concept of applying LLMs to ADS scenarios is quite universal in nature and is eligible for application for virtually *all* datasets.

The experimentation for developing and evaluating the LLM4DD tool focused on the scenarios `Accident`, `CutIn`, `NoSignalJunctionCrossing` and `FollowLeadingVehicle`.

5.1.2 Metrics

The goal of this project is to decrease the driveability of the scenario such that we are able to 'flag' potential issues in the ADS. As 'driveability' is a very broad concept (see The concept of driveability → p. 7), there are several relevant metrics and datapoints that can potentially be used.

For this project, due to its broad scope, a similarly broad metric will be used for indicating that an experiment has yielded a meaningful result – jerk. As outlined in Autonomous driving system testing metrics → p. 10, jerk has been shown to be related to several factors contributing to driveability.

Again owning to the broad scope, exact jerk *numbers* are not relevant – for our purposes, it suffices to have a binary relation to the jerk being either (1) more or less unchanged[1], or (2) worsened. If the driveability has decreased, we will have found a result.

Lastly, it's worth noting that qualitative analysis by intuition from visually inspecting the scenarios is also useful here. If the LLM has introduced an obstacle in the course of the road where the ADS is to drive, we intuitively know that this enhanced scenario is more complex and less drivable – which also indicates a result.

5.2 LLM-related aspects

The last section of this chapter explores aspects related to the LLM usage in the experiments, detailing the usage of prompts and determining what LLM is to be used, along with some preliminary results.

5.2.1 Prompts

Prompting is our principal way of interfacing with the LLM. For this reason, our results depend on (1) good, and (2) fitting prompts. Without this, we won't get far.

We therefore propose several prompting strategies, taking after related research (Related work → p. 18).

Prompts were determined by trial and error in an iterative manner, in conjunction with GitHub Copilot. They are all descendant of listing 5.1, each subsequent iteration improving on the last based on what worked or did not worked when assessing the output. Due to a technical detail of the LLM4DD implementation (LLM interface and prompt applications – Odin → p. 27), the datatype of the prompt is a lambda function that takes the raw scenario

[1] Keep in mind that ADS motion planners are indeterministic

represented as a string and then inserts it into the prompt in runtime. This is represented by the curly braces on line 3 in listing 5.1.

```
1 lambda python_carla_scenario_raw: f"""
2 1 - Context: We are working with a driving simulation environment for the Carla simulator.
3 2 - Task: Decrease the driveability of the scenario by enhancing it with more details and
      complexity.
4 3 - Input: {python_carla_scenario_raw}
5 4 - Output: An enhanced version of the scenario description with additional
6 details and complexity, still in Python carla scenario format.
7 """,
```

Listing 5.1: The first prompt.

5.2.2 Finding a suitable LLM

As we learnt in Section 2.4.5 → p. 15, there are several LLMs extant. We should be able to experiment with various different LLMs to maximize our chance of testing with a 'good' LLM that goes well with our stated purpose. Due to the LLM-agnostic implementation of the `Odin` module, this is quite straightforward.

The experiments were first carried out using a locally hosted 7.2B parameter Mistral model. This model is interesting in that it has been shown to outperform significantly larger models across various benchmarks[2]. Similarly, the Gemini model `Gemini 2.5 flash` running on Google's infrastructure was used. This is a mid-size multimodal model that supports up to 1 million tokens, released in June of 2025, with support for thinking and long contexts[3].

All data presented in the Results → p. 37 chapter, are obtained using the Gemini model.

5.2.3 Output of the LLM – general overview

The following reviews (1) what works well, (2) why it works, and (3) what does *not* work and (4) why this is.

Depending on the prompt, our results show that it is very possible to get reasonable-looking Python out of the LLM. One somewhat cumbersome detail is their bent to mark the code as specific syntax, the entire LLM response being a Markdown-formatted code block indicating both that the output *is* code, and what language it is in,to the first and last line of the output (Listing 5.2). Several leaked LLM system prompts corroborate this behaviour[4].

[2] https://ollama.com/library/mistral

[3] https://deepmind.google/models/gemini/flash/

[4] See e.g.https://github.com/search?q=repo%3Ajujumilk3%2Fleaked-system-prompts%20markdown&type=code.

```
1 ```python
2
3 [ scenario code ]
4
5 ```
```

Listing 5.2: LLM-generated Python code with Markdown syntax. The bracketed part on line 3 has been added for demonstration purposes, removing the actual code for brevity.

Upon removing these syntactic artefacts, we can go ahead with executing the scenario. As previously mentioned, not all enhanced scenarios immediately work with the Carla simulator. This primarily comes down to (1) hallucination of Python code, and (2) Carla problems, along with the aforementioned (3) markdown-formatted output.

Something worth noting is that the LLM demonstrates a promising ability to explain back to the user *how* it enhanced the scenario, e.g. in the form of bullets in a docstring of the output code (see listing 5.3).

```
1  #!/usr/bin/env python
2
3  # Copyright (c) 2019-2020 Intel Corporation
4  #
5  # This work is licensed under the terms of the MIT license.
6  # For a copy, see <https://opensource.org/licenses/MIT>.
7
8  """
9  Cut in scenario:
10
11 The scenario realizes a driving behavior on the highway.
12 The user-controlled ego vehicle is driving straight and keeping its velocity at a constant
       level.
13 Another car is cutting just in front, coming from left or right lane.
14
15 The ego vehicle may need to brake to avoid a collision.
16
17 Enhanced scenario:
18 - Increased background traffic with varying speeds to create a more crowded environment.
19 - Challenging weather conditions (heavy rain, fog, strong winds) to reduce visibility and grip
       .
20 - Nighttime setting to further decrease visibility.
21 - Randomization of speeds and trigger distances for increased unpredictability.
22 """
23 [...]
```

Listing 5.3: Head of an LLM-enhanced scenario, highlighting how the LLM can add an explenation of how it enhanced the scenario.

5.2.4 Hallucinations in the enhanced scenarios

The LLM typically seems to be on the right track, outlining something that *sounds* like a good approach to satisfying our prompt of decreasing the driveability of the scenario. But in practice, it will often hallucinate methods that don't exist, or use terms and phrasing that are not valid keywords in the Carla specification. This is in line with what was found by e.g. Aiersilan [1, p. 14542] (See AutoSceneGen → p. 20 in Related work).

Non-existing methods

As mentioned, the LLM seems to have the right idea of what it can do to achieve the stated goal. But the way that it goes about obtaining it, does not always work. The enhanced scenario code will often call methods that don't exist. This leads to a runtime exception in the scenario runner when executing the enhanced scenario.

Non-existing arguments

In a similar vein to the non-existing methods, non-existing *arguments* were also shown to appear. The LLM could simply call methods that were already being used, with additional arguments that made semantic sense, but that were not a part of the function definition. This also causes runtime exceptions in the scenario runner.

Illegal property keywords

Another trend we observed was the usage of various keywords that simply don't exist in the Carla repertoire. Where Carla would recognize the word 'snowstorm', the Large Language Model (LLM) could propose using the word 'blizzard'.

Chapter 6

Results

> It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong.
>
> Feynman

This chapter will survey a selection of results from performing various experiments (see Experiment methodology → p. 32). This chapter will *present* the results, and they will then be analysed later in Result analysis → p. 48.

See the listings in the Scenario file diffs → p. 75 appendix for some demonstrations of what the Large Language Model is capable of doing to an scenarios with regard to the syntax of scenarios. Going ahead, we first look at some general aspects that are shared between all our experiments, before narrowing the scope and reviewing a selection of individual scenarios, highlighting the value added by the LLM4DD tool.

Table 6.1 renders a table of the status of executing various scenarios after they have been processed by a Gemini LLM.

There appears to be a bug in Carla version 0.9.15[1] which causes the program to *hard crash* when executing certain scenarios with metric recording enabled. This has been reported to the project GitHub[2], but as of 2025-10-15 it has not been resolved. Testing shows that the same scenarios may be ran without crashing when **not recording**, but this naturally has severe implications for

[1] Which is the version employed for this project.

[2] By several members of the scientific community, see e.g.

- https://github.com/carla-simulator/carla/issues/9170,
- https://github.com/carla-simulator/carla/issues/9152 and
- https://github.com/carla-simulator/carla/issues/9349

Table 6.1: Statistics of simulator execution of LLM-enhanced scenarios, across all prompts

Execution status	Number of Scenarios
Unexpected keyword argument	8
Hard simulator crash	3
Illegal object placement	1
Non-existing import	3
No execution issues	9
Number of enhanced scenarios	24
Number of crashed scenarios	15
Failure ratio	63%

our opportunities of obtaining data from the simulation run. The 'record' functionality of the scenario runner is the crux of measuring the driveability of the scenario.

Note that this is a different kind of problem from those presented in Hallucinations in the enhanced scenarios → p. 35 – *this* problem is relevant for *base* scenarios, provided first party by the Carla simulator team. A major consequence of this is that it hindered what sort of base scenarios we could experiment with. We were naturally unable to experiment with enhancing base scenarios that we were unable to run, as we would have no baseline to measure against, and it is highly improbable that the scenario would magically start working after having gone through an LLM with a prompt aiming at *worsening* its complexity.

6.1 Examples of enhanced scenarios

With all these generalities in mind, let us now narrow the scope and evaluate some tangible scenarios. We will contrast the baseline, original, scenarios, with some that have been enhanced by a LLM, focusing on what changes the LLM proposes and how they affect the driveability of the scenario.

The base scenarios used for these experiments come from the official Carla scenario runner software library[3], but the concept is applicable to scenarios of other repositories as well. Several alternative options are presented in Related work → p. 18. Due to the aforementioned challenges with getting scenarios to run on the Carla simulator, these basic scenarios are used for the purposes of the thesis experiments, serving as a validation of the concept and laying the

[3] https://github.com/carla-simulator/scenario_runner

groundwork for adapting the method to other scenario sources in the future.

As mentioned in section 4.2.2, Carla scenario executions are saved to binary log files. But these files are *huge*, typically being several hundred megabytes depending on the duration of the scenario execution[4]. As such, publishing all our raw files is not feasible. A selection is available in the LLM4DD Github repo.

6.1.1 Base scenario: Follow vehicle

The 'follow vehicle' is the most basic kind of scenario out there. It simply consists of one ego vehicle, and one external vehicle. Our ego is to follow the other vehicle along a straight road in a residential area.

Figure 6.1: A screenshot of the base 'follow' scenario where our ego chases an external actor.

Figure 6.1 →p.39 gives a visual representation of the initial state of the scenario. Due to the low complexity of this scenario, we won't gain any substantial insight into how well our ADS works, if it is to execute the scenario properly. We want to make it more complex in order to provide our ADS with a more challenging environment in which it is more likely to fail.

To this end, we employ an LLM to decrease the driveability.

If we prompt the LLM with a broad allowance of ways of decreasing the

[4]Keep in mind that they record data for all actors in the scenario over time.

driveability such as the prompt demonstrated in listing 6.1, we run into issues with excessive hallucination. The LLM wants to import a class that simply does not exist in the Carla API. See listing B.1 in the Scenario error messages → p. 94 appendix for the complete error message.

```
            "no_explanation": lambda python_carla_scenario_raw: f"""
    1 - Context: You are a tool for decreasing the driveability of scenarios in the driving
     simulator Carla.
    2 - Task: Decrease the driveability of the scenario by enhancing it with more details and
     complexity.
    3 - Input, the Python specification for the scenario: {python_carla_scenario_raw}
    4 - Output: An enhanced version of the scenario with additional details and
    complexity, still in Python carla scenario format. Only ever output the code,
    without any additional text or explanation.
    """,
```

Listing 6.1: The most basic prompt first used in the experiments. This leads to excessive halluciantion.

Due to not being able to run, there is not much to show for here. We need to revise the prompt and discourage such hallucinations in order to obtain meaningful results. In line with the Experiment methodology → p. 32, we iterate on the prompt. We first tell the LLM to *strictly adhere to the Carla API*.

The complete output resulting from this prompt is shown in listing A.1 in the appendix. Keep in mind that LLMs by nature are not deterministic, and as such it is probable that trying to reproduce this output might not be straight-forward.

```
        "no_explanation_strict": lambda python_carla_scenario_raw: f"""
    1 - Context: You are a tool for decreasing the driveability of scenarios in the driving
     simulator Carla.
    2 - Task: Decrease the driveability of the scenario by enhancing it with more details and
     complexity.
    3 - Input, the Python specification for the scenario: {python_carla_scenario_raw}
    4 - Output: An enhanced version of the scenario with additional details and
    complexity, still in Python carla scenario format. Only ever output the code,
    without any additional text or explanation. It is important that you only
    use methods and classes that are part of the official Carla API, and do not
    invent new ones or use non-existent ones.
    """,
```

Listing 6.2: A slightly more advanced prompt instructing the LLM to strictly adhere to the Carla API.

As shown in listing 6.2, we iterate by instructing the LLM to make sure to strictly adhere to the Carla API. This yields a similar problem where the LLM attempts to make an import that does not exist. This diff is presented in listing A.2.

Iterating further, we realize that we must walk before we can run. We therefore instruct the LLM to make as few changes as possible. The intuition being that if it does this and relies on the options that are already present in the file, it is more plausible that we will get a runnable output. This prompt is presented in listing 6.3.

```
        "minimal_changes": lambda python_carla_scenario_raw: f"""
    1 - Context: You are a tool for decreasing the driveability of scenarios in the driving
     simulator Carla.
    2 - Task: Decrease the driveability of the scenario by enhancing it with
    more details and complexity, using only methods that are part of the
    official Carla API, version 0.9.15.
    3 - Input, the Python specification for the scenario: {python_carla_scenario_raw}
    4 - Reasoning: Think step by step about how to make the scenario more complex and less
     driveable, considering possible obstacles, traffic, weather, and other factors using
     only the official Carla API.
    5 - Output: Only output the enhanced scenario code in Python Carla scenario
    format, with no additional text or explanation. Make sure to only use
    methods and concepts that are already present in the input scenario, and
    do not introduce any new methods or concepts. The changes should be as
    minimal as possible while still achieving the goal of decreasing driveability.
    """,
```

Listing 6.3: A prompt instructing the LLM to make as few changes as possible to increase the likelyhood of it working without issues.

This approach works well. We have been able to obtain several working scenarios with decreased driveability with this prompting strategy.

Let us now review some of these enhanced versions of the scenario.

Enhanced scenarios

The enhanced scenarios are simply presented here, and then analysed later in Result analysis → p. 48.

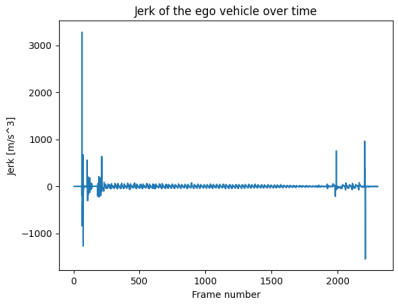
Figure 6.2 → p. 42 gives a visual representation of the initial state of one enhanced scenario.

Another result (Figure 6.3 → p. 42) places a vehicle parked on the edge of the road. This is in line with our prompt, representing a change that is both (1) minimal, and still (2) decreasing driveability.

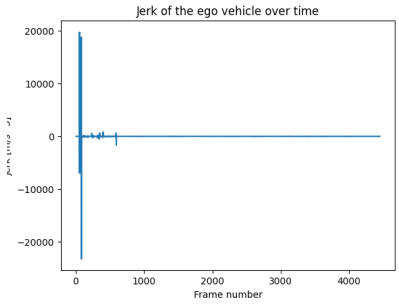
Let us now review the jerk metric for these variations of the scenario (Figure 6.4 → p. 43).

Figure 6.2: A screenshot of a minimally enhanced 'follow' scenario with a truck in the road.

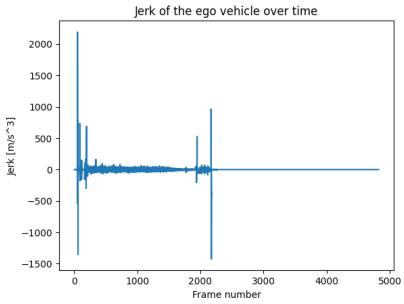
Figure 6.3: A screenshot of another minimally enhanced 'follow' scenario with a van parked on the side of the road.



(a) Jerk in the base scenario, rendered in fig 6.1



(b) Jerk of the ego vehicle in the first minimally enhanced 'follow' scenario, the one from fig 6.2

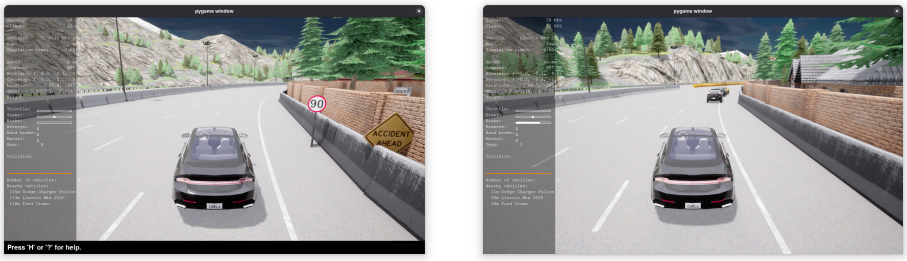


(c) Jerk of the ego vehicle in the 2nd minimally enhanced 'follow' scenario, the one from fig 6.3

Figure 6.4: Jerk of the ego vehicle in the base and enhanced 'follow' scenarios.

6.1.2 Base scenario: Accident

The accident scenario is a bit more complex than the 'follow' scenario, representing a scene on a highway where several cars have piled up in front, and the ego vehicle comes around a corner. Figure 6.5a → p. 44 visually renders the starting point of the scenario. Note the 'accident ahead' sign on the right-hand side of the road. Upon continuing further, a pileup of several vehicles appear in the distance. The base scenario ends with our ego coming to a halt behind the piled up vehicles. Figure 6.5 → p. 44 shows the start point and the progression of the ego continuing around the corner. Figure 6.6 → p. 44 shows how the situation ends – with the ADS ego stopping behind the pileup of other vehicles.



