UNIVERSITY
OF OSLO

Master's thesis

Can AI make our roads safer?

Utilising Large language models to enhance Autonomous driving system test cases

Oliver Ruste Jahren

Informatics: Programming and Systems architecture
60 ECTS study points

Department of Informatics
Faculty of Mathematics and Natural Sciences

Fall 2025

Oliver Ruste Jahren

Can AI make our roads safer?

Utilising Large language models to enhance
Autonomous driving system test cases

Supervisor:
Shaukat Ali
Karoline Nylænder

Simula Research Laboratory

Abstract

Here come 3–6 sentences describing your thesis.

Sammendrag

Here comes the abstract in a different language.

Contents

List of Figures

Preface

Here comes your preface, including acknowledgments and thanks.

Part I

Introduction

Chapter 1

Motivation

Conventional cars are ubiquitous in society. 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. 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 efficiency will increase and road fatalities will fall.

Due to the critical safety situation of manoeuvring a car In a public setting where other external actors are present, it is essential that Autonomous driving systems are thoroughly tested before they are deployed so that they are confirmed to be sufficiently safe and capable of handling the situations in which they may typically end up. But due to the complicated nature of the typical ADS operating environment, coming up with exhaustive system test solutions is near impossible. For this reason we want a way of testing the system that is capable of pushing the Autonomous driving system to its limits such that we can measure its performance and see if it is capable of handling complex scenarios.

Having an existing repository of Autonomous driving system test cases, such as DeepScenario we wish to improve them. **Large Language Models (LLMs)** have demonstrated great capabilities of context learning and emergent abilities, which begs the question of their applicability for ADS testing. There are various methods of testing Autonomous driving systems. Can these existing test methods be improved by applying LLM technology to them?

Chapter 2

Background

2.1 Testing

First, we need to establish some basic testing concepts.

2.1.1 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.

2.1.2 Test coverage

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 [27, 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 [27, p. 187]. A test case that exercised every single code line of the system would by definition have perfect test coverage.

2.2 Autonomous driving systems (ADSs)

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 systems maintain a high safety standard. A common way to assert safety is to use simulator based testing [25, p. 1].

2.2.1 Autonomous driving system testing

Testing is essential for assuring Autonomous driving system operative safety [16, p. 163]. Several methods for testing exist, testing various aspects of the Autonomous driving system. An ADS typically exists of several modules, all working together and handling different aspect of the Autonomous driving system.

Huang et al. 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* [16, pp. 163–164].

2.2.2 Autonomous driving system driveability

Driveability is a high-level estimator of the overall driving condition of an ADS, derived from several lower-level sources [15, 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 Section 2.2.7 → p. 8, 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 [15, pp. 3135–3136].

The main method for the assessing driveability of a scene comes form assessing the environment of the scene. Factors such as (1) weather, (2) traffic flow, (3) road condition, (4) obstacles all play into this. The ADS infers information from observation [15, 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 [15, 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) [15, 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 [15, 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 [15, p. 3137].

Implicit driveability factors consist of behaviours and intent of other road users interacting with the autonomous car [15, 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** [15, 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 [15, 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 [15, 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 [15, 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 [15, p. 3139], causing potential issues for any ADS that wishes to operate in a shared traffic environment with this group.

2.2.3 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 2.2.2 → p. 6), 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.

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 [15, 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 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 [15, 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 [15, p. 3140]. Furthermore Guo, Kurup, and Shah describe how this metric also allows more than one ADSs to be labelled as 'conflicting' or 'not conflicting' [15, 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.

2.2.4 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*[1] 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* [28, 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 [28, p. 1182]. Their formula also takes into account the ratio of *V2X*-capable vehicles [28, p. 1182], i.e. the vehicles that are connected and capable of communicating [36, p. 1].

2.2.5 Autonomous driving system simulation

Due to the complexity involved in testing Autonomous driving systems (Section 2.2.4 → p. 7), simulators are typically used for this purpose [25]. While the same

[1] See Test coverage → p. 5

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.