(a) Start of the 'accident' scenario. (b) Ego vehicle underway.

Figure 6.5: Progression of the base 'accident' scenario: start and underway.

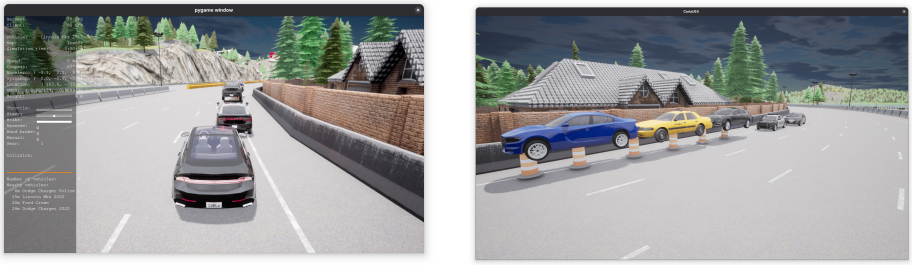
(a) The ego stopped behind the pileup. (b) The same situation from a different angle.

Figure 6.6: The final ego vehicle state in the base 'accident' scenario.

As for the 'follow' scenario, we will enhance the scenario using LLMs and measure the jerk of the ego vehicle between the executions of the scenarios, only presenting the enhanced scenarios and their data points here, and then analysing them later in Result analysis → p. 48.

Enhanced scenarios

The first enhancement is done using the 'minimal changes'-prompt (listing 6.3). It is rendered visually in Figure 6.7 →p.45 and the complete diff of this enhancement is rendered in listing A.3 in Section A.2 →p.88.



(a) The ego stopped behind the pileup, with the vehicles on bollards.

(b) The same situation from a different perspective.

Figure 6.7: The final ego vehicle state in the minimally enhanced 'accident' scenario.

```
"minimal_changes_specific_metric": lambda python_carla_scenario_raw, specific_metric: f"""
  1 - Context: You are a tool for decreasing the driveability of scenarios in the driving
    simulator Carla.
  2 - Task: Decrease the driveability of the scenario by enhancing it with
  more details and complexity, using only methods that are part of the
  official Carla API, version 0.9.15.
  3 - Input, the Python specification for the scenario: {python_carla_scenario_raw}
  4 - Reasoning: Think step by step about how to make the scenario more complex and less
    driveable, considering possible obstacles, traffic, weather, and other factors using
    only the official Carla API.
  5 - Output: Only output the enhanced scenario code in Python Carla scenario
  format, with no additional text or explanation. Make sure to only use
  methods and concepts that are already present in the input scenario, and
  do not introduce any new methods or concepts. The changes should be as
  minimal as possible while still achieving the goal of decreasing
  driveability.
  Focus on making the scenario more difficult with respect to the
  following specific metric: {specific_metric}
  """,
```

Listing 6.4: A prompt instructing the LLM to make as few changes as possible, while maximizing a specific metric.

In order to obtain more interesting results, we again iterate on the prompt and add the requirement of the LLM optimizing for the *jerk* metric. The iterated prompt is rendered in listing 6.4. Note how the prompt is generic and takes the metric as an argument, allowing for other metrics to be used in a similar fashion.

So, what *did* the LLM do? For brevity, the diff is rendered in the appendix Section A.2 →p.88 (listing A.4). From inspecting the diff, it is clear that the LLM has focused on *tweaking* the already existing properties of the scenario, opting to not add additional ontological entities. Not only that, but it has also carried out the modifications for *other* scenarios that are also represented

in the same file. Thus, for our ‘accident‘ scenario, it has only really done the following enhancement, as seen in listing 6.5.

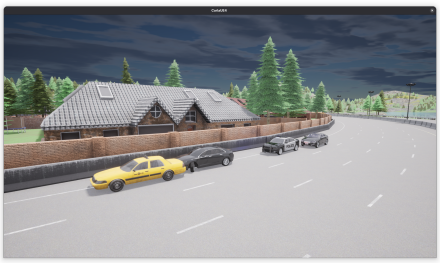
```
@@ -69,9 +69,9 @@ class Accident(BasicScenario):
        self._first_distance = 10
        self._second_distance = 6

-       self._trigger_distance = 50
+       self._trigger_distance = 20  # Decreased to force sharper reactions
        self._end_distance = 50
-       self._wait_duration = 5
+       self._wait_duration = 1  # Decreased to allow less reaction time
        self._offset = 0.6
```

Listing 6.5: The relevant subset of the diff from instructing the LLM to make as few changes as possible, while maximizing a specific metric.

Therefore, we do yet another iteration of the prompt, underlining what scenario the LLM should focus on (listing 6.6).

```
"minimal_changes_shared_file_specific_metric": lambda python_carla_scenario_raw,
 scenario_name, specific_metric: f"""
1 - Context: You are a tool for decreasing the driveability of scenarios in the driving
 simulator Carla.
2 - Task: Decrease the driveability of the scenario by enhancing it with
more details and complexity, using only methods that are part of the
official Carla API, version 0.9.15.
3 - Input, the Python specification for the scenario:
{python_carla_scenario_raw}. Note that there are several scenarios in the file,
but you should only modify the one called {scenario_name}. Don't change any of the
other scenarios.
4 - Reasoning: Think step by step about how to make the scenario more complex and less
 driveable, considering possible obstacles, traffic, weather, and other factors using
 only the official Carla API.
5 - Output: Only output the enhanced scenario code in Python Carla scenario
format, with no additional text or explanation. Make sure to only use
methods and concepts that are already present in the input scenario, and
do not introduce any new methods or concepts. The changes should be as
minimal as possible while still achieving the goal of decreasing
driveability.
Focus on making the scenario more difficult with respect to the
following specific metric: {specific_metric}
""",
```

Listing 6.6: A prompt instructing the LLM to make as few changes as possible, while maximizing a specific metric only in a specified scenario.

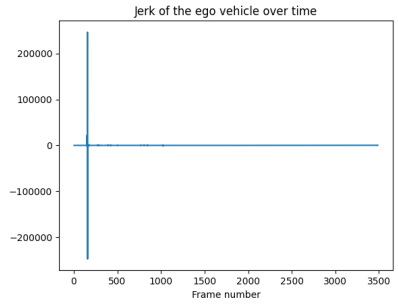


(a) A screenshot from the jerk optimized ‘accident’ scenario with the focused on modifying the correct scenario. See listing A.5.

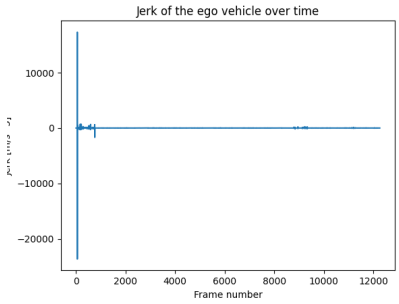
(b) A screenshot from the jerk optimized ‘accident’ scenario with the ego stopped.

Figure 6.8: The final ego vehicle state in two enhanced ‘accident’ scenarios.

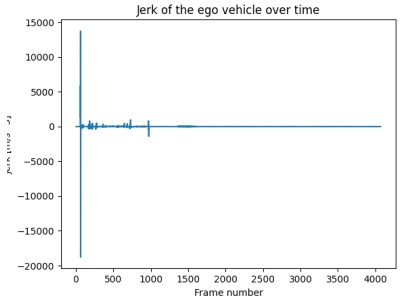
Note how the vehicle behind the taxicab is moving in Figure 6.8a → p. 46.



(a) Jerk in the base scenario, rendered in fig 6.5



(b) Jerk of the ego vehicle in the first minimally enhanced 'accident' scenario, the one from fig 6.7



(c) Jerk of the ego vehicle in the 3rd minimally enhanced 'accident' scenario, the one from fig 6.8a

Figure 6.9: Jerk of the ego vehicle in the base and enhanced 'accident' scenarios.

Chapter 7

Discussion

> I would rather have questions that can't be answered than answers that can't be questioned.
>
> Feynman

This chapter will analyse the Results → p. 37 in light of what they achieve and evaluate the feasibility of the LLM4DD approach in comparison to other methods. Finally, we discuss some broader aspects related to scenario-based testing of ADSs in general.

7.1 Result analysis

The following section will analyse the data presented in the Results → p. 37 chapter, evaluating the degree to which the results satisfy the research questions, and whether the LLM was able to decrease driveability in a meaningful degree.

Overall, the experiments indicate that the initially proposed solution of feeding bare ADS scenarios represented by Python code into LLMs, can yield meaningful results. While the LLM can propose excessive changes that causes issues when performing the simulation when left to its own initiative, the output scenarios are *good* and provide increased insight into the ADS when the LLM is restricted to propose minimal changes to the scenario. Keeping the major components as they were appears to increase the likelihood of the scenario still being able to be be ran without excessive problems.

7.1.1 LLM evaluation

As outlined in Finding a suitable LLM → p. 34, the experimentation was done using a locally hosted 7.2B parameter Mistral model, and a far larger Gemini model – `Gemini 2.5 flash`. The Gemini model obtained significantly better results than the Mistral model, probably due to its size. The mistral model produced incoherent output and was a lot slower[1]. This is why all the following results are done with the Gemini model.

7.1.2 Table of scenario failures

As outlined in Hallucinations in the enhanced scenarios → p. 35, the LLM was found to be prone to hallucinate problematic output in certain scenarios. Table 6.1 gives a quantification of the various statuses encountered while performing LLM experiments for this thesis, outlining (1) to what extent scenarios just worked, and (2) giving statistics of the various kinds of failures. Note that each modified scenario can only ever be counted as 1 sort of failure. So if a scenario fails due to hallucinated imports, it will not also count as a scenario that failed due to having illegally placed objects regardless of however many cases of this it might have attempted. In a similar fashion, we are only able to track the first instance of the error within a given scenario. So if a modified scenario contains 2 hallucinated imports, it will only be counted as 1 in table 6.1.

Something the that the table fails to represent, which is interesting, is the dispersion of errors. Certain scenarios tend to share the same kind of error across several enhancements. This can sway the results and make them appear worse than they are.

7.1.3 Enhanced scenarios: Follow vehicle

As presented in Base scenario: Follow vehicle → p. 39, the LLM does be able to *enhance* this scenario. We show how the most basic prompt (listing 6.1) fails to rein in the LLM which leads to hallucination that causes the scenario to fail to run.

We then iterate on the prompt (listing 6.2) in order to reduce the hallucinations and be able to get a runnable scenario out form the LLM. But even this does not work – the enhanced scenario still fails to run.

This is why we iterate on the prompt again, instructing the LLM to make *minimal changes* (listing 6.3), with the rationale that if the changes to an

[1] Arguably probably due to it running on quite weak hardware. Speed is not a fair comparison.

already working scenario are minimal, there is a greater chance that the output scenario will also work due to how similar they are.

While this reduces our potential gain – it would naturally be more interesting to see significantly modified scenarios – this does work, and the output scenario is runnable.

The first enhanced scenario – Figure 6.2 → p. 42 – entails having there be a truck in the middle of the road. This significantly decreases the driveability of the scenario – it is no longer possible to simple drive straight, the ADS *must* handle the problem of a static vehicle in its path.

Comparing this visually to the base scenario (Figure 6.1 → p. 39) clearly highlights how the ADS has decreased the driveability of the scenario by presenting additional motion planning challenges. This intuition is corroborated by the jerk figures from executing the 2 scenarios (Figure 6.4 → p. 43)

In the case of the base scenario (Figure 6.4a → p. 43), we see that the jerk is mostly stable after the initial acceleration. This is in line with what we expect form the vehicle simply being able to continually drive straight at constant speed. The dip at the end represents the vehicle coming to a halt at a traffic light. This is the end of the scenario.

In the case of the first minimally enhanced scenario (Figure 6.4b → p. 43), we see that the jerk is substantive at first, while converging at 0 after a short while. This is due to the ADS simply not being able to pass the truck in the middle of the road. This demonstrates a significant value in our proposed tool – here lies a scenario that can be manually reviewed and used for evaluating *why* the ADS failed to drive past the parked truck.

On the other hand, the other enhanced 'follow' scenario (Figure 6.3 → p. 42) is not indicated to have decreased driveability in the jerk graph. The jerk in 2nd enhanced follow scenario (Figure 6.4c → p. 43) is stable. This tells us that our ego vehicle was able to pass the parked van without additional issues[2].

And this makes sense. The ADS *has* been presented with an additional challenge (the parked van on the side of the road), making the scene more complex, but it was able to solve the problem without any extra effort. This has given us increased insight into the workings of the ADS, making this count as a result. The jerk need not always *increase* to count as a result – the same jerk is still a meaningful result[3].

[2] In the provided simulation execution, the other vehicle spent a substantial amount of time at the intersection, which is why the jerk appears as 0 for some time before the simulation terminates.

[3] See also Section 8.2.8 → p. 64.

7.1.4 Enhanced scenarios: Accident

The first enhancement is done using the 'minimal changes'-prompt (listing 6.3), and similarly to the 'follow' scenario, it also works well for the 'accident' scenario, resulting in an output scenario that is able to run on the Carla simulator without significant issues.

However, due to the nature of the enhancement the LLM has opted for, no changes are being reflected in the jerk metrics. The LLM tries to make 2 modifications: (1) situating the piled-up vehicles on bollards, and (2) spawning pedestrian actors. The spawning of the pedestrian actors fails, but situating some of the vehicles on cones works. This has, however, minimal effect on the ego vehicle. It completes the same action as it did when the other vehicles were situated on the ground. Furthermore, the concept of having vehicles balancing on bollards in the middle of the road has some seriously dubious realism.

Iterating on the prompt, we still want to optimize for the jerk metric and experiment with including this specific metric *in* the prompt (c.f. listing 6.4). Utilising the jerk prompt yields similar results to the base prompt that doesn't mention any metrics. While one could argue that this *is* **an** enhancement, it is not that interesting. It is too minimal to be of any real value. Note that the prompt *does* instruct the LLM to indeed do be minimal, so this is not strictly speaking a mistake by the LLM.

Note how prompt 6.4 led to the LLM modifying *several* scenarios. This is caused by the file in question housing several scenarios. This is not in itself *wrong* per se, but we naturally wish to isolate the scenario we are working on and not distract the LLM with other scenarios. We therefore iterate on the prompt again and specify the *name* of the relevant scenario in the file, as shown in listing 6.6.

Upon executing this final prompt, we get a better result, with a set of changes that only affect the relevant 'accident' scenario. But even then, the execution of the scenario is in many ways the same. The most striking difference is that another vehicle is now moving. But this has no effect on the ego vehicle, as it stops before interacting with the now moving vehicle. The diff is rendered in listing A.5, and the visual state is shown in Figure 6.8a → p. 46.

Finally, let us review the calculated jerk metrics from these runs of the 'accident' scenario, similarly to the initial 'follow' scenario. As shown in Figure 6.9 → p. 47, there are no significant gains. They all follow the same pattern, which is in line with what we expect from having coming to a halt behind stationary vehicles.

Overall, this indicates this being a feasible way of obtaining less driveable ADS scenarios with minimal costs, in many ways helping with solving our stated problems (Section 1.2 →p.2). LLMs seem capable of performing Conceptual blending →p.13 between the concept of an ADS scenario and the concept of driveability.

7.2 Research question analysis

Let us finish the result analysis by returning to our initial research questions from Chapter 1 →p.1 and review to what extent this work and its Results →p.37 has answered them.

7.2.1 RQ1

Recall that RQ1 says 'Can Large Language Models be used to decrease the driveability of Autonomous driving system simulator scenarios?'.

The answer to this is clearly *yes*. Examples of enhanced scenarios →p.38 demonstrate decreased driveability in several scenarios, across 2 different base scenarios. The difference between the base and its enhanced counterpart is however typically not that significant, as various problems arose when the LLM was allowed to make excessive changes. The LLM oriented aspects →p.59 section of Chapter 8 →p.59 proposes several strategies for remedying this fact.

7.2.2 RQ2

Recall that RQ2 says 'Is it feasible to employ LLMs for obtaining unseen scenarios for ADS testing without human intervention?'.

This is less clearly a success. While certain scenarios did work without human intervention, a greater number needed minor human adjustments (see Table 6.1 →p.38). The LLM oriented aspects →p.59 section of Future work →p.59 also proposes several strategies for remedying this aspect.

7.3 Broader discussion

This section will discuss various aspects related to this LLM-based approach to decrease driveability in ADS simulator scenarios that are of a general nature and not necessarily tied to any specific experiment result.

7.3.1 Scenario modification versus scenario generation

This project concerns itself with taking existing scenarios and *modifying* them. This is in many ways similar to scenario *generation*, but there are also certain key differences. Let us review some of these.

Unknown substrate

When generating a new scenario, the principal factor in determining what it will contain, is a combination of your generation technique, and what sort of training data you will have used for obtaining this technique. As such, you might not always have complete control of what underlying data will be used for the generation of your specific scenario.

With the LLM4DD approach wherein the user provides a base scenario themself, there is 100% certainty of what the base scenario will be. Thus, we obtain significant inherent knowledge of the scenario substrate[4]. Naturally, you may object, the same problem of the 'unknown substrate' will present itself in form of our LLM executing the prompt and inferring what the new – enhanced – scenario will look like. But then, I maintain that we still propose a significant value in the increased awareness *of* the base scenario and its substrate. As we have seen in the Results → p. 37, the changes made by the LLM typically don't alter the underlying ontology of the scene.

This touches on a broader topic concerning the *value* of having this knowledge of the scenario substrate. While this is not the focus of this work, one potential aspect could be a sort of grounding related to the 'sim2real' gap, and the realism of the scene. Let us now delve further into this.

Yao et al. point out that generating new scenarios compared to enhancing existing ones may lead to a distributional shift from the original underlying scenes, which can undermine the validity of using these scenes for testing purposes [54, p. 1], whereas using real-world driving data as a basis for enhancing the scenario would to a greater extent assert that their data distribution is in line with what can be expected in the real world [54, p. 2].

7.3.2 Realism in the enhanced scenario

If your task is to 'obtain bad driveability in a scenario using Large Language Models' in a very general sense, one can imagine all sorts of creative ways this can be achieved. But in our more narrow scope of wishing to highlight practical faults in the ADS, we must add another criterium – realism.

If your scene is bonkers, it will be very easy to get bad driveability. But there is little value and/or practical applicability in these scenarios. Realism in ADS simulator scenarios is a research field in itself with major implications for how scenario-based testing ought to be done.

Wu et al. have looked into evaluating ADS simulator scenario realism using LLMs [53, p. 40]. Note the distinction between this sort of realism in the

[4] Assuming of course that the user *has* this knowledge of their base scenario.

'aligns with our understanding of reality'-approach, and the more technical understanding of realism that posits that the scenario must adhere to the laws of physics, and make sure to not spawn objects in such a way that they intersect (which would also violate the laws of physics) etc. Chang et al. also underline the importance of realism, proposing a scoring function that takes realism into account for evaluating their LLM-generated ADS simulator scenarios [6, pp. 6581–6582][5].

When modifying an existing scenario, the output scenario will in my ways resemble the initial version. This asserts a certain realism grounded in the initial scenario, assuming that it was itself generally realistic. When generating a brand new scenario, it is plausible that we may lose this connection to reality.

Sim 2 real gap

The so called 'sim2real'-gap refers to the distance between a simulation and the real world. We typically wish to minimize this gap in order to increase the applicability of our simulations to real world aspects. If there were a *significant* gap between our simulated ADSs and the real world, there would be minimal value in performing the simulations and the motivation for *using* simulator would falter.

One quite clear example of this, we can find in Section 6.1.2 → p. 44 – the LLM proposes making changes in such a way that several vehicles are situated on top of bollards. This is quite questionable. Not only because it makes no sense that cars could ever balance in such a way on the bollards, but also because – why would the bollards be *under* vehicles in the first place? It would be far more realistic to have the bollards be placed *around* the piled-up vehicles such that the ADS would need to go around them. This would be a more realistic enhancement, and provide a more valuable insight into the performance of the ADS. That said, it would probably require more logic in order to calculate the positions dynamically based on what bollards are located where, in such a way that base LLMs typically don't have the possibility of doing without additional tooling that allows for executing Python code and determining what these position should have been.

7.3.3 LLM aspects

This section discusses various aspects related to employing specifically LLMs for this project. It outlines potential ethical issues and discusses why LLMs may not be perfectly suited for the task at hand.

[5] See also LLM driven scenario generation → p. 21 in Related work → p. 18 for more on this work.

Cost/benefits of using LLMs

LLMs consume significant resources (Section 2.4.4 → p. 15). In order to justify their adaptation for ADS scenario enhancement, we should evaluate whether the pros outweigh the cons. While this remain out of scope for the current thesis and while we demonstrated promising results in Chapter 6 → p. 37, it is important to keep in mind the environmental cost of using the LLMs for this purpose. How good should the results need to be in order to justify using LLMs?

Perhaps future work can look into obtaining similar results using greener strategies.

Context size

By their nature, LLMs have a set context size. If your prompt exceeds the context size of the LLM, it will appear to 'forget' parts of the input. This poses a potential problem for our scenarios. As the scenario complexity increases, so will the length of the prompt. There will inevitably come a point where the LLM can no longer take the entire scenario into account. One potential way of remedying this could perhaps be to only include a subset of the scenario definition file into the prompt, so that it will consume fewer tokens.

Furthermore, certain scenarios sometimes exist in one file. I.e., we use a single file to represent several scenarios. This can potentially cause issues for the LLM in that it will need to know which scenario it is to focus on. This has, however, not caused significant issues for this project.

Alignment

AI alignment is a topic in of itself. How can we know that goals of the AI are the same goals as we have?

Bias

One of the motivations for performing this project, was to circumvent potential bias with the scientists that manually create scenarios. But it's worth noting that this LLM4DD scenario modification approach is still vulnerable to bias based on the LLM training data.

Furthermore, how can we be sure that the enhanced scenarios will reflect the diverse set of possible driving situations? While we indeed know that we *will* get scenarios with decreased driveability, we have no measure of these scenario's relevance for global communities where traffic standards vary.

Perhaps the LLM has some hidden bias preventing it from allowing us to exploit certain aspects of the scenes?

What is being changed in the scenario?

Scenarios centre on the ego vehicle. But what if the LLM changes parts of the scene that don't affect the ego? In these cases, all metrics should be more or less the same[6]. But then there is no point in having done the modification. This has also not been a problem for these experiments.

7.3.4 Scenario formats

This project utilises scenarios in the Python format. Scenarios can also be represented by other formats as we saw in Section 2.1.1 → p. 7, the principal option being OpenSCENARIO. OpenSCENARIO exists both as an XML-based format, and as a Domain specific language (DSL). These 3 options all have their pros and cons. Let us discuss some of these.

It's worth noting that the proposed solution can trivially be adapted to work with other formats.

Python

One major advantage with using Python formats is that most LLMs have significant familiarity with it due to the significant extant Python training data. But this is a double-edged sword – Python has been around for a significant amount of time, and has undergone significant changes. Thus, when prompting the LLM to enhance a Python scenario, there arises several potential issues related to e.g.:

- language syntax, and
- dependency versions.

As outlined in the Results → p. 37, the LLMs are prone to hallucinating in a way that causes issues for executing the enhanced scenarios. But what *is* our evidence that these execution problems stem from 'true' hallucination and not one of these other sources of potential error? Perhaps the LLM did the right thing, only to assume some antiquated generation target (e.g. older version of the Carla simulator with its dependencies, or and older version of the Python programming language).

In an ideal world, we would want an LLM that was fine tuned for this specific purpose to optimize its knowledge of the relevant software versions

[6] Keep in mind that the ADS motion planners are not deterministic, and as such the metrics will typically not be identical.

and decrease the likelihood of it for whatever reason attempting to use bad versions that cause issues.

XML-based OpenSCENARIO

The XML-based format is probably the most traditional. Due to its significantly less widespread usage, LLMs are less likely to have widespread domain knowledge of it. A major advantage with this approach is that the OpenSCENARIO specification is *made* and is less prone to being changed.

Something to note is that the Carla simulator only has limited support for the features of OpenSCENARIO.

DSL-based OpenSCENARIO

Similar to the XML-based format, the scenario DSL is less widely available in training corpora. However, it also maintains the advantage of being more static and less prone to change.

```
import basic.osc

type acceleration is SI(m: 1, s: -2)
unit kphps               of acceleration is SI(m: 1, s: -1, factor: 0.277777778)

scenario top:
    path: Path                      # A path in the map
    path.set_map("Town04")    # specify map to use in this test
    path.path_min_driving_lanes(2)         # Path should have at least two lanes

    ego_vehicle: Model3                # ego car
    npc: Rubicon               # The other car

    event start
    event end
    do serial:
        get_ahead: parallel(duration: 15s):
            ego_vehicle.drive(path) with:
                speed(30kph)
                lane(1, at: start) # left to right: [1..n]
            npc.drive(path) with:
                lane(right_of: ego_vehicle, at: start)
                position(15m, behind: ego_vehicle, at: start)
                position(20m, ahead_of: ego_vehicle, at: end)

        slow: parallel(duration: 10s):
            ego_vehicle.drive(path)
            npc.drive(path) with:
                speed(10kph)

        accelerate: parallel(duration: 13s):
            ego_vehicle.drive(path)
            npc.drive(path) with:
                acceleration(15kphps)
```

Listing 7.1: An example of an OpenSCENARIO DSL scenario.

Listing 7.1 presents an example of an OpenSCENARIO DSL scenario, providing additional context for understanding the concept of the format. The example is copied from the Carla scenario runner examples.

7.3.5 When is enough – when is the ADS *safe*?

An interesting perspective regarding this scenario based testing of ADSs is this: How can we ever know that it has been sufficiently tested and is ready for being deployed on public roads? When investigation this question, several sub-problems arise.

As mentioned in the Problem description $^{\rightarrow p.2}$, there are infinitely many edge cases that are possible for ADS execution. Even if we propose one billion test cases, and our ADS passes these, how can we know that it will also pass the hypothetical 'one billion and one-th'?

This is one of the principal problems precented in the motivation, and this project has perhaps contributed *somewhat* to the problem by presenting a way by which we can increase our confidence in the ADS, but the initial problem presented in the Introduction, as outlined by Leahy et al. remain.

Furthermore, even if it does pass all these tests, how can we be sure that it acts in an ethical manner? There are several possible courses of action that it can take in various situations and we need to be confident in the ADS making the choice we deem the correct one on several levels – not only whatever the unexplainable ADS motion planner places the highest. The *trolley problem* of ethical philosophy [46] comes to mind – we should be prepared for our ADS to have to handle such cases in operation.

Chapter 8

Future work

> Whenever a theory appears to you as the only possible one, take this as a sign that you have neither understood the theory nor the problem which it was intended to solve.
>
> Popper

This penultimate chapter proposes directions for further research aimed at solving the same problem in a similar manner, based on (1) the experiences gained during this work, and (2) approaches that there was not time to go through with.

8.1 LLM oriented aspects

LLMs are naturally central for a work like this. The following section reviews various LLM-related aspects that may stand to serve the goal of employing LLMs for decreasing driveability in ADS simulator scenarios. The proposed strategies stem from the experiences gained while performing the experiments and analysing the results, and they relate both (1) to the actual LLMs, (2) how the LLM can be exploited with prompts, and (3) recent techniques that aim to reduce problems with LLM hallucinations.

8.1.1 Prompting strategies

For a work like this, the importance of the *prompts* cannot be understated. Further work may stand to benefit from exploring different prompting strategies. See for example [18]. Prompting techniques for persuading LLMs to perform actions they may find objectionable are presented by Meincke et al. and shown to bear fruitful results [36, p. 1]. This might stand to remedy potential alignment and bias issues (see section 7.3.3).

8.1.2 Experimenting with other models

LLMs typically have different properties. Further work could evaluate different models on the same experiments. Both for models that already exist today, and models that will exist in the future.

8.1.3 Fine-tuned model

This work has exclusively used generalist LLMs. Given the necessary resources, one could fine-tune a model specifically for this purpose, using scenario definitions as training data.

8.1.4 Temperature configurations

As a consequence of how LLMs work internally, they have a *temperature* that affects their output. For the same reasons that further work can explore the applicability of other LLMs, testing the same LLMs with different temperature configurations may also yield an interesting insight into their applicability for decreasing driveability in ADS simulator scenarios.

Keep in mind how hallucination was a significant factor when undertaking the experimentation (see section 5.2.4). In order to decrease the hallucinations, one would assume a lower temperature LLM to be more relevant.

8.1.5 Retrieval-augmented generation (RAG)

Retrieval augmented generation (RAG) is a novel technique that inserts pieces of relevant external knowledge into an LLM's context [56, pp. 88–89]. It would be interesting to employ such a strategy for this work. By adding existing scenarios to the LLM's context, it would be able to leverage these as a source of legal ways of modifying scenarios in a way that we would know to be both (1) legal, and (2) supported by the simulator. Perhaps this could provide better results with less hallucinations?

8.1.6 Model context protocol (MCP)

Recently, a technique called Model context protocol (MCP) has been proposed[1], that allows for defining a specific interface for LLMs to interact with. Further work could construct such an interface for modifying scenarios, clearly demarcating what the LLM can and cannot do.

In a somewhat similar fashion to RAG, albeit with more strict limitations imposed on the LLM, this could allow us to be more confident in knowing that it would perform legal actions.

[1] See e.g. https://www.anthropic.com/news/model-context-protocol

8.1.7 Tool manipulation

Tool manipulation is a technique for granting LLMs access to *tools* [56, p. 62], e.g. the ability to call certain functions. Somewhat similar to MCP, but less general – not standardized. For the same reasons that MCP can be used, tool manipulation can be used. Define functions to add objects at locations that the script determines to be legal with regard to the laws of physics and the placement of other objects, and the LLM will then be able to utilise this function as a black-box utility, without having to concern itself with the ontological issues of object spawning.

8.2 Implementation oriented aspects

This section proposes methods for enhancing the existing implementation (see chapter 4) with additional features or better techniques, motivating why the additional features would be useful.

8.2.1 Static analysis of the enhanced scenario

As of now, the principal way of determining whether a modified scenario is good and runnable, is by simply attempting to run it and see if it works. This is not ideal as it wastes lot of time. Even more time is wasted if the scenario causes the Carla simulator to crash[2].

By employing techniques from the realm of programming language compilers, we could perform a static analysis of the output scenario and evaluate whether it adheres to the specification. This would allow us to filter out bad apples before attempting to run them, circumventing the need for wasting time. For example, we *know* that we should be able to *parse* the scenario. If parsing fails, there is no need of sending the scenario to the simulation running pipeline.

An advanced version of this could also look into the proposed spawn locations of items in the scenario[3], making sure that they don't intersect and adhere to the relevant laws of physics. Perhaps this could also serve to yield some insight into the realism of the scenario.

8.2.2 GUI visualisation

Scenarios in ADS simulators are inherently *visual*. This can be leveraged to create a tool for rendering visual representations of the scenarios in a Graphical user interface (GUI). The GUI tool could show side-by-side the

[2]Getting it back online is a quite involved and slow-to-complete process.
[3]That may overlap...

base scenario and its enhanced counterpart, highlighting what has been modified. This will be more user friendly, and it will more clearly underline the motivation behind *why* the enhanced scenario will have worsened driveability compared to the base scenario. Perhaps one could even *execute* the scenarios side-by-side, showing live metrics to the user.

8.2.3 Other datasets

For this thesis, a selection of quite basic scenarios from the Carla scenario runner example scenario collection was used. However, the concept should work for *any* Carla-compatible ADS scenarios. Several such options are presented in Related work → p. 18.

8.2.4 More diverse scenarios

The base scenarios used for this work are generally western in nature. Global ADS adaptation requires universal testing on a broad range of scenarios with various traffic conditions and cultural aspects. Future work may stand to benefit from employ this LLM-based approach in order to obtain a greater number of scenarios from less widely available scenarios from regions where relevant data is less prolific and existing scenarios are scarce.

8.2.5 Efficiency

Due to the somewhat 'proof of concept' nature of the current LLM4DD implementation, performance and efficiency has not been afforded significant attention. There are however some quite low-hanging fruits that stand to be exploited. This subsection outlines some of these.

Concurrency of scenario processing

ADS scenarios are independent of each other. This means that our problem is *embarrassingly parallelizable*[4] and we can trivially process several test cases in parallel. Both performing the LLM enhancement, and executing them on Carla, can be done regardless of all the other scenarios.

Python-specific enhancements

The Python-based LLM4DD implementation can be optimized using Just-In-Time (JIT) compilers such as Numba [24], which can speed up our execution times. Libraries such as Joblib provide Python with plug-and-play meomization, which will allow us to re-use values that have already been computed, saving time and energy.

[4] https://en.wikipedia.org/wiki/Embarrassingly_parallel

8.2.6 Domain specific file format

These experiments have been performed using only default file-formats, relying on manual labour to track (1) what scenario x' is the modified version of what scenario x, (2) what prompt was used, (3) what[5] manual changes was needed for executing, etc.

The scientific community could stand to benefit from developing a domain-specific file format for this specific purpose. E.g. a JSON-based file format that contained a copy of the (1) base scenario, the (2) enhanced scenario, (3) what prompt was used, (4) the diff, (5) timestamp, etc. Perhaps even what metrics were achieved while executing the 2 forms of the scenario. It would however not be optimal to track several copies of the same base scenario – there would be a lot of duplicates, and they would all have virtually the same metrics.

This could all be achieved by wrapping a JSON file. Perhaps we could zip or encode it to make it less prone to manual modification[6].

8.2.7 More stable Carla setup

As outlined in Chapter 6 → p. 37, the Carla simulator just does not work with certain scenarios. Future work could stand to benefit from having the simulator be more stable. That would allow from experimenting with a more diverse array of base scenarios, which would stand expand the range of potential scenarios we would be able to challenge the ADS with.

Jia et al. posit that the Nvidia GPU driver version of the host machine is relevant for the Carla stability, finding that version 470 supposedly is optimal and that version 550[7] 'has lots of bugs'[8]. The experiments of this project have been done using version 580.

A possible alternate strategy could be to undertake a similar work within the CommonRoad framework, bypassing Carla altogether[9].

[5] If any ...

[6] Which would invalidate its content and potentially serve to spoof the results.

[7] And presumably following versions.

[8] See https://github.com/Thinklab-SJTU/Bench2Drive/tree/main?tab=readme-ov-file#deal-with-carla, from the repo of their paper [23].

[9] See e.g. https://commonroad.in.tum.de/tools/drivability-checker.

8.2.8 More scientific way of evaluating a result

As of now, the process of evaluating an experiment as failed or succeeded (see Metrics → p. 33), is not that scientific. It works sufficiently for our purposes, but there is a lot to be gained by formalizing the process.

Further work could look into defining mathematical properties for what qualifies as a 'meaningfully decreased driveability'. Jahangirova, Stocco, and Tonella propose several evaluation criteria [21, p. 196]. One typical example could be to look at whether the jerk exceeded a certain threshold during scenario execution – but even then, we need to define what this threshold should be. Several heuristics may be applicable. And it really isn't the case that 'more jerk' indicates the presence of a result, as maintaining *the same* jerk could be equally indicative of a result. This has to be tailored to the specific scene, in light of what the LLM has changed. In certain cases it would be good to see an increased jerk, and in other cases we would want it to remain the same. The reason for this is that if the LLM changes something on the other side of the map that doesn't affect the ADS ego (c.f. Section 7.3.3 → p. 56), we should not count it as a failure for the ADS if it fails to obtain more jerk – it should be counted as a failure for the LLM.

This is a *hard* question and has been one of the major recurring challenges while working on this project.