2.2.6 The ADS simulator jungle

Due to the appeal of running ADS simulation, several contenders exist on the market.

Carla is a widely used ADS simulator [11]. It is implemented using the game engine UnrealEngine [12] 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 applied in projects such as KITTI-Carla, which generated a KITTI dataset using Carla [9].

LGSVL is a deprecated simulator from LG [30]. It was used in projects such as DeepScenario [25]. It allowed for running various maps with various vehicles and tracking their data. It was also capable of generating HD maps [2]. DeepScenario is a project similar to this, concerned with testing Autonomous driving systems. Further details about it in are located in Related work → p. 21.

AirSim is Microsoft's offering [32]. 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.

2.2.7 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 [34, 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* [34, 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, which are to be considered by a robot for its selection of an appropriate behaviour pattern in a particular moment'*[3], in Wershofen and Graefe [38, 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 [31, 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* [34, p. 985].

scenario refers to *'the temporal development between several scenes in a sequence of scenes'*[34, 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*

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

[3] The translation from German is borrowed from Ulbrich et al., [34, p. 984]

this temporal development in a scenario' [34, 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* [14].

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 [6, 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 [6, p. 652]. The simulator Carla (outlined in Section 2.2.6 → p. 8) supports this standard [6, p. 652].

Another widely popular scenario format is **CommonRoad** [23, 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 [23, p. 4941].

2.3 Large Language Models (LLMs)

Large Language Models (LLMs) are transformer-based language models that typically contain several hundred billion parameters and are trained on massive text data [40, p. 4]. Base language models, as the name implies, *model language*. They are typically statistical models and an example of Machine learning (ML).

2.3.1 Large Language Model (LLM) architecture

A Large Language Model is a neural network trained on big data [40, 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 [40, p. 3] (Emergent abilities → p. 10). 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 [35] 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 [40, p. 9]

Jurafsky and Martin describe how LLMs rely on *pretraining*.

The importance of training data

As a consequence of LLMs being statistical models of a certain input data [40, p. 1], what data the model is trained on is of great importance for the capabilities of the model [40, p. 6]. Zhao et al. give an overview of various LLMs and what kinds of corpora[4] they have been trained on [40, 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

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

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. 12 for further information about hallucination.

2.3.2 Emergent abilities

Wei et al. outline how *emergent abilities* appear when scaling up language models [37, p. 1]. They define *emergent ability* to refer to abilities that are not present in smaller models, but present in the larger ones[37, p. 1], building on physicist Anderson stating that *Emergence is when quantitative changes in a system result in qualitative changes in behaviour.* [37, 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 [37, 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 [37, p. 5].

2.3.3 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.

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 2.3.5 → p. 12).

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 [13, 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. 11)