Chapter 9

Conclusion

> This inductively justifies the conclusion that induction cannot justify any conclusions.
>
> David Deutsch

In this master's thesis, we propose a tool – LLM4DD – for using LLMs to decrease the driveability of ADS scenarios in order to increase confidence in the ADS and expose underlying weaknesses in the system. We do a literature review with both a theoretical and applied focus. We show that this work is in line with what is to be expected from comparison with other related works in the field, validating that the LLM4DD concept works quite well when focusing on making less significant changes to the original scenario with a focus on the jerk metric. When allowing the LLM to make excessive changes to the scenario, the results indicate that problems related to hallucination and simulator crashes arise under our current LLM- and prompting strategies.

The LLM4DD tool is a modularized pipeline tool, consisting of the components `Odin`, `Thor`, and `Loki`, each component being respectively responsible for handling (1) LLM integration, (2) ADS integration, and (3) user-based orchestration.

We used the LLMs `Mistral 7.2B` and `gemini-2.5-flash` and various original prompts. The scenarios we used for evaluating the tool came from the example set provided with the Carla scenario runner. We primarily focused on the scenarios `Accident`, `CutIn`, `NoSignalJunctionCrossing` and `FollowLeadingVehicle`. The results obtained indicate a 63% error rate of running a scenario after it has been enhanced. We classified these errors into 4 categories based on what caused the execution to fail.

We propose several strategies for improving this initial version of the tool in Future work → p. 59.

In conclusion, this thesis shows that LLMs can indeed be used to decrease the driveability of ADS scenarios. The LLM-based approach presented in this thesis and evaluated by implementing the LLM4DD tool, appears as a promising contribution to furthering the state of the art of ADS verification research.

References

[1] Aizierjiang Aiersilan. "Generating Traffic Scenarios via In-Context Learning to Learn Better Motion Planner". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 14. 2025, pp. 14539–14547. DOI: 10.1609/aaai.v39i14.33593 (cited on pages 6, 20, 35).

[2] Matthias Althoff, Markus Koschi, and Stefanie Manzinger. "CommonRoad: Composable benchmarks for motion planning on roads". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 719–726. DOI: 10.1109/IVS.2017.7995802 (cited on page 7).

[3] Philip W Anderson. "More Is Different: Broken symmetry and the nature of the hierarchical structure of science." In: *Science* 177.4047 (1972), pp. 393–396 (cited on page 12).

[4] Emily M. Bender et al. "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 610–623. ISBN: 9781450383097. DOI: 10.1145/3442188.3445922. URL: https://doi.org/10.1145/3442188.3445922 (cited on page 13).

[5] Roger Brown. "Reference in memorial tribute to Eric Lenneberg". In: *Cognition* 4.2 (1976), pp. 125–153. ISSN: 0010-0277. DOI: https://doi.org/10.1016/0010-0277(76)90001-9. URL: https://www.sciencedirect.com/science/article/pii/0010027776900019 (cited on page 13).

[6] Cheng Chang et al. "LLMScenario: Large Language Model Driven Scenario Generation". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 54.11 (2024), pp. 6581–6594. DOI: 10.1109/TSMC.2024.3392930 (cited on pages 21, 54).

[7] He Chen et al. "Generating Autonomous Driving Test Scenarios based on OpenSCENARIO". In: *2022 9th International Conference on Dependable Systems and Their Applications (DSA)*. 2022, pp. 650–658. DOI: 10.1109/DSA56465.2022.00093 (cited on page 7).

[8] Zhengxian Chen et al. "Predicting driving comfort in autonomous vehicles using road information and multi-head attention models". In: *Nature Communications* 16.1 (Mar. 2025), p. 2709. ISSN: 2041-1723. DOI: 10.1038/s41467-025-57845-z. URL: https://doi.org/10.1038/s41467-025-57845-z (cited on pages 22, 23).

[9] Can Cui et al. "Drive As You Speak: Enabling Human-Like Interaction With Large Language Models in Autonomous Vehicles". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*. Jan. 2024, pp. 902–909 (cited on page 16).

[10] Can Cui et al. *Large Language Models for Autonomous Driving (LLM4AD): Concept, Benchmark, Experiments, and Challenges*. 2025. arXiv: 2410.15281 [cs.RO]. URL: https://arxiv.org/abs/2410.15281 (cited on pages 2, 16, 18).

[11] Jean-Emmanuel Deschaud. *KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator*. 2021. arXiv: 2109.00892 [cs.CV]. URL: https://arxiv.org/abs/2109.00892 (cited on page 6).

[12] Yi Ding and Tianyao Shi. "Sustainable LLM Serving: Environmental Implications, Challenges, and Opportunities : Invited Paper". In: *2024 IEEE 15th International Green and Sustainable Computing Conference (IGSC)*. 2024, pp. 37–38. DOI: 10.1109/IGSC64514.2024.00016 (cited on page 15).

[13] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16 (cited on page 6).

[14] Epic Games. *Unreal Engine*. Version 4.22.1. Apr. 25, 2019. URL: https://www.unrealengine.com (cited on page 6).

[15] Gilles Fauconnier and Mark Turner. "Conceptual Blending, Form and Meaning". In: *Recherches en Communication; No 19: Sémiotique cognitive — Cognitive Semiotics; 57-86* 19 (Mar. 2003). DOI: 10.14428/rec.v19i19.48413 (cited on page 13).

[16] Fred Feng et al. "Can vehicle longitudinal jerk be used to identify aggressive drivers? An examination using naturalistic driving data". In: *Accident Analysis & Prevention* 104 (2017), pp. 125–136. ISSN: 0001-4575. DOI: https://doi.org/10.1016/j.aap.2017.04.012. URL: https://www.sciencedirect.com/science/article/pii/S0001457517301409 (cited on page 11).

[17] Erwin de Gelder, Maren Buermann, and Olaf Op Den Camp. "Coverage Metrics for a Scenario Database for the Scenario-Based Assessment of Automated Driving Systems". In: *2024 IEEE International Automated Vehicle Validation Conference (IAVVC)*. IEEE. 2024, pp. 1–8 (cited on page 7).

[18] Louie Giray. "Prompt Engineering with ChatGPT: A Guide for Academic Writers". In: *Annals of Biomedical Engineering* 51.12 (Dec. 2023), pp. 2629–2633. ISSN: 1573-9686. DOI: 10.1007/s10439-023-03272-4. URL: https://doi.org/10.1007/s10439-023-03272-4 (cited on page 59).

[19] Junyao Guo, Unmesh Kurup, and Mohak Shah. "Is it safe to drive? An overview of factors, metrics, and datasets for driveability assessment in autonomous driving". In: *IEEE Transactions on Intelligent Transportation Systems* 21.8 (2019), pp. 3135–3151 (cited on pages 7–11).

[20] WuLing Huang et al. "Autonomous vehicles testing methods review". In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2016, pp. 163–168 (cited on pages 9, 10).

[21] Gunel Jahangirova, Andrea Stocco, and Paolo Tonella. "Quality Metrics and Oracles for Autonomous Vehicles Testing". In: *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. 2021, pp. 194–204. DOI: 10.1109/ICST49551.2021.00030 (cited on page 64).

[22] Nidhal Jegham et al. *How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference*. 2025. arXiv: 2505.09598 [cs.CY]. URL: https://arxiv.org/abs/2505.09598 (cited on page 15).

[23] Xiaosong Jia et al. "Bench2Drive: Towards Multi-Ability Benchmarking of Closed-Loop End-To-End Autonomous Driving". In: *arXiv preprint arXiv:2406.03877* (2024) (cited on page 63).

[24] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba: A llvm-based python jit compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, pp. 1–6 (cited on page 62).

[25] Kevin Leahy et al. *Grand Challenges in the Verification of Autonomous Systems*. 2024. arXiv: 2411.14155 [cs.RO]. URL: https://arxiv.org/abs/2411.14155 (cited on pages 2, 58).

[26] Krzysztof Lebioda et al. *Are requirements really all you need? A case study of LLM-driven configuration code generation for automotive simulations*. 2025. arXiv: 2505.13263 [cs.SE]. URL: https://arxiv.org/abs/2505.13263 (cited on page 20).

[27] Baolin Li et al. "Sprout: Green Generative AI with Carbon-Efficient LLM Inference". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 21799–21813. DOI: 10.

18653/v1/2024.emnlp-main.1215. URL: https://aclanthology.org/2024.emnlp-main.1215/ (cited on page 15).

[28] Pengfei Li et al. *Making AI Less "Thirsty": Uncovering and Addressing the Secret Water Footprint of AI Models*. 2025. arXiv: 2304.03271 [cs.LG]. URL: https://arxiv.org/abs/2304.03271 (cited on page 15).

[29] Wei Li and Yuyue Liu. "Adoption of autonomous driving technology: An integration of innovation diffusion theory and motivation theory". In: *Human Systems Management* 0.0 (2025), p. 01672533251365121. DOI: 10.1177/01672533251365121. eprint: https://doi.org/10.1177/01672533251365121. URL: https://doi.org/10.1177/01672533251365121 (cited on page 1).

[30] Xuan Li et al. "ChatGPT-Based Scenario Engineer: A New Framework on Scenario Generation for Trajectory Prediction". In: *IEEE Transactions on Intelligent Vehicles* 9.3 (2024), pp. 4422–4431. DOI: 10.1109/TIV.2024.3363232 (cited on page 21).

[31] Yuanfei Lin, Michael Ratzel, and Matthias Althoff. "Automatic Traffic Scenario Conversion from OpenSCENARIO to CommonRoad". In: *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. 2023, pp. 4941–4946. DOI: 10.1109/ITSC57777.2023.10422422 (cited on page 7).

[32] Pengfei Liu et al. "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing". In: *ACM Comput. Surv.* 55.9 (Jan. 2023). ISSN: 0360-0300. DOI: 10.1145/3560815. URL: https://doi.org/10.1145/3560815 (cited on pages 13, 14).

[33] Chengjie Lu, Tao Yue, and Shaukat Ali. "DeepScenario: An Open Driving Scenario Dataset for Autonomous Driving System Testing". In: *IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)* (2023), pp. 52–56 (cited on pages 1, 5, 6, 9, 19).

[34] Chengjie Lu et al. "Learning Configurations of Operating Environment of Autonomous Vehicles to Maximize their Collisions". In: *IEEE Transactions on Software Engineering* 49.1 (2023), pp. 384–402. DOI: 10.1109/TSE.2022.3150788 (cited on page 19).

[35] Y.K. Malaiya et al. "The relationship between test coverage and reliability". In: *Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering*. 1994, pp. 186–195. DOI: 10.1109/ISSRE.1994.341373 (cited on page 9).

[36] Lennart Meincke et al. "Call Me A Jerk: Persuading AI to Comply with Objectionable Requests". In: *The Wharton School Research Paper* (July 2025). DOI: 10.2139/ssrn.5357179. URL: https://ssrn.com/abstract=5357179 (cited on page 59).

[37] Demin Nalic et al. "Scenario Based Testing of Automated Driving Systems: A Literature Survey". English. In: FISITA Web Congress 2020 ; Conference date: 24-11-2020 Through 24-11-2020. Nov. 2020, pp. 1–10. URL: https://go.fisita.com/fisita2020 (cited on page 1).

[38] Youngseok Park, Ji Hyun Yang, and Sejoon Lim. "Development of Complexity Index and Predictions of Accident Risks for Mixed Autonomous Driving Levels". In: *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2018, pp. 1181–1188. DOI: 10.1109/SMC.2018.00208 (cited on page 10).

[39] Nenad Petrovic et al. "LLM-Driven Testing for Autonomous Driving Scenarios". In: *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*. 2024, pp. 173–178. DOI: 10.1109/FLLM63129.2024.10852505 (cited on page 20).

[40] Andreas Riener. "The Driver as the Weak Point in Interaction". In: *Sensor-Actuator Supported Implicit Interaction in Driver Assistance Systems*. Wiesbaden: Vieweg+Teubner, 2010, pp. 67–70. ISBN: 978-3-8348-9777-0. DOI: 10.1007/978-3-8348-9777-0_7. URL: https://doi.org/10.1007/978-3-8348-9777-0_7 (cited on page 1).

[41] Guodong Rong et al. "LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving". In: *arXiv preprint arXiv:2005.03778* (2020) (cited on page 6).

[42] Max Theo Schmidt, Ulrich Hofmann, and M. Essayed Bouzouraa. "A novel goal oriented concept for situation representation for ADAS and automated driving". In: *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. 2014, pp. 886–893. DOI: 10.1109/ITSC.2014.6957801 (cited on page 7).

[43] Shital Shah et al. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles". In: *Field and Service Robotics*. 2017. eprint: arXiv:1705.05065. URL: https://arxiv.org/abs/1705.05065 (cited on page 6).

[44] Shuhan Tan et al. *Language Conditioned Traffic Generation*. 2023. arXiv: 2307.07947 [cs.CV]. URL: https://arxiv.org/abs/2307.07947 (cited on pages 20, 21).

[45] John Thøgersen et al. "Why do people continue driving conventional cars despite climate change? Social-psychological and institutional insights from a survey of Norwegian commuters". In: *Energy Research & Social Science* 79 (2021), p. 102168. ISSN: 2214-6296. DOI: https://doi.org/10.1016/j.erss.2021.102168. URL: https://www.sciencedirect.com/science/article/pii/S2214629621002619 (cited on page 1).

[46] Judith Jarvis Thomson. "The Trolley Problem". In: *The Yale Law Journal* 94.6 (1985), pp. 1395–1415. ISSN: 00440094. URL: http://www.jstor.org/stable/796133 (visited on 11/11/2025) (cited on page 58).

[47] Bill Tomlinson et al. "The carbon emissions of writing and illustrating are lower for AI than for humans". In: *Scientific Reports* 14.1 (2024), p. 3732 (cited on page 15).

[48] Simon Ulbrich et al. "Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving". In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 2015, pp. 982–988. DOI: 10.1109/ITSC.2015.164 (cited on pages 6, 7).

[49] Ashish Vaswani et al. "Attention is all you need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964 (cited on page 11).

[50] Jian Wang et al. "A Survey of Vehicle to Everything (V2X) Testing". In: *Sensors* 19.2 (2019). ISSN: 1424-8220. DOI: 10.3390/s19020334. URL: https://www.mdpi.com/1424-8220/19/2/334 (cited on page 10).

[51] Jason Wei et al. *Emergent Abilities of Large Language Models*. 2022. arXiv: 2206.07682 [cs.CL]. URL: https://arxiv.org/abs/2206.07682 (cited on page 12).

[52] Klaus Peter Wershofen and Volker Graefe. "Situationserkennung als Grundlage der Verhaltenssteuerung eines mobilen Roboters". In: *Autonome Mobile Systeme 1996*. Ed. by Günther Schmidt and Franz Freyberger. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 170–179. ISBN: 978-3-642-80324-6 (cited on page 7).

[53] Jiahui Wu et al. "Reality Bites: Assessing the Realism of Driving Scenarios with Large Language Models". In: *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*. FORGE '24. Lisbon, Portugal: Association for Computing Machinery, 2024, pp. 40–51. ISBN: 9798400706097. DOI: 10.1145/3650105.3652296. URL: https://doi.org/10.1145/3650105.3652296 (cited on page 53).

[54] Yu Yao et al. *AGENTS-LLM: Augmentative GENeration of Challenging Traffic Scenarios with an Agentic LLM Framework*. 2025. arXiv: 2507.13729 [cs.RO]. URL: https://arxiv.org/abs/2507.13729 (cited on pages 2, 3, 24, 53).

[55] Tao Yue, Shaukat Ali, and Man Zhang. "RTCM: a natural language based, automated, and practical test case generation framework". In: *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ISSTA 2015. Baltimore, MD, USA: Association for Computing Machinery, 2015, pp. 397–408. ISBN: 9781450336208. DOI:

10.1145/2771783.2771799. URL: https://doi.org/10.1145/2771783.2771799 (cited on page 19).

[56] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2025. arXiv: 2303.18223 [cs.CL]. URL: https://arxiv.org/abs/2303.18223 (cited on pages 11–14, 60, 61).

[57] Yongqi Zhao et al. *A Survey on the Application of Large Language Models in Scenario-Based Testing of Automated Driving Systems*. 2025. arXiv: 2505.16587 [cs.SE]. URL: https://arxiv.org/abs/2505.16587 (cited on pages 1, 2, 17, 18).

[58] Yongqi Zhao et al. "Chat2Scenario: Scenario Extraction From Dataset Through Utilization of Large Language Model". In: *2024 IEEE Intelligent Vehicles Symposium (IV)*. 2024, pp. 559–566. DOI: 10.1109/IV55156.2024.10588843 (cited on pages 21, 22).

Appendix

Appendix A

Scenario file diffs

The diffs represent the *diff*erence between two files, highlighting what has changed. In this context – the red indicates something that was changed from the original scenario, and the green indicates something that was added by the LLM. The lines in black are unchanged.

A.1 Follow vehicle

See Section 6.1.1 → p.39 for an overview and Section 7.1.3 → p.49 for analysis of this output.

A.1.1 Initial enhancement

```
diff --git a/srunner/scenarios/follow_leading_vehicle.py b/srunner/scenarios/follow_leading_vehicle.py
index 5f60061..d09ccfb 100644
--- a/srunner/scenarios/follow_leading_vehicle.py
+++ b/srunner/scenarios/follow_leading_vehicle.py
@@ -27,265 +27,237 @@ from srunner.scenariomanager.scenarioatomics.atomic_behaviors import (ActorTrans
                                                                      ActorDestroy,
                                                                      KeepVelocity,
                                                                      StopVehicle,
-                                                                     WaypointFollower)
+                                                                     WaypointFollower,
+                                                                     PedestrianWalk,
+                                                                     SetTrafficLightState,
+                                                                     ChangeAutoPilot)
 from srunner.scenariomanager.scenarioatomics.atomic_criteria import CollisionTest
 from srunner.scenariomanager.scenarioatomics.atomic_trigger_conditions import (InTriggerDistanceToVehicle,

     InTriggerDistanceToNextIntersection,
                                                                                DriveDistance,
-                                                                               StandStill)
+                                                                               StandStill,
+
     TriggerDistanceToLocation)
 from srunner.scenariomanager.timer import TimeOut
 from srunner.scenarios.basic_scenario import BasicScenario
-from srunner.tools.scenario_helper import get_waypoint_in_distance
+from srunner.tools.scenario_helper import get_waypoint_in_distance, get_location_in_distance_from_wp, get_crossing_point, get_next_traffic_light


```

```
27  class FollowLeadingVehicle(BasicScenario):
28  -
29      """
30  -    This class holds everything required for a simple "Follow a leading vehicle"
31  -    scenario involving two vehicles.  (Traffic Scenario 2)
32  -
33  -    This is a single ego vehicle scenario
34  +    This class holds a complex "Follow a leading vehicle" scenario, enhancing
35  +    the basic scenario with:
36  +    - A parked vehicle on the side of the road, potentially narrowing the path.
37  +    - A pedestrian crossing the road in front of the leading vehicle.
38  +    - A dynamic vehicle that is on autopilot, adding to general traffic.
39  +    - A traffic light that turns red as the leading vehicle approaches, forcing
40  +      a more abrupt stop.
41  +
42  +    This is a single ego vehicle scenario.
43      """
44
45  -    timeout = 120            # Timeout of scenario in seconds
46  +    timeout = 180  # Increased timeout for added complexity
47
48      def __init__(self, world, ego_vehicles, config, randomize=False, debug_mode=False,
        criteria_enable=True,
49  -                 timeout=60):
50  +                 timeout=180):
51          """
52  -        Setup all relevant parameters and create scenario
53  +        Setup all relevant parameters and create scenario.
54
55  -        If randomize is True, the scenario parameters are randomized
56  +        If randomize is True, the scenario parameters are randomized.
57          """
58  -
59          self._map = CarlaDataProvider.get_map()
60  -        self._first_vehicle_location = 25
61  -        self._first_vehicle_speed = 10
62  +        self._leading_vehicle_spawn_distance = 25  # Distance from ego trigger point
63  +        self._leading_vehicle_speed = 10  # m/s
64          self._reference_waypoint = self._map.get_waypoint(config.trigger_points[0].location)
65  -        self._other_actor_max_brake = 1.0
66  -        self._other_actor_stop_in_front_intersection = 20
67  -        self._other_actor_transform = None
68  -        # Timeout of scenario in seconds
69  +        self._leading_actor_max_brake = 1.0
70  +        self._leading_actor_stop_in_front_intersection = 10 # Stop closer to intersection
71  +
72  +        # New actor parameters (distances relative to ego trigger point)
73  +        self._parked_vehicle_distance = self._leading_vehicle_spawn_distance + 30
74  +        self._pedestrian_crossing_distance = self._leading_vehicle_spawn_distance + 45
75  +        self._pedestrian_speed = 1.0 # m/s
76  +        self._dynamic_vehicle_distance = self._leading_vehicle_spawn_distance + 15
77  +        self._dynamic_vehicle_speed = 12 # m/s
78  +
79  +        self._parked_vehicle_offset = 3.0 # Meters to the side of the lane
80  +        self._traffic_light_state_change_trigger_distance = self.
        _leading_actor_stop_in_front_intersection + 5 # When leading vehicle is this far from
        intersection, turn light red
81  +
82  +        # Internal actor references
83  +        self._leading_vehicle = None
84  +        self._parked_vehicle = None
85  +        self._pedestrian = None
86  +        self._dynamic_vehicle = None
87  +        self._traffic_light = None
88  +        self._next_intersection_waypoint = None
89  +
90          self.timeout = timeout
91
92  -        super(FollowLeadingVehicle, self).__init__("FollowVehicle",
93  -                                                   ego_vehicles,
94  -                                                   config,
95  -                                                   world,
96  -                                                   debug_mode,
97  -                                                   criteria_enable=criteria_enable)
98  +        super(ComplexFollowLeadingVehicle, self).__init__("ComplexFollowLeadingVehicle",
99  +                                                          ego_vehicles,
100 +                                                          config,
101 +                                                          world,
102 +                                                          debug_mode,
103 +                                                          criteria_enable=criteria_enable)
104
105         if randomize:
106 -           self._ego_other_distance_start = random.randint(4, 8)
107 +           self._leading_vehicle_speed = random.uniform(8, 12)
```

```
+            self._pedestrian_speed = random.uniform(0.8, 1.5)
+            self._dynamic_vehicle_speed = random.uniform(10, 15)
+            self._leading_vehicle_spawn_distance = random.uniform(20, 30)
+            self._parked_vehicle_distance = self._leading_vehicle_spawn_distance + random.uniform(25, 35)
+            self._pedestrian_crossing_distance = self._leading_vehicle_spawn_distance + random.uniform(40, 50)
+            self._dynamic_vehicle_distance = self._leading_vehicle_spawn_distance + random.uniform(10, 20)

-            # Example code how to randomize start location
-            # distance = random.randint(20, 80)
-            # new_location, _ = get_location_in_distance(self.ego_vehicles[0], distance)
-            # waypoint = CarlaDataProvider.get_map().get_waypoint(new_location)
-            # waypoint.transform.location.z += 39
-            # self.other_actors[0].set_transform(waypoint.transform)

     def _initialize_actors(self, config):
         """
-        Custom initialization
-        """
-        waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self._first_vehicle_location)
-        transform = waypoint.transform
-        transform.location.z += 0.5
-        first_vehicle = CarlaDataProvider.request_new_actor('vehicle.nissan.patrol', transform)
-        self.other_actors.append(first_vehicle)
-
-    def _create_behavior(self):
-        """
-        The scenario defined after is a "follow leading vehicle" scenario. After
-        invoking this scenario, it will wait for the user controlled vehicle to
-        enter the start region, then make the other actor to drive until reaching
-        the next intersection. Finally, the user-controlled vehicle has to be close
-        enough to the other actor to end the scenario.
-        If this does not happen within 60 seconds, a timeout stops the scenario
+        Custom initialization for all actors.
         """
+        # 1. Leading Vehicle
+        leading_vehicle_waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self._leading_vehicle_spawn_distance)
+        leading_vehicle_transform = leading_vehicle_waypoint.transform
+        leading_vehicle_transform.location.z += 0.5 # Avoid z-fighting with road
+        self._leading_vehicle = CarlaDataProvider.request_new_actor('vehicle.nissan.patrol', leading_vehicle_transform)
+        self.other_actors.append(self._leading_vehicle)
+
+        # Determine the target intersection and its traffic light for the leading vehicle to stop
+        self._traffic_light = CarlaDataProvider.get_next_traffic_light(self._leading_vehicle, False)
+        if self._traffic_light:
+            self._next_intersection_waypoint = CarlaDataProvider.get_map().get_waypoint(self._traffic_light.get_transform().location)
+        else:
+            # Fallback if no traffic light found, find the next junction waypoint
+            curr_wp = leading_vehicle_waypoint
+            junction_wp = None
+            while True:
+                next_wps = curr_wp.next(5.0) # Look 5 meters ahead
+                if not next_wps: # End of road or invalid
+                    break
+                for nw in next_wps:
+                    if nw.is_junction:
+                        junction_wp = nw
+                        break
+                if junction_wp:
+                    break
+                curr_wp = next_wps[0] # Continue along the first path
+
+            self._next_intersection_waypoint = junction_wp if junction_wp else get_waypoint_in_distance(leading_vehicle_waypoint, 100)[0]
+
+
+        # 2. Parked Vehicle
+        parked_vehicle_waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self._parked_vehicle_distance)
+        # Offset to the right of the lane
+        parked_vehicle_offset_waypoint = get_location_in_distance_from_wp(parked_vehicle_waypoint, self._parked_vehicle_offset, False)
+        parked_vehicle_transform = carla.Transform(parked_vehicle_offset_waypoint.location, parked_vehicle_waypoint.transform.rotation)
+        parked_vehicle_transform.location.z += 0.5
```

```
178 +        self._parked_vehicle = CarlaDataProvider.request_new_actor('vehicle.tesla.model3',
    parked_vehicle_transform)
179 +        self._parked_vehicle.set_simulate_physics(False) # Parked, no physics needed
180 +        self.other_actors.append(self._parked_vehicle)
181 +
182 +        # 3. Pedestrian
183 +        pedestrian_spawn_waypoint, _ = get_waypoint_in_distance(self._reference_waypoint,
    self._pedestrian_crossing_distance)
184 +        pedestrian_start_location, pedestrian_end_location = get_crossing_point(
    pedestrian_spawn_waypoint)
185 +        pedestrian_transform = carla.Transform(pedestrian_start_location, carla.Rotation())
186 +        pedestrian_transform.location.z += 0.5 # Pedestrians are usually higher than 0 z
187 +        self._pedestrian = CarlaDataProvider.request_new_actor('walker.pedestrian.0001',
    pedestrian_transform)
188 +        self.other_actors.append(self._pedestrian)
189 +
190 +        # 4. Dynamic Vehicle (adding general traffic complexity)
191 +        dynamic_vehicle_waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self
    ._dynamic_vehicle_distance)
192 +        # Spawn in current lane or an adjacent one
193 +        if random.random() < 0.5 and dynamic_vehicle_waypoint.get_left_lane():
194 +            dynamic_vehicle_waypoint = dynamic_vehicle_waypoint.get_left_lane()
195 +        dynamic_vehicle_transform = dynamic_vehicle_waypoint.transform
196 +        dynamic_vehicle_transform.location.z += 0.5
197 +        self._dynamic_vehicle = CarlaDataProvider.request_new_actor('vehicle.audi.a2',
    dynamic_vehicle_transform)
198 +        self._dynamic_vehicle.set_autopilot(True, CarlaDataProvider.get_traffic_manager_port
    ())
199 +        self.other_actors.append(self._dynamic_vehicle)
200
201 -        # let the other actor drive until next intersection
202 -        driving_to_next_intersection = py_trees.composites.Parallel(
203 -            "DrivingTowardsIntersection",
204 -            policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ONE)
205 -
206 -        driving_to_next_intersection.add_child(WaypointFollower(self.other_actors[0], self.
    _first_vehicle_speed))
207 -        driving_to_next_intersection.add_child(InTriggerDistanceToNextIntersection(
208 -            self.other_actors[0], self._other_actor_stop_in_front_intersection))
209 -
210 -        # stop vehicle
211 -        stop = StopVehicle(self.other_actors[0], self._other_actor_max_brake)
212 -
213 -        # end condition
214 -        endcondition = py_trees.composites.Parallel("Waiting for end position",
215 -                                                    policy=py_trees.common.ParallelPolicy.
    SUCCESS_ON_ALL)
216 -        endcondition_part1 = InTriggerDistanceToVehicle(self.other_actors[0],
217 -                                                        self.ego_vehicles[0],
218 -                                                        distance=20,
219 -                                                        name="FinalDistance")
220 -        endcondition_part2 = StandStill(self.ego_vehicles[0], name="StandStill", duration=1)
221 -        endcondition.add_child(endcondition_part1)
222 -        endcondition.add_child(endcondition_part2)
223 -
224 -        # Build behavior tree
225 -        sequence = py_trees.composites.Sequence("Sequence Behavior")
226 -        sequence.add_child(driving_to_next_intersection)
227 -        sequence.add_child(stop)
228 -        sequence.add_child(endcondition)
229 -        sequence.add_child(ActorDestroy(self.other_actors[0]))
230 -
231 -        return sequence
232 -
233 -    def _create_test_criteria(self):
234 -        """
235 -        A list of all test criteria will be created that is later used
236 -        in parallel behavior tree.
237 -        """
238 -        criteria = []
239 -
240 -        collision_criterion = CollisionTest(self.ego_vehicles[0])
241 -
242 -        criteria.append(collision_criterion)
243 -
244 -        return criteria
245 -
246 -    def __del__(self):
247 -        """
248 -        Remove all actors upon deletion
249 -        """
250 -        self.remove_all_actors()
251 -
252 -
```

```
-class FollowLeadingVehicleWithObstacle(BasicScenario):
-
-    """
-    This class holds a scenario similar to FollowLeadingVehicle
-    but there is an obstacle in front of the leading vehicle
-
-    This is a single ego vehicle scenario
-    """
-
-    timeout = 120            # Timeout of scenario in seconds
-
-    def __init__(self, world, ego_vehicles, config, randomize=False, debug_mode=False, criteria_enable=True):
-        """
-        Setup all relevant parameters and create scenario
-        """
-        self._map = CarlaDataProvider.get_map()
-        self._first_actor_location = 25
-        self._second_actor_location = self._first_actor_location + 41
-        self._first_actor_speed = 10
-        self._second_actor_speed = 1.5
-        self._reference_waypoint = self._map.get_waypoint(config.trigger_points[0].location)
-        self._other_actor_max_brake = 1.0
-        self._first_actor_transform = None
-        self._second_actor_transform = None
-
-        super(FollowLeadingVehicleWithObstacle, self).__init__("FollowLeadingVehicleWithObstacle",
-                                                                ego_vehicles,
-                                                                config,
-                                                                world,
-                                                                debug_mode,
-                                                                criteria_enable=criteria_enable)
-        if randomize:
-            self._ego_other_distance_start = random.randint(4, 8)
-
-    def _initialize_actors(self, config):
-        """
-        Custom initialization
-        """
-
-        first_actor_waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self._first_actor_location)
-        second_actor_waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self._second_actor_location)
-        first_actor_transform = carla.Transform(
-            carla.Location(first_actor_waypoint.transform.location.x,
-                           first_actor_waypoint.transform.location.y,
-                           first_actor_waypoint.transform.location.z - 500),
-            first_actor_waypoint.transform.rotation)
-        self._first_actor_transform = carla.Transform(
-            carla.Location(first_actor_waypoint.transform.location.x,
-                           first_actor_waypoint.transform.location.y,
-                           first_actor_waypoint.transform.location.z + 1),
-            first_actor_waypoint.transform.rotation)
-        yaw_1 = second_actor_waypoint.transform.rotation.yaw + 90
-        second_actor_transform = carla.Transform(
-            carla.Location(second_actor_waypoint.transform.location.x,
-                           second_actor_waypoint.transform.location.y,
-                           second_actor_waypoint.transform.location.z - 500),
-            carla.Rotation(second_actor_waypoint.transform.rotation.pitch, yaw_1,
-                           second_actor_waypoint.transform.rotation.roll))
-        self._second_actor_transform = carla.Transform(
-            carla.Location(second_actor_waypoint.transform.location.x,
-                           second_actor_waypoint.transform.location.y,
-                           second_actor_waypoint.transform.location.z + 1),
-            carla.Rotation(second_actor_waypoint.transform.rotation.pitch, yaw_1,
-                           second_actor_waypoint.transform.rotation.roll))
-
-        first_actor = CarlaDataProvider.request_new_actor(
-            'vehicle.nissan.patrol', first_actor_transform)
-        second_actor = CarlaDataProvider.request_new_actor(
-            'vehicle.diamondback.century', second_actor_transform)
-
-        first_actor.set_simulate_physics(enabled=False)
-        second_actor.set_simulate_physics(enabled=False)
-        self.other_actors.append(first_actor)
-        self.other_actors.append(second_actor)

     def _create_behavior(self):
         """
-        The scenario defined after is a "follow leading vehicle" scenario. After
-        invoking this scenario, it will wait for the user controlled vehicle to
```

```
-        enter the start region, then make the other actor to drive towards obstacle.
-        Once obstacle clears the road, make the other actor to drive towards the
-        next intersection. Finally, the user-controlled vehicle has to be close
-        enough to the other actor to end the scenario.
-        If this does not happen within 60 seconds, a timeout stops the scenario
+        Creates the enhanced behavior tree for the complex scenario.
         """
-
-        # let the other actor drive until next intersection
-        driving_to_next_intersection = py_trees.composites.Parallel(
-            "Driving towards Intersection",
-            policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ONE)
-
-        obstacle_clear_road = py_trees.composites.Parallel("Obstalce clearing road",
-                                                           policy=py_trees.common.
    ParallelPolicy.SUCCESS_ON_ONE)
-        obstacle_clear_road.add_child(DriveDistance(self.other_actors[1], 4))
-        obstacle_clear_road.add_child(KeepVelocity(self.other_actors[1], self.
    _second_actor_speed))
-
-        stop_near_intersection = py_trees.composites.Parallel(
-            "Waiting for end position near Intersection",
-            policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ONE)
-        stop_near_intersection.add_child(WaypointFollower(self.other_actors[0], 10))
-        stop_near_intersection.add_child(InTriggerDistanceToNextIntersection(self.
    other_actors[0], 20))
-
-        driving_to_next_intersection.add_child(WaypointFollower(self.other_actors[0], self.
    _first_actor_speed))
-        driving_to_next_intersection.add_child(InTriggerDistanceToVehicle(self.other_actors
    [1],
-                                                                          self.other_actors
    [0], 15))
-
-        # end condition
-        endcondition = py_trees.composites.Parallel("Waiting for end position",
-                                                    policy=py_trees.common.ParallelPolicy.
    SUCCESS_ON_ALL)
-        endcondition_part1 = InTriggerDistanceToVehicle(self.other_actors[0],
-                                                        self.ego_vehicles[0],
-                                                        distance=20,
-                                                        name="FinalDistance")
-        endcondition_part2 = StandStill(self.ego_vehicles[0], name="FinalSpeed", duration=1)
-        endcondition.add_child(endcondition_part1)
-        endcondition.add_child(endcondition_part2)
-
-        # Build behavior tree
-        sequence = py_trees.composites.Sequence("Sequence Behavior")
-        sequence.add_child(ActorTransformSetter(self.other_actors[0], self.
    _first_actor_transform))
-        sequence.add_child(ActorTransformSetter(self.other_actors[1], self.
    _second_actor_transform))
-        sequence.add_child(driving_to_next_intersection)
-        sequence.add_child(StopVehicle(self.other_actors[0], self._other_actor_max_brake))
-        sequence.add_child(TimeOut(3))
-        sequence.add_child(obstacle_clear_road)
-        sequence.add_child(stop_near_intersection)
-        sequence.add_child(StopVehicle(self.other_actors[0], self._other_actor_max_brake))
-        sequence.add_child(endcondition)
-        sequence.add_child(ActorDestroy(self.other_actors[0]))
-        sequence.add_child(ActorDestroy(self.other_actors[1]))
+        # Initial actor placement
+        sequence = py_trees.composites.Sequence("ComplexFollowLeadingVehicle Scenario")
+        sequence.add_child(ActorTransformSetter(self._leading_vehicle, self._leading_vehicle.
    get_transform()))
+        sequence.add_child(ActorTransformSetter(self._parked_vehicle, self._parked_vehicle.
    get_transform()))
+        sequence.add_child(ActorTransformSetter(self._pedestrian, self._pedestrian.
    get_transform()))
+        sequence.add_child(ActorTransformSetter(self._dynamic_vehicle, self._dynamic_vehicle.
    get_transform()))
+
+        # --- Leading Vehicle Driving Behavior ---
+        leading_vehicle_driving = py_trees.composites.Parallel(
+            "LeadingVehicleDrivingToIntersection", policy=py_trees.common.ParallelPolicy.
    SUCCESS_ON_ONE
+        )
+        leading_vehicle_driving.add_child(WaypointFollower(self._leading_vehicle, self.
    _leading_vehicle_speed))
+        # Trigger when leading vehicle is close enough to its intended stop point (next
    intersection)
+        leading_vehicle_driving.add_child(InTriggerDistanceToLocation(
+            self._leading_vehicle, self._next_intersection_waypoint.transform.location,
```

```
399 +            self._leading_actor_stop_in_front_intersection + 10 # Trigger to initiate braking sequence
400 +        ))
401 +
402 +        # --- Pedestrian Crossing Behavior ---
403 +        pedestrian_cross_atomic = py_trees.composites.Sequence("Pedestrian Crossing Sequence")
404 +        # Trigger pedestrian to cross when leading vehicle is sufficiently close
405 +        pedestrian_crossing_trigger = TriggerDistanceToLocation(
406 +            self._leading_vehicle,
407 +            self._pedestrian.get_location(),
408 +            distance=15 # Trigger when leading vehicle is 15m from pedestrian's spawn point
409 +        )
410 +        _, pedestrian_end_location = get_crossing_point(self._map.get_waypoint(self._pedestrian.get_location()))
411 +        pedestrian_cross_atomic.add_child(pedestrian_crossing_trigger)
412 +        pedestrian_cross_atomic.add_child(PedestrianWalk(self._pedestrian,
413 +                                                         pedestrian_end_location.x,
414 +                                                         pedestrian_end_location.y,
415 +                                                         self._pedestrian_speed))
416 +        pedestrian_cross_atomic.add_child(StandStill(self._pedestrian, name="PedestrianStops", duration=2)) # Stand still after crossing
417 +
418 +        # --- Traffic Light Control Behavior ---
419 +        traffic_light_behavior = py_trees.composites.Sequence("Traffic Light Control")
420 +        if self._traffic_light:
421 +            traffic_light_approach_trigger = TriggerDistanceToLocation(
422 +                self._leading_vehicle, self._next_intersection_waypoint.transform.location,
423 +                distance=self._traffic_light_state_change_trigger_distance
424 +            )
425 +            traffic_light_behavior.add_child(traffic_light_approach_trigger)
426 +            traffic_light_behavior.add_child(SetTrafficLightState(self._traffic_light, carla.TrafficLightState.Red))
427 +            traffic_light_behavior.add_child(TimeOut(5)) # Stay red for 5 seconds
428 +            traffic_light_behavior.add_child(SetTrafficLightState(self._traffic_light, carla.TrafficLightState.Green))
429 +        else:
430 +            traffic_light_behavior.add_child(py_trees.behaviours.Success("No Traffic Light to Control")) # Placeholder if no traffic light
431 +
432 +        # --- Combine all dynamic elements leading up to the stop ---
433 +        mid_scenario_complexities = py_trees.composites.Parallel(
434 +            "MidScenarioComplexities", policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ALL
435 +        )
436 +        mid_scenario_complexities.add_child(leading_vehicle_driving) # Leading vehicle drives
437 +        mid_scenario_complexities.add_child(pedestrian_cross_atomic) # Pedestrian crosses
438 +        mid_scenario_complexities.add_child(traffic_light_behavior) # Traffic light changes
439 +
440 +
441 +        # --- Leading Vehicle Final Stop ---
442 +        leading_vehicle_stop_condition = InTriggerDistanceToLocation(
443 +            self._leading_vehicle, self._next_intersection_waypoint.transform.location,
444 +            self._leading_actor_stop_in_front_intersection
445 +        )
446 +        leading_vehicle_final_stop = py_trees.composites.Sequence("LeadingVehicleFinalStop")
447 +        leading_vehicle_final_stop.add_child(leading_vehicle_stop_condition)
448 +        leading_vehicle_final_stop.add_child(StopVehicle(self._leading_vehicle, self._leading_actor_max_brake))
449 +
450 +
451 +        # --- End Condition for Ego Vehicle ---
452 +        end_condition = py_trees.composites.Parallel("Waiting for Ego End Position",
453 +                                                     policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ALL)
454 +        end_condition.add_child(InTriggerDistanceToVehicle(
455 +            self._leading_vehicle, self.ego_vehicles[0], distance=10, name="FinalDistanceToLeading"))
456 +        end_condition.add_child(StandStill(self.ego_vehicles[0], name="EgoStandStill", duration=2))
457 +
458 +        # --- Build the full behavior tree ---
459 +        sequence.add_child(mid_scenario_complexities) # All parallel challenges (LV driving, pedestrian, traffic light)
460 +        sequence.add_child(leading_vehicle_final_stop) # Leading vehicle stops based on condition
461 +        sequence.add_child(end_condition)             # Ego must stop behind
462 +        sequence.add_child(ActorDestroy(self._leading_vehicle))
463 +        sequence.add_child(ActorDestroy(self._parked_vehicle))
464 +        sequence.add_child(ActorDestroy(self._pedestrian))
465 +        sequence.add_child(ActorDestroy(self._dynamic_vehicle)) # Clean up dynamic vehicle
466
467          return sequence
468
469 @@ -296,9 +268,17 @@ class FollowLeadingVehicleWithObstacle(BasicScenario):
```

```
470             """
471             criteria = []
472 
473 -           collision_criterion = CollisionTest(self.ego_vehicles[0])
474 +           collision_ego_leading = CollisionTest(self.ego_vehicles[0], self._leading_vehicle,
        name="EgoLeadingCollision")
475 +           collision_ego_parked = CollisionTest(self.ego_vehicles[0], self._parked_vehicle, name
        ="EgoParkedCollision")
476 +           collision_ego_pedestrian = CollisionTest(self.ego_vehicles[0], self._pedestrian, name
        ="EgoPedestrianCollision")
477 +           collision_ego_dynamic = CollisionTest(self.ego_vehicles[0], self._dynamic_vehicle,
        name="EgoDynamicCollision")
478 +           collision_leading_pedestrian = CollisionTest(self._leading_vehicle, self._pedestrian,
         name="LeadingPedestrianCollision")
479 
480 -           criteria.append(collision_criterion)
481 +           criteria.append(collision_ego_leading)
482 +           criteria.append(collision_ego_parked)
483 +           criteria.append(collision_ego_pedestrian)
484 +           criteria.append(collision_ego_dynamic)
485 +           criteria.append(collision_leading_pedestrian)
486 
487             return criteria
488 
489 @@ -306,4 +286,4 @@ class FollowLeadingVehicleWithObstacle(BasicScenario):
490             """
491             Remove all actors upon deletion
492             """
493 -           self.remove_all_actors()
494 +           self.remove_all_actors()
495 \ No newline at end of file
```