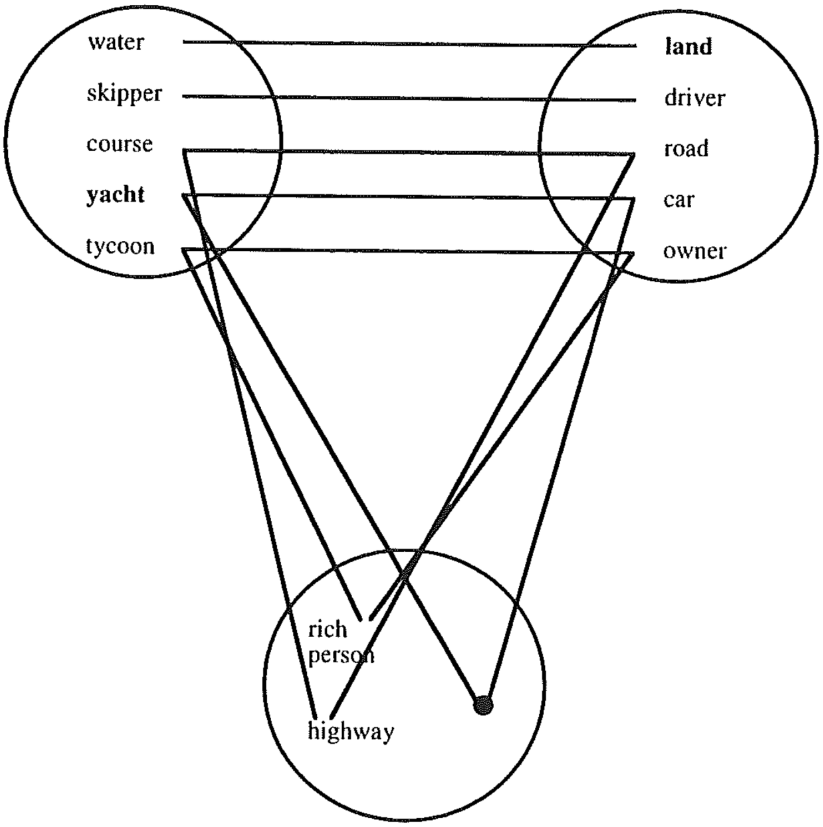

Figure 2.1: The conceptual blend of a *land yacht*[5]

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

2.3.4 Utilising LLMs - Prompt engineering

A typical way of interacting with LLMs is *prompting* [40, p. 44]. You prompt the model to solve various tasks. As we saw in Emergent abilities → p. 10, 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** [40, 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 [40, p. 44]. It is based on *tuning-free prompting* and it, as the name implies, never tunes the parameters of the LLM [24, p. 15]. One the one hand, this allows 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 [24, 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 [40, pp. 44, 52]. The

[5]Diagram borrowed from Fauconnier and Turner, [13, p. 67].

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 [40, 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 [40, 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 [40, 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 [40, pp. 54–55].

2.3.5 General challenges with LLMs

We have seen that LLMs demonstrate promising abilities (Emergent abilities →p.10) But they have nevertheless certain issues attached to them that we need to be aware of.

Hallucination

As we saw in Section 2.3.3 →p.10, 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 [17]. 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 [17].

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 [33, 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 [22, 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 Denmarks, or half of the United Kingdom [22, p. 1]. Recent research indicates that *serving* LLMs currently account for more emissions than training them [10, 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 [21, p. 21799].

2.3.6 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 in Section 2.3.1 →p.9, 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 2.3.3 → p. 10).

2.3.7 Existing LLM applications for ADSs

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 [8, 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 ADSs [7, p. 1].

Chapter 3

Problem description

3.1 Cost

3.2 Edge cases

3.3 Impossible to test all scenarios

Chapter 4

Literature review

TODO: Write literature review
Can move some things from related work such as LLM4AD?

Part II

The project

Chapter 5

Related work

5.1 DeepScenario

DeepScenario is both a dataset and a toolset aimed at Autonomous driving system testing [25]. 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 [25, p. 52]. The authors therefore propose an open driving scenario of more than 30 000 driving scenarios focusing on ADS testing [25, 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 2.2.6 → p. 8).

5.2 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 [39, 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 [39, p. 397]. The proposed tools were evaluated in experiments with industry partners, successfully generating executable test cases [39, p. 397].

5.3 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 [26, 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 envisioned project.

5.4 AutoSceneGen

AutoSceneGen is a framework for ADS testing using LLMs, focusing on the motion planning of Autonomous driving systems [1, p. 14539]. Aiersilan highlights how LLMs provide opportunities for efficiently evaluating ADSs 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].

5.5 LLM4AD

LLM4AD is a paper that gives a broad overview of LLMs for Autonomous driving systems. 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 [8, p. 2]. Furthermore, the authors also propose a comprehensive benchmark for evaluating the instruction-following abilities of an LLM based system in ADS simulation [8, p. 1].

5.6 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 [29, 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 [29, p. 173].

5.7 Requirements All You Need?

Lebioda et al. provide an overview of LLMs for ADSs in their recent preprint *Are requirements really all you need? A case study of LLM-driven configuration code generation for automotive simulations*[1], focusing on LLM's abilities for translating abstract requirements extracted from automotive standards and documents into configuration for Carla (Section 2.2.6 → p. 8) simulations [20]. 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 (Pre- and post-conditions → p. 5) [20]. 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 [20].