Listing A.1: The diff of the initial LLM-enhanced Follow vehicle scenario, highlighting *how* the LLM enhanced the scenario.

A.1.2 Strictly adhering to the Carla API

```
1  diff --git a/srunner/scenarios/follow_leading_vehicle.py b/srunner/scenarios/
       follow_leading_vehicle.py
2  index 5f60061..2117eae 100644
3  --- a/srunner/scenarios/follow_leading_vehicle.py
4  +++ b/srunner/scenarios/follow_leading_vehicle.py
5  @@ -6,14 +6,19 @@
6   # For a copy, see <https://opensource.org/licenses/MIT>.
7 
8   """
9  -Follow leading vehicle scenario:
10 +Enhanced Follow leading vehicle scenario:
11 
12  The scenario realizes a common driving behavior, in which the
13  user-controlled ego vehicle follows a leading car driving down
14 -a given road. At some point the leading car has to slow down and
15 +a given road. This enhanced version introduces more complexity:
16 +1. A pedestrian crosses the road in front of the leading vehicle, forcing it
17 +   to react and slow down/stop unexpectedly.
18 +2. A parked vehicle partially obstructs the road near the final stopping point,
19 +   decreasing visibility and narrowing the path.
20 +At some point, the leading car has to slow down and
21  finally stop. The ego vehicle has to react accordingly to avoid
22 -a collision. The scenario ends either via a timeout, or if the ego
23 -vehicle stopped close enough to the leading vehicle
24 +a collision and navigate the increased complexity. The scenario ends either via a timeout, or
        if the ego
25 +vehicle stopped close enough to the leading vehicle.
26  """
27 
28  import random
29 @@ -27,7 +32,8 @@ from srunner.scenariomanager.scenarioatomics.atomic_behaviors import (
       ActorTrans
30                                                                      ActorDestroy,
31                                                                      KeepVelocity,
32                                                                      StopVehicle,
33 -                                                                    WaypointFollower)
34 +                                                                    WaypointFollower,
35 +                                                                    WalkToLocation)
36  from srunner.scenariomanager.scenarioatomics.atomic_criteria import CollisionTest
37  from srunner.scenariomanager.scenarioatomics.atomic_trigger_conditions import (
       InTriggerDistanceToVehicle,
38 
       InTriggerDistanceToNextIntersection,
39 @@ -41,13 +47,13 @@ from srunner.tools.scenario_helper import get_waypoint_in_distance
```

```
class FollowLeadingVehicle(BasicScenario):

    """
-    This class holds everything required for a simple "Follow a leading vehicle"
-    scenario involving two vehicles.  (Traffic Scenario 2)
+    This class holds everything required for a complex "Follow a leading vehicle"
+    scenario involving two vehicles, a pedestrian, and a parked car. (Traffic Scenario 2)

-    This is a single ego vehicle scenario
+    This is a single ego vehicle scenario, with enhanced complexity.
    """

-    timeout = 120            # Timeout of scenario in seconds
+    timeout = 180            # Increased timeout of scenario in seconds for more complexity

    def __init__(self, world, ego_vehicles, config, randomize=False, debug_mode=False, criteria_enable=True,
                 timeout=60):
@@ -58,16 +64,22 @@ class FollowLeadingVehicle(BasicScenario):
        """

        self._map = CarlaDataProvider.get_map()
-        self._first_vehicle_location = 25
+        self._first_vehicle_location = 25 # Distance of leading vehicle from ego's trigger point
        self._first_vehicle_speed = 10
        self._reference_waypoint = self._map.get_waypoint(config.trigger_points[0].location)
        self._other_actor_max_brake = 1.0
-        self._other_actor_stop_in_front_intersection = 20
-        self._other_actor_transform = None
+        self._other_actor_stop_in_front_intersection = 20 # Distance from intersection for leading vehicle to stop
+        self._leading_vehicle_transform = None # Store leading vehicle's initial transform
+
+        # New parameters for added complexity
+        self._pedestrian_start_distance = self._first_vehicle_location + 15 # Pedestrian appears 15m ahead of leading car's start
+        self._parked_vehicle_distance = self._first_vehicle_location + 30 # Parked car appears 30m ahead of leading car's start
+        self._parked_vehicle_lateral_offset = 3.0 # Offset from center of lane for parked car
+
        # Timeout of scenario in seconds
-        self.timeout = timeout
+        self.timeout = timeout if timeout else self.timeout

-        super(FollowLeadingVehicle, self).__init__("FollowVehicle",
+        super(FollowLeadingVehicle, self).__init__("FollowVehicleComplex", # Changed scenario name
                                                   ego_vehicles,
                                                   config,
                                                   world,
@@ -76,216 +88,114 @@ class FollowLeadingVehicle(BasicScenario):

        if randomize:
            self._ego_other_distance_start = random.randint(4, 8)
-
-            # Example code how to randomize start location
-            # distance = random.randint(20, 80)
-            # new_location, _ = get_location_in_distance(self.ego_vehicles[0], distance)
-            # waypoint = CarlaDataProvider.get_map().get_waypoint(new_location)
-            # waypoint.transform.location.z += 39
-            # self.other_actors[0].set_transform(waypoint.transform)
+            # Further randomization of speeds, distances, pedestrian timing could be added here.

    def _initialize_actors(self, config):
        """
-        Custom initialization
+        Custom initialization of leading vehicle, pedestrian, and parked vehicle.
        """
+        # 1. Leading Vehicle
        waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self._first_vehicle_location)
        transform = waypoint.transform
        transform.location.z += 0.5
        first_vehicle = CarlaDataProvider.request_new_actor('vehicle.nissan.patrol', transform)
        self.other_actors.append(first_vehicle)
+        self._leading_vehicle_transform = transform # Store for potential ActorTransformSetter later if needed

+        # 2. Pedestrian
+        ped_waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self._pedestrian_start_distance)
```

```
+
+        # Calculate pedestrian spawn point on the right sidewalk (relative to vehicle's forward direction)
+        ped_spawn_location = ped_waypoint.transform.location
+        # Move pedestrian to the right of the lane, assuming 1.0m for sidewalk
+        ped_spawn_location += ped_waypoint.transform.get_right_vector() * (ped_waypoint.lane_width / 2.0 + 1.0)
+        ped_spawn_location.z += 0.5 # Adjust Z to be on the ground
+
+        ped_transform = carla.Transform(ped_spawn_location, ped_waypoint.transform.rotation)
+        pedestrian = CarlaDataProvider.request_new_actor('walker.pedestrian.0001', ped_transform)
+        self.other_actors.append(pedestrian)
+
+        # 3. Parked Vehicle
+        parked_waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self._parked_vehicle_distance)
+        # Parked vehicle on the side of the road, slightly obstructing vision/lane
+        parked_transform = carla.Transform(
+            carla.Location(parked_waypoint.transform.location.x,
+                           parked_waypoint.transform.location.y + self._parked_vehicle_lateral_offset, # Offset to the side
+                           parked_waypoint.transform.location.z + 0.1), # Ensure it's on the ground
+            parked_waypoint.transform.rotation
+        )
+        parked_vehicle = CarlaDataProvider.request_new_actor('vehicle.volkswagen.t2', parked_transform)
+        parked_vehicle.set_autopilot(False) # Ensure it doesn't move
+        parked_vehicle.set_simulate_physics(False) # Optional, prevents accidental movement
+        self.other_actors.append(parked_vehicle)
+
-    def _create_behavior(self):
-        """
-        The scenario defined after is a "follow leading vehicle" scenario. After
-        invoking this scenario, it will wait for the user controlled vehicle to
-        enter the start region, then make the other actor to drive until reaching
-        the next intersection. Finally, the user-controlled vehicle has to be close
-        enough to the other actor to end the scenario.
-        If this does not happen within 60 seconds, a timeout stops the scenario
-        """
-
-        # let the other actor drive until next intersection
-        driving_to_next_intersection = py_trees.composites.Parallel(
-            "DrivingTowardsIntersection",
-            policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ONE)
-
-        driving_to_next_intersection.add_child(WaypointFollower(self.other_actors[0], self._first_vehicle_speed))
-        driving_to_next_intersection.add_child(InTriggerDistanceToNextIntersection(
-            self.other_actors[0], self._other_actor_stop_in_front_intersection))
-
-        # stop vehicle
-        stop = StopVehicle(self.other_actors[0], self._other_actor_max_brake)
-
-        # end condition
-        endcondition = py_trees.composites.Parallel("Waiting for end position",
-                                                    policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ALL)
-        endcondition_part1 = InTriggerDistanceToVehicle(self.other_actors[0],
-                                                        self.ego_vehicles[0],
-                                                        distance=20,
-                                                        name="FinalDistance")
-        endcondition_part2 = StandStill(self.ego_vehicles[0], name="StandStill", duration=1)
-        endcondition.add_child(endcondition_part1)
-        endcondition.add_child(endcondition_part2)
-
-        # Build behavior tree
-        sequence = py_trees.composites.Sequence("Sequence Behavior")
-        sequence.add_child(driving_to_next_intersection)
-        sequence.add_child(stop)
-        sequence.add_child(endcondition)
-        sequence.add_child(ActorDestroy(self.other_actors[0]))
-
-        return sequence
-
-    def _create_test_criteria(self):
-        """
-        A list of all test criteria will be created that is later used
-        in parallel behavior tree.
-        """
-        criteria = []
-
-        collision_criterion = CollisionTest(self.ego_vehicles[0])
```

```
188 -
189 -        criteria.append(collision_criterion)
190 -
191 -        return criteria
192 -
193 -    def __del__(self):
194 -        """
195 -        Remove all actors upon deletion
196 -        """
197 -        self.remove_all_actors()
198 -
199 -
200 -class FollowLeadingVehicleWithObstacle(BasicScenario):
201 -
202 -    """
203 -    This class holds a scenario similar to FollowLeadingVehicle
204 -    but there is an obstacle in front of the leading vehicle
205 -
206 -    This is a single ego vehicle scenario
207 -    """
208 -
209 -    timeout = 120            # Timeout of scenario in seconds
210 -
211 -    def __init__(self, world, ego_vehicles, config, randomize=False, debug_mode=False,
      criteria_enable=True):
212 -        """
213 -        Setup all relevant parameters and create scenario
214 -        """
215 -        self._map = CarlaDataProvider.get_map()
216 -        self._first_actor_location = 25
217 -        self._second_actor_location = self._first_actor_location + 41
218 -        self._first_actor_speed = 10
219 -        self._second_actor_speed = 1.5
220 -        self._reference_waypoint = self._map.get_waypoint(config.trigger_points[0].location)
221 -        self._other_actor_max_brake = 1.0
222 -        self._first_actor_transform = None
223 -        self._second_actor_transform = None
224 -
225 -        super(FollowLeadingVehicleWithObstacle, self).__init__("
      FollowLeadingVehicleWithObstacle",
226 -                                                               ego_vehicles,
227 -                                                               config,
228 -                                                               world,
229 -                                                               debug_mode,
230 -                                                               criteria_enable=
      criteria_enable)
231 -        if randomize:
232 -            self._ego_other_distance_start = random.randint(4, 8)
233 -
234 -    def _initialize_actors(self, config):
235 -        """
236 -        Custom initialization
237 -        """
238 -
239 -        first_actor_waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self.
      _first_actor_location)
240 -        second_actor_waypoint, _ = get_waypoint_in_distance(self._reference_waypoint, self.
      _second_actor_location)
241 -        first_actor_transform = carla.Transform(
242 -            carla.Location(first_actor_waypoint.transform.location.x,
243 -                           first_actor_waypoint.transform.location.y,
244 -                           first_actor_waypoint.transform.location.z - 500),
245 -            first_actor_waypoint.transform.rotation)
246 -        self._first_actor_transform = carla.Transform(
247 -            carla.Location(first_actor_waypoint.transform.location.x,
248 -                           first_actor_waypoint.transform.location.y,
249 -                           first_actor_waypoint.transform.location.z + 1),
250 -            first_actor_waypoint.transform.rotation)
251 -        yaw_1 = second_actor_waypoint.transform.rotation.yaw + 90
252 -        second_actor_transform = carla.Transform(
253 -            carla.Location(second_actor_waypoint.transform.location.x,
254 -                           second_actor_waypoint.transform.location.y,
255 -                           second_actor_waypoint.transform.location.z - 500),
256 -            carla.Rotation(second_actor_waypoint.transform.rotation.pitch, yaw_1,
257 -                           second_actor_waypoint.transform.rotation.roll))
258 -        self._second_actor_transform = carla.Transform(
259 -            carla.Location(second_actor_waypoint.transform.location.x,
260 -                           second_actor_waypoint.transform.location.y,
261 -                           second_actor_waypoint.transform.location.z + 1),
262 -            carla.Rotation(second_actor_waypoint.transform.rotation.pitch, yaw_1,
263 -                           second_actor_waypoint.transform.rotation.roll))
264 -
265 -        first_actor = CarlaDataProvider.request_new_actor(
266 -            'vehicle.nissan.patrol', first_actor_transform)
```

```
267 -         second_actor = CarlaDataProvider.request_new_actor(
268 -             'vehicle.diamondback.century', second_actor_transform)
269 -
270 -         first_actor.set_simulate_physics(enabled=False)
271 -         second_actor.set_simulate_physics(enabled=False)
272 -         self.other_actors.append(first_actor)
273 -         self.other_actors.append(second_actor)
274
275       def _create_behavior(self):
276           """
277 -         The scenario defined after is a "follow leading vehicle" scenario. After
278 -         invoking this scenario, it will wait for the user controlled vehicle to
279 -         enter the start region, then make the other actor to drive towards obstacle.
280 -         Once obstacle clears the road, make the other actor to drive towards the
281 -         next intersection. Finally, the user-controlled vehicle has to be close
282 -         enough to the other actor to end the scenario.
283 -         If this does not happen within 60 seconds, a timeout stops the scenario
284 -         """
285 -
286 -         # let the other actor drive until next intersection
287 -         driving_to_next_intersection = py_trees.composites.Parallel(
288 -             "Driving towards Intersection",
289 -             policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ONE)
290 -
291 -         obstacle_clear_road = py_trees.composites.Parallel("Obstalce clearing road",
292 -                                                            policy=py_trees.common.
    ParallelPolicy.SUCCESS_ON_ONE)
293 -         obstacle_clear_road.add_child(DriveDistance(self.other_actors[1], 4))
294 -         obstacle_clear_road.add_child(KeepVelocity(self.other_actors[1], self.
    _second_actor_speed))
295 -
296 -         stop_near_intersection = py_trees.composites.Parallel(
297 -             "Waiting for end position near Intersection",
298 -             policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ONE)
299 -         stop_near_intersection.add_child(WaypointFollower(self.other_actors[0], 10))
300 -         stop_near_intersection.add_child(InTriggerDistanceToNextIntersection(self.
    other_actors[0], 20))
301 -
302 -         driving_to_next_intersection.add_child(WaypointFollower(self.other_actors[0], self.
    _first_actor_speed))
303 -         driving_to_next_intersection.add_child(InTriggerDistanceToVehicle(self.other_actors
    [1],
304 -                                                                           self.other_actors
    [0], 15))
305 -
306 -         # end condition
307 +         The scenario defined after is an enhanced "follow leading vehicle" scenario.
308 +         It integrates a pedestrian crossing and a parked vehicle.
309 +         """
310 +
311 +         # 1. Leading vehicle drives, potentially reacting to pedestrian
312 +         leading_vehicle_driving = py_trees.composites.Parallel(
313 +             "LeadingVehicleDrivingAndPedestrianCrossing",
314 +             policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ALL)
315 +
316 +         # Leading vehicle follows waypoints
317 +         leading_vehicle_follower = WaypointFollower(self.other_actors[0], self.
    _first_vehicle_speed) # self.other_actors[0] is leading vehicle
318 +
319 +         # Pedestrian crosses the road (self.other_actors[1])
320 +         # Calculate target location for pedestrian on the left sidewalk
321 +         ped_waypoint_for_crossing, _ = get_waypoint_in_distance(self._reference_waypoint,
    self._pedestrian_start_distance)
322 +         ped_target_location = ped_waypoint_for_crossing.transform.location
323 +         # Move pedestrian to the left of the lane, assuming 1.0m for sidewalk
324 +         ped_target_location -= ped_waypoint_for_crossing.transform.get_right_vector() * (
    ped_waypoint_for_crossing.lane_width / 2.0 + 1.0)
325 +         ped_target_location.z += 0.5
326 +
327 +         pedestrian_cross_behavior = WalkToLocation(self.other_actors[1], ped_target_location)
328 +
329 +         # Introduce a delay for pedestrian to start crossing AFTER leading vehicle has
    started moving.
330 +         pedestrian_start_sequence = py_trees.composites.Sequence("PedestrianStartSequence")
331 +         pedestrian_start_sequence.add_child(TimeOut(5)) # Wait 5 seconds after scenario start
332 +         pedestrian_start_sequence.add_child(pedestrian_cross_behavior)
333 +
334 +         # Leading vehicle drives while the pedestrian sequence is active.
335 +         # Carla's AI for vehicles should automatically react to the pedestrian.
336 +         leading_vehicle_driving.add_child(leading_vehicle_follower)
337 +         leading_vehicle_driving.add_child(pedestrian_start_sequence)
338 +
339 +         # Wait for leading vehicle to be close to the intersection to decide to stop (its
    final stop)
```

```
+        leading_vehicle_at_intersection_trigger = InTriggerDistanceToNextIntersection(
+            self.other_actors[0], self._other_actor_stop_in_front_intersection
+        )
+
+        # Sequence for the leading vehicle's overall journey: drive, reach intersection, then stop
+        leading_vehicle_overall_journey = py_trees.composites.Sequence("LeadingVehicleJourney")
+        leading_vehicle_overall_journey.add_child(leading_vehicle_driving) # Includes pedestrian interaction
+        leading_vehicle_overall_journey.add_child(leading_vehicle_at_intersection_trigger)
+        leading_vehicle_overall_journey.add_child(StopVehicle(self.other_actors[0], self._other_actor_max_brake)) # Stops at intersection
+
+        # 2. End condition for Ego Vehicle
+        # The ego vehicle must stop close enough to the leading vehicle and stand still.
         endcondition = py_trees.composites.Parallel("Waiting for end position",
                                                     policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ALL)
-        endcondition_part1 = InTriggerDistanceToVehicle(self.other_actors[0],
+        endcondition_part1 = InTriggerDistanceToVehicle(self.other_actors[0], # Leading vehicle
                                                         self.ego_vehicles[0],
-                                                        distance=20,
+                                                        distance=10, # Reduced distance for tighter following
                                                         name="FinalDistance")
-        endcondition_part2 = StandStill(self.ego_vehicles[0], name="FinalSpeed", duration=1)
+        endcondition_part2 = StandStill(self.ego_vehicles[0], name="StandStill", duration=2) # Increased duration for stand still
         endcondition.add_child(endcondition_part1)
         endcondition.add_child(endcondition_part2)

-        # Build behavior tree
-        sequence = py_trees.composites.Sequence("Sequence Behavior")
-        sequence.add_child(ActorTransformSetter(self.other_actors[0], self._first_actor_transform))
-        sequence.add_child(ActorTransformSetter(self.other_actors[1], self._second_actor_transform))
-        sequence.add_child(driving_to_next_intersection)
-        sequence.add_child(StopVehicle(self.other_actors[0], self._other_actor_max_brake))
-        sequence.add_child(TimeOut(3))
-        sequence.add_child(obstacle_clear_road)
-        sequence.add_child(stop_near_intersection)
-        sequence.add_child(StopVehicle(self.other_actors[0], self._other_actor_max_brake))
+        # Build overall behavior tree
+        sequence = py_trees.composites.Sequence("ComplexFollowVehicleBehavior")
+        sequence.add_child(leading_vehicle_overall_journey)
         sequence.add_child(endcondition)
-        sequence.add_child(ActorDestroy(self.other_actors[0]))
-        sequence.add_child(ActorDestroy(self.other_actors[1]))
+
+        # Clean up all actors at the end
+        sequence.add_child(ActorDestroy(self.other_actors[0], name="DestroyLeadingVehicle"))
+        sequence.add_child(ActorDestroy(self.other_actors[1], name="DestroyPedestrian"))
+        sequence.add_child(ActorDestroy(self.other_actors[2], name="DestroyParkedVehicle")) # self.other_actors[2] is parked vehicle

         return sequence

@@ -297,6 +207,10 @@ class FollowLeadingVehicleWithObstacle(BasicScenario):
         criteria = []

         collision_criterion = CollisionTest(self.ego_vehicles[0])
+        # Additional criteria could be added here, e.g.,
+        #    - OffRoadTest: if the parked car forces the ego off the road.
+        #    - RunningRedLightTest: if there's an intersection with a traffic light.
+        #    - DrivenDistanceTest: to ensure progress.

         criteria.append(collision_criterion)

@@ -306,4 +220,4 @@ class FollowLeadingVehicleWithObstacle(BasicScenario):
         """
         Remove all actors upon deletion
         """
-        self.remove_all_actors()
+        self.remove_all_actors()
\ No newline at end of file
```

Listing A.2: The diff of the iterated, more strict LLM-enhanced Follow vehicle scenario.

A.2 Accident

A.2.1 Minimal changes

This scenario is outlined in Section 6.1.2 → p.44 and analysed in Section 7.1.4 → p.51.

```
diff --git a/srunner/scenarios/route_obstacles.py b/srunner/scenarios/route_obstacles.py
index 8a0a3dc..86892b7 100644
--- a/srunner/scenarios/route_obstacles.py
+++ b/srunner/scenarios/route_obstacles.py
@@ -66,13 +66,17 @@ class Accident(BasicScenario):
         self._map = CarlaDataProvider.get_map()
         self.timeout = timeout

+        # Original distances
         self._first_distance = 10
         self._second_distance = 6
+
+        # NEW: Third accident vehicle distance, increasing complexity
+        self._third_distance = 6

         self._trigger_distance = 50
         self._end_distance = 50
         self._wait_duration = 5
-        self._offset = 0.6
+        self._offset = 0.6 # Original offset for accident vehicles

         self._lights = carla.VehicleLightState.Special1 | carla.VehicleLightState.Special2 | carla.VehicleLightState.Position

@@ -84,6 +88,70 @@ class Accident(BasicScenario):
         self._max_speed = get_value_parameter(config, 'speed', float, 60)
         self._scenario_timeout = 240

+        # NEW: Pedestrian parameters, enhancing dynamic complexity
+        self._pedestrian_bps = [ "NOTE: Removed for brevity in the thesis"  ]
+
+        self._pedestrian_speed = get_value_parameter(config, 'pedestrian_speed', float, 1.5) # ~5.4 km/h
+        self._pedestrian_drive_distance = get_value_parameter(config, 'pedestrian_drive_distance', float, 10)
+        self._pedestrian_offset = 0.9 # Further to the side, potentially on sidewalk, for spawning
+        self._pedestrian_trigger_distance = 20 # Smaller trigger for pedestrian start
+
+        # NEW: Traffic cone parameters, enhancing static obstruction
+        self._cone_offset = 0.8 # Slightly more off the lane than accident cars, but less than pedestrian
+
         super().__init__(
             "Accident", ego_vehicles, config, world, randomize, debug_mode, criteria_enable=criteria_enable)

@@ -178,37 +246,125 @@ class Accident(BasicScenario):
         second_vehicle_wp = self._move_waypoint_forward(self._first_vehicle_wp, self._second_distance)
         second_actor = self._spawn_obstacle(second_vehicle_wp, 'vehicle.*', True)

-        self._accident_wp = second_vehicle_wp
-        self._end_wp = self._move_waypoint_forward(second_vehicle_wp, self._end_distance)
-
         # Set its initial conditions
         second_actor.apply_control(carla.VehicleControl(hand_brake=True))
         self.other_actors.append(second_actor)

+        # NEW: Create a third vehicle that has been in the accident, increasing obstruction
+        third_vehicle_wp = self._move_waypoint_forward(second_vehicle_wp, self._third_distance)
+        third_actor = self._spawn_obstacle(third_vehicle_wp, 'vehicle.*', True)
+
+        # Set its initial conditions
+        third_actor.apply_control(carla.VehicleControl(hand_brake=True))
+        self.other_actors.append(third_actor)
+
+        self._accident_wp = third_vehicle_wp # Update the main accident waypoint to the last car
+        self._end_wp = self._move_waypoint_forward(third_vehicle_wp, self._end_distance)
+
+        # NEW: Spawn traffic cones around the accident, further narrowing the path
+        cone_ref_wps = [
+            self._move_waypoint_forward(self._accident_wp, -self._first_distance - self._second_distance - 2), # Before first car
```

```
+            self._first_vehicle_wp,
+            self._move_waypoint_forward(self._first_vehicle_wp, self._second_distance / 2), # In between first and second
+            second_vehicle_wp,
+            self._move_waypoint_forward(second_vehicle_wp, self._third_distance / 2), # In between second and third
+            third_vehicle_wp,
+            self._move_waypoint_forward(third_vehicle_wp, 2) # After third car
+        ]
+
+        for i, wp in enumerate(cone_ref_wps):
+            displacement = self._cone_offset * wp.lane_width / 2
+            r_vec = wp.transform.get_right_vector()
+            if self._direction == 'left':
+                r_vec *= -1
+            # Make a copy of the transform!
+            cone_transform = carla.Transform(wp.transform.location, wp.transform.rotation)
+            cone_transform.location += carla.Location(x=displacement * r_vec.x, y=displacement * r_vec.y, z=0.1)
+            # cone_blueprint = 'static.prop.construction.trafficcone01'
+            cone_blueprint = 'static.prop.trafficcone01'  # Or any available propcone_blueprint = 'static.prop.construction.trafficcone01'
+            cone_actor = CarlaDataProvider.request_new_actor(cone_blueprint, cone_transform)
+            if not cone_actor:
+                raise ValueError(f"Couldn't spawn cone {i}")
+            cone_actor.set_simulate_physics(False)
+            self.other_actors.append(cone_actor)
+
+        # NEW: Spawn a pedestrian near the accident, adding dynamic interaction
+        pedestrian_spawn_wp = self._move_waypoint_forward(self._first_vehicle_wp, self._second_distance / 2) # In between first and second accident cars
+        displacement = self._pedestrian_offset * pedestrian_spawn_wp.lane_width / 2 # Further to the side, simulate sidewalk
+        r_vec = pedestrian_spawn_wp.transform.get_right_vector()
+        if self._direction == 'left':
+            r_vec *= -1
+        pedestrian_transform = pedestrian_spawn_wp.transform
+        pedestrian_transform.location += carla.Location(x=displacement * r_vec.x, y=displacement * r_vec.y, z=0.1) # Pedestrian on ground
+        # pedestrian_transform.location.z = 0.5
+
+        rng = CarlaDataProvider.get_random_seed()
+        ped_blueprint = rng.choice(self._pedestrian_bps)
+        pedestrian_actor = CarlaDataProvider.request_new_actor(ped_blueprint, pedestrian_transform)
+        if not pedestrian_actor:
+            #raise ValueError("Couldn't spawn pedestrian actor")
+            print("Couldn't spawn pedestrian actor")
+            return
+        pedestrian_actor.set_simulate_physics(True) # Pedestrians are typically simulated
+        self.other_actors.append(pedestrian_actor)
+
+        # Define pedestrian target location (slightly moving towards the road to be more intrusive)
+        ped_target_wp = self._move_waypoint_forward(pedestrian_spawn_wp, self._pedestrian_drive_distance)
+        self._pedestrian_target_loc = ped_target_wp.transform.location
+        # Shift target location slightly into the lane for more challenge
+        if self._direction == 'right':
+            self._pedestrian_target_loc.y -= 0.5 # Shift left (towards road center) for right-hand traffic
+        else: # 'left'
+            self._pedestrian_target_loc.y += 0.5 # Shift right (towards road center) for left-hand traffic
+
     def _create_behavior(self):
         """
-        The vehicle has to drive the reach a specific point but an accident is in the middle of the road,
-        blocking its route and forcing it to lane change.
+        The vehicle has to drive to reach a specific point but an accident is in the middle of the road,
+        blocking its route and forcing it to lane change. Enhanced with more complexity.
         """
         root = py_trees.composites.Sequence(name="Accident")
         if self.route_mode:
-            total_dist = self._distance + self._first_distance + self._second_distance + 20
+            # Update total_dist for LeaveSpaceInFront to include the new third vehicle
+            total_dist = self._distance + self._first_distance + self._second_distance + self._third_distance + 20
             root.add_child(LeaveSpaceInFront(total_dist))

-        end_condition = py_trees.composites.Parallel(policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ONE)
```

```
-         end_condition.add_child(ScenarioTimeout(self._scenario_timeout, self.config.name))
-         end_condition.add_child(WaitUntilInFrontPosition(self.ego_vehicles[0], self._end_wp.transform, False))
+         # Main parallel behavior for the active part of the scenario
+         main_behavior_parallel = py_trees.composites.Parallel(
+             policy=py_trees.common.ParallelPolicy.SUCCESS_ON_ONE, name="Main Behavior Parallel")
+
+         main_behavior_parallel.add_child(ScenarioTimeout(self._scenario_timeout, self.config.name))

-         behavior = py_trees.composites.Sequence()
-         behavior.add_child(InTriggerDistanceToLocation(
+         # 1. Ego-centric behavior (triggers speed/waiting)
+         ego_trigger_behavior = py_trees.composites.Sequence(name="Ego Trigger Behavior")
+         ego_trigger_behavior.add_child(InTriggerDistanceToLocation(
              self.ego_vehicles[0], self._first_vehicle_wp.transform.location, self._trigger_distance))
-         behavior.add_child(Idle(self._wait_duration))
+         ego_trigger_behavior.add_child(Idle(self._wait_duration))
          if self.route_mode:
-             behavior.add_child(SetMaxSpeed(self._max_speed))
-         behavior.add_child(WaitForever())
-
-         end_condition.add_child(behavior)
-         root.add_child(end_condition)
+             ego_trigger_behavior.add_child(SetMaxSpeed(self._max_speed))
+         ego_trigger_behavior.add_child(WaitForever()) # This ensures ego behavior lasts until something else finishes the parallel
+
+         main_behavior_parallel.add_child(ego_trigger_behavior)
+
+         # 2. Pedestrian movement behavior (last actor added is the pedestrian)
+         pedestrian_actor = self.other_actors[-1]
+         if pedestrian_actor.type_id.startswith('walker'): # Ensure it's actually a walker type
+             pedestrian_behavior = py_trees.composites.Sequence(name="Pedestrian Movement")
+             # Pedestrian starts moving when ego is closer to its location
+             pedestrian_behavior.add_child(InTriggerDistanceToLocation(
+                 self.ego_vehicles[0], pedestrian_actor.get_location(), self._pedestrian_trigger_distance))
+             # BasicAgentBehavior for the pedestrian
+             pedestrian_behavior.add_child(BasicAgentBehavior(
+                 pedestrian_actor, self._pedestrian_target_loc, target_speed=self._pedestrian_speed))
+             pedestrian_behavior.add_child(HandBrakeVehicle(pedestrian_actor, 1)) # Stop movement after reaching target (or collision)
+             pedestrian_behavior.add_child(WaitForever()) # Keep it active, don't let it stop the parallel
+             main_behavior_parallel.add_child(pedestrian_behavior)
+
+         # 3. Scenario End Condition (ego reaches a point past the accident)
+         scenario_end_condition = py_trees.composites.Sequence(name="Scenario End Condition")
+         scenario_end_condition.add_child(WaitUntilInFrontPosition(self.ego_vehicles[0], self._end_wp.transform, False))
+         main_behavior_parallel.add_child(scenario_end_condition)
+
+         root.add_child(main_behavior_parallel)

          if self.route_mode:
              root.add_child(SetMaxSpeed(0))
```