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

Chapter 6

Proposed solution

We have seen that ADS testing is *complex* and that it is difficult to get a good test coverage (Section 2.2.4 → p. 7). Furthermore, we have seen that LLMs have *emergent abilities* (Section 2.3.2 → p. 10). We therefore propose 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.

This will allow us to learn the extent to which LLMs can be applied for enhancing Autonomous driving system test cases. We will survey several LLMs and evaluate their applicability for the problem at hand, in light of what we know about LLMs (The different kinds of LLMs → p. 12). We want to have a pipeline that is able to process several test cases in succession, in order to get a substantial dataset.

Let the pipeline tool be known as HEFE. The tool follows a natural pipeline structure. We have some base test cases that need to be ran in order to get a baseline for the results, we then have to improve these, and run the improved versions and compare them to their original versions. The architecture of the tool is visualised in Figure 6.1 → p. 24.

We need to define what requirement we will use for determining the *result* of a test case run. Without this, we cannot compare it to other test cases.

Furthermore, as outlined in Cui et al., Large Language Models can be applied to several aspects of Autonomous driving systems. It is not feasible that we focus on *all* these aspects, and as such we should narrow down our scope. Let us review some of the relevant aspects.

The applicability of LLMs in ADS testing

Autonomous driving systems are typically modular, as we have seen in Section 2.2.4 → p. 7. LLMs are applicable to the different modules in different ways as we saw in Related work → p. 21.