Listing A.3: The diff of a minimally LLM-enhanced Accident scenario.

A.2.2 Optimized for jerk

This scenario is also outlined in Section 6.1.2 →p.44 and analysed in Section 7.1.4 →p.51.

First iteration

```
diff --git a/srunner/scenarios/route_obstacles.py b/srunner/scenarios/route_obstacles.py
index 8a0a3dc..fe46632 100644
--- a/srunner/scenarios/route_obstacles.py
+++ b/srunner/scenarios/route_obstacles.py
@@ -69,9 +69,9 @@ class Accident(BasicScenario):
         self._first_distance = 10
```

```
          self._second_distance = 6

-         self._trigger_distance = 50
+         self._trigger_distance = 20  # Decreased to force sharper reactions
          self._end_distance = 50
-         self._wait_duration = 5
+         self._wait_duration = 1  # Decreased to allow less reaction time
          self._offset = 0.6

          self._lights = carla.VehicleLightState.Special1 | carla.VehicleLightState.Special2 |
     carla.VehicleLightState.Position
@@ -297,9 +297,9 @@ class ParkedObstacle(BasicScenario):
          self._map = CarlaDataProvider.get_map()
          self.timeout = timeout

-         self._trigger_distance = 50
+         self._trigger_distance = 20  # Decreased to force sharper reactions
          self._end_distance = 50
-         self._wait_duration = 5
+         self._wait_duration = 1  # Decreased to allow less reaction time
          self._offset = 0.7

          self._lights = carla.VehicleLightState.RightBlinker | carla.VehicleLightState.
     LeftBlinker | carla.VehicleLightState.Position
@@ -503,13 +503,13 @@ class HazardAtSideLane(BasicScenario):
          self._map = CarlaDataProvider.get_map()
          self.timeout = timeout

-         self._obstacle_distance = 9
-         self._trigger_distance = 50
+         self._obstacle_distance = 5  # Reduced distance between bicycles for tighter space
+         self._trigger_distance = 20  # Decreased to force sharper reactions
          self._end_distance = 50
          self._extra_space = 30

-         self._offset = 0.55
-         self._wait_duration = 5
+         self._offset = 0.8  # Increased to make bicycles encroach more into the lane
+         self._wait_duration = 1  # Decreased to allow less reaction time

          self._target_locs = []

@@ -517,7 +517,7 @@ class HazardAtSideLane(BasicScenario):

          self._distance = get_value_parameter(config, 'distance', float, 100)
          self._max_speed = get_value_parameter(config, 'speed', float, 60)
-         self._bicycle_speed = get_value_parameter(config, 'bicycle_speed', float, 10)
+         self._bicycle_speed = get_value_parameter(config, 'bicycle_speed', float, 20)  #
     Increased bicycle speed
          self._bicycle_drive_distance = get_value_parameter(config, 'bicycle_drive_distance',
     float, 50)
          self._scenario_timeout = 240

@@ -661,7 +661,7 @@ class HazardAtSideLaneTwoWays(HazardAtSideLane):
     """
     def __init__(self, world, ego_vehicles, config, randomize=False, debug_mode=False,
     criteria_enable=True, timeout=180):

-         self._opposite_frequency = get_value_parameter(config, 'frequency', float, 100)
+         self._opposite_frequency = get_value_parameter(config, 'frequency', float, 20) #
     Increased opposite traffic frequency (lower spawn_dist)

          super().__init__(world, ego_vehicles, config, randomize, debug_mode, criteria_enable,
      timeout)

@@ -716,4 +716,4 @@ class HazardAtSideLaneTwoWays(HazardAtSideLane):
          for actor in self.other_actors:
              root.add_child(ActorDestroy(actor))

-         return root
+         return root
\ No newline at end of file
```

Listing A.4: The diff of the jerk-optimized Accident scenario.

Second iteration, with a more strict prompt

```
diff --git a/srunner/scenarios/route_obstacles.py b/srunner/scenarios/route_obstacles.py
index 8a0a3dc..c37c080 100644
--- a/srunner/scenarios/route_obstacles.py
+++ b/srunner/scenarios/route_obstacles.py
@@ -84,6 +84,11 @@ class Accident(BasicScenario):
```

```
6           self._max_speed = get_value_parameter(config, 'speed', float, 60)
7           self._scenario_timeout = 240
8
9   +       # Additional parameters for enhancing complexity and jerk
10  +       self._creep_distance = get_value_parameter(config, 'creep_distance', float, 2.0)
11  +       self._creep_speed = get_value_parameter(config, 'creep_speed', float, 1.0) # m/s (3.6
        km/h)
12  +       self._lateral_creep_offset = get_value_parameter(config, 'lateral_creep_offset',
        float, 0.5) # meters
13  +
14          super().__init__(
15              "Accident", ego_vehicles, config, world, randomize, debug_mode, criteria_enable=
        criteria_enable)
16
17  @@ -199,15 +204,57 @@ class Accident(BasicScenario):
18          end_condition.add_child(ScenarioTimeout(self._scenario_timeout, self.config.name))
19          end_condition.add_child(WaitUntilInFrontPosition(self.ego_vehicles[0], self._end_wp.
        transform, False))
20
21  -       behavior = py_trees.composites.Sequence()
22  -       behavior.add_child(InTriggerDistanceToLocation(
23  +       # Ego's main behavior sequence: approach, idle, then wait
24  +       ego_behavior = py_trees.composites.Sequence()
25  +       ego_behavior.add_child(InTriggerDistanceToLocation(
26              self.ego_vehicles[0], self._first_vehicle_wp.transform.location, self.
        _trigger_distance))
27  -       behavior.add_child(Idle(self._wait_duration))
28  +       ego_behavior.add_child(Idle(self._wait_duration))
29          if self.route_mode:
30  -           behavior.add_child(SetMaxSpeed(self._max_speed))
31  -       behavior.add_child(WaitForever())
32  +           ego_behavior.add_child(SetMaxSpeed(self._max_speed))
33  +       ego_behavior.add_child(WaitForever())
34  +
35  +       end_condition.add_child(ego_behavior)
36  +
37  +       # NEW: Behavior for the first accident vehicle to creep forward and laterally
38  +       # This will introduce sudden changes in the environment, leading to increased jerk
        for the ego.
39  +       first_actor = self.other_actors[2] # Based on the order of actor initialization
40  +
41  +       # Calculate target waypoint for longitudinal creep
42  +       first_actor_start_wp = self._map.get_waypoint(first_actor.get_location())
43  +       creep_target_wp_forward = self._move_waypoint_forward(first_actor_start_wp, self.
        _creep_distance)
44  +
45  +       # Calculate the lateral offset for the BasicAgentBehavior.
46  +       # This offset is relative to the center of the target waypoint's lane.
47  +       # Initial lateral position of the obstacle (from lane center, positive to right).
48  +       initial_lateral_offset_from_center = self._offset * first_actor_start_wp.lane_width /
         2
49  +       if self._direction == 'left': # If original placement was on the left, make this
        negative
50  +           initial_lateral_offset_from_center *= -1
51  +
52  +       # Desired change to increase blockage and induce jerk.
53  +       # If ego passes left (direction 'right'), obstacle is on right. To block more, move
        further right (increase offset).
54  +       # If ego passes right (direction 'left'), obstacle is on left. To block more, move
        further left (decrease offset).
55  +       target_lateral_offset = initial_lateral_offset_from_center
56  +       if self._direction == 'right':
57  +           target_lateral_offset += self._lateral_creep_offset
58  +       else: # self._direction == 'left'
59  +           target_lateral_offset -= self._lateral_creep_offset
60  +
61  +       opt_dict_creep = {'offset': target_lateral_offset}
62  +
63  +       creep_behavior = py_trees.composites.Sequence(name="First Accident Car Creep")
64  +       # Trigger the creep behavior when ego is closer, after initial detection
65  +       creep_behavior.add_child(InTriggerDistanceToLocation(
66  +           self.ego_vehicles[0], self._first_vehicle_wp.transform.location, self.
        _trigger_distance / 2))
67  +       creep_behavior.add_child(Idle(self._wait_duration / 2)) # Short idle to ensure ego is
         committed
68  +       creep_behavior.add_child(HandBrakeVehicle(first_actor, False)) # Release handbrake
69  +       # Make the actor creep forward and laterally with BasicAgentBehavior
70  +       creep_behavior.add_child(BasicAgentBehavior(
71  +           first_actor, creep_target_wp_forward.transform.location, target_speed=self.
        _creep_speed, opt_dict=opt_dict_creep))
72  +       creep_behavior.add_child(HandBrakeVehicle(first_actor, True)) # Re-apply handbrake
        after moving
73  +       creep_behavior.add_child(WaitForever()) # Keep this behavior alive until scenario
        ends
```

```
74 +
75 +         end_condition.add_child(creep_behavior) # Add this parallel behavior to the scenario
76
77 -         end_condition.add_child(behavior)
78           root.add_child(end_condition)
79
80           if self.route_mode:
```

Listing A.5: The diff of the jerk-optimized Accident scenario with stricter specification.

Appendix B

Scenario error messages

This appendix contains a collection of error messages from running scenarios on the Carla simulator. For brevity they are not included in the running text of the thesis.

B.1 Follow vehicle

B.1.1 Initially enhanced scenario

```
1    oliverrj@pensie7:~/Documents/master-extern/scenario_runner-0.9.15|hakcs   python3.7
      scenario_runner.py --scenario FollowLeadingVehicle_1 --reloadWorld --record logs
2  Preparing scenario: FollowLeadingVehicle_1
3  The scenario cannot be loaded
4  Traceback (most recent call last):
5    File "scenario_runner.py", line 406, in _load_and_run_scenario
6      scenario_class = self._get_scenario_class_or_fail(config.type)
7    File "scenario_runner.py", line 157, in _get_scenario_class_or_fail
8      scenario_module = importlib.import_module(module_name)
9    File "/usr/local/lib/python3.7/importlib/__init__.py", line 127, in import_module
10     return _bootstrap._gcd_import(name[level:], package, level)
11   File "<frozen importlib._bootstrap>", line 1006, in _gcd_import
12   File "<frozen importlib._bootstrap>", line 983, in _find_and_load
13   File "<frozen importlib._bootstrap>", line 967, in _find_and_load_unlocked
14   File "<frozen importlib._bootstrap>", line 677, in _load_unlocked
15   File "<frozen importlib._bootstrap_external>", line 728, in exec_module
16   File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
17   File "/home/oliverrj/Documents/master-extern/scenario_runner-0.9.15/srunner/scenarios/
       follow_leading_vehicle.py", line 26, in <module>
18     from srunner.scenariomanager.scenarioatomics.atomic_behaviors import (ActorTransformSetter
       ,
19 ImportError: cannot import name 'PedestrianWalk' from 'srunner.scenariomanager.scenarioatomics
       .atomic_behaviors' (/home/oliverrj/Documents/master-extern/scenario_runner-0.9.15/
       srunner/scenariomanager/scenarioatomics/atomic_behaviors.py)
20 cannot import name 'PedestrianWalk' from 'srunner.scenariomanager.scenarioatomics.
       atomic_behaviors' (/home/oliverrj/Documents/master-extern/scenario_runner-0.9.15/srunner
       /scenariomanager/scenarioatomics/atomic_behaviors.py)
21 No more scenarios .... Exiting
```

Listing B.1: Error message when running the the initially enhanced FollowLeadingVehicle scenario with halluciantion.

The scenario that triggered this error is presented in Section 6.1.1 →p.39 and the contents of the LLM enhancement are presented in listing A.1.

```
oliverrj@pensie7:~/Documents/master-extern/scenario_runner-0.9.15|hakcs   python3.7
     scenario_runner.py --scenario FollowLeadingVehicle_1 --reloadWorld --record logs
Preparing scenario: FollowLeadingVehicle_1
The scenario cannot be loaded
Traceback (most recent call last):
  File "scenario_runner.py", line 406, in _load_and_run_scenario
    scenario_class = self._get_scenario_class_or_fail(config.type)
  File "scenario_runner.py", line 157, in _get_scenario_class_or_fail
    scenario_module = importlib.import_module(module_name)
  File "/usr/local/lib/python3.7/importlib/__init__.py", line 127, in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
  File "<frozen importlib._bootstrap>", line 1006, in _gcd_import
  File "<frozen importlib._bootstrap>", line 983, in _find_and_load
  File "<frozen importlib._bootstrap>", line 967, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 677, in _load_unlocked
  File "<frozen importlib._bootstrap_external>", line 728, in exec_module
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "/home/oliverrj/Documents/master-extern/scenario_runner-0.9.15/srunner/scenarios/
     follow_leading_vehicle.py", line 31, in <module>
    from srunner.scenariomanager.scenarioatomics.atomic_behaviors import (ActorTransformSetter
     ,
ImportError: cannot import name 'WalkToLocation' from 'srunner.scenariomanager.scenarioatomics
     .atomic_behaviors' (/home/oliverrj/Documents/master-extern/scenario_runner-0.9.15/
     srunner/scenariomanager/scenarioatomics/atomic_behaviors.py)
cannot import name 'WalkToLocation' from 'srunner.scenariomanager.scenarioatomics.
     atomic_behaviors' (/home/oliverrj/Documents/master-extern/scenario_runner-0.9.15/srunner
     /scenariomanager/scenarioatomics/atomic_behaviors.py)
No more scenarios .... Exiting
```

Listing B.2: Error message when running the the strictly enhanced FollowLeadingVehicle scenario with halluciantion.

Glossary

ADCP	Autonomous driving comfort prediction
ADS	Autonomous driving system
AI	Artificial intelligence
API	Application programming interface
CoT	Chain-of-Thought
DSC	Driving situation complexity
DSL	Domain specific language
FOSS	Free and open source
GUI	Graphical user interface
ICL	In-context learning
JIT	Just-In-Time
K8s	Kubernetes
LLM	Large Language Model
MCP	Model context protocol
ML	Machine learning
NLP	Natural language processing
OAI	OpenAI
RAG	Retrieval augmented generation
RL	Reinforcement learning
RPC	Remote procedure call