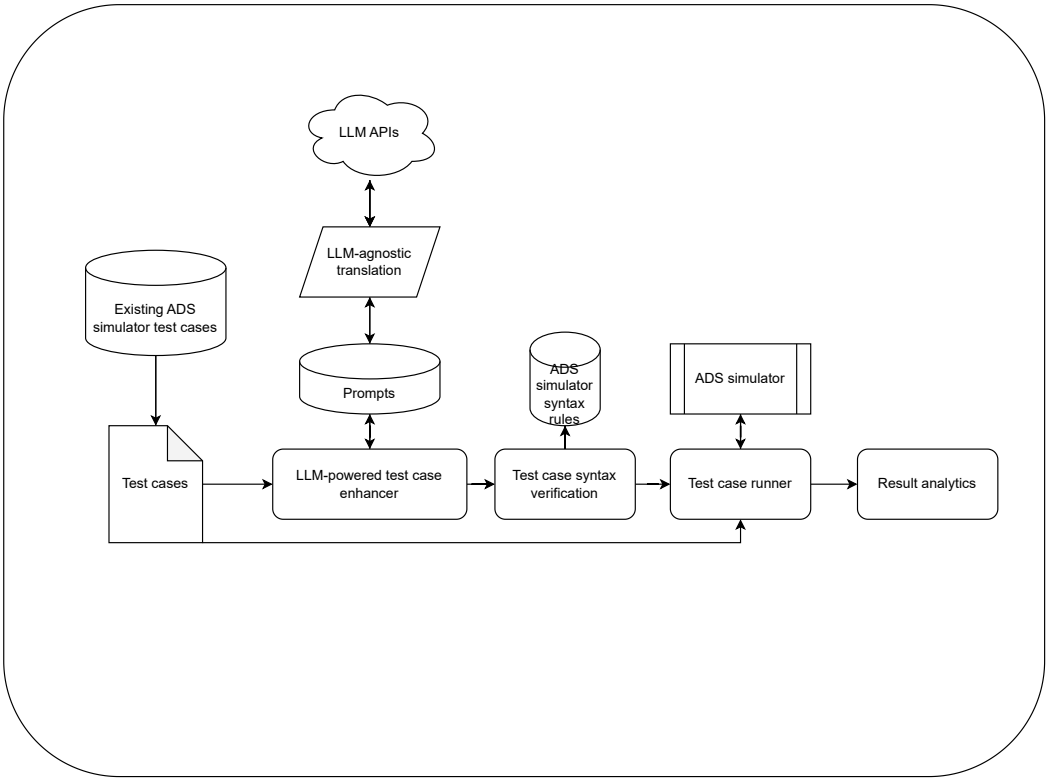

Figure 6.1: Hefe pipeline architecture

User history of using Hefe

I have a set of Autonomous driving system (ADS) test cases. I provide this set to Hefe. It will run the entire set, and generate a baseline of my ADS performance. Hefe will then improve my test cases using Large Language Models and run them again.
Lastly Hefe will report how the results differ from running the base and enhanced version of a test case.
This will give me insight into what caused my ADS to fail so that I can look into the cause of the error state and uncover underlying faults in the Autonomous driving system.

6.1 Implementation language

The programming language Python is widely used for Autonomous driving system (ADS) simulation. It is a high level language, allowing the user great flexibility and developer experience. For this reason, I will implement Hefe using Python.

Python can be optimized using Just-In-Time (JIT) compilers such as Numba [19], 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.

6.1.1 The room for concurrency

When evaluating ADS test cases, the test cases are independent of each other. This means that our problem is *embarrassingly parallelizable* [1] and we can trivially process several test cases in parallel. Due to practical limitations in Carla, *running* the test cases should however probably be done sequentially. But (1) prompting, (2) enhancing, and (3) validating, can all be done concurrently. While Python lacks support of traditional threads, it has some support for multiprocessing [2].

6.2 Overview of the components of the HEFE pipeline

The pipeline architecture is visualised in HEFE pipeline architecture → p. 24. Here we present the major components and their responsibilities

6.2.1 Test case enhancement

Test case repositories

We have seen in Related work → p. 21 that there are existing repositories of ADS test cases. These will provide us with (1) a baseline, and (2) data onto which we can apply our LLM enhancements.

LLM enhancement

The base test cases will individually be enhanced by prompting the LLM. We will experiment with several LLMs.

For performing the actual improvement, it is essential that we (1) test several LLM, (2) give clear prompts and (3) verify that the returned test case adheres to the strictly necessary syntax rules. This last point is important due to our knowledge of LLMs hallucinating (see General challenges with LLMs → p. 12).

In order to facilitate testing various Large Language Models, we should employ LLM agnostic software as a translation layer. This will allow us to write code for a common interface and test several LLMs that may all have different internal Application programming interfaces (APIs) without having to modify our test code for specific APIs. This (1) saves time and (2) makes for more even test conditions . Some pieces of software providing this type of functionality include AISUITE[3], RamaLama from RedHat[4], and the MIT licensed Ollama[5], both supporting a plethora of Large Language Models.

GUIDANCE[6] is a framework for limiting the room in which LLMs may operate, which might be useful if we run into issues with excessive hallucination.

Enhanced test case validation

We must expect the LLM to hallucinate to some extent (Section 2.3.5 → p. 12). We therefore propose to verify the format of the enhanced file before running it.

[1] https://en.wikipedia.org/wiki/Embarrassingly_parallel
[2] https://docs.python.org/3/library/multiprocessing.html
[3] https://github.com/andrewyng/aisuite
[4] https://github.com/containers/ramalama
[5] https://github.com/ollama/ollama
[6] https://github.com/guidance-ai/guidance

As we saw in the section for ADS scenario formats → p. 9, there exists several formats for ADS scenarios. In order to verify that the syntax of our enhanced test case is valid, we simply need to apply the syntax rules of our format.

The CommonRoad format is XML-based [2, p. 720] and as such we can to some extent assess the degree of hallucination by parsing the XML structure. Furthermore, it has an exhaustive Python library with several utilities[7].

OpenSCENARIO exists both as XML and a domain-specific language (DSL). If we utilise the XML version, we can apply the same methodology as for the CommonRoad format. If using the DSL version, one way the OpenSCENARIO format can be verified is by using free online cloud services such as this offering from AVL [8]. We should however strive for running a local verification service to (1) save time and compute, and (2) preserve data privacy. Besides, it is generally a good idea to limit the number of external dependencies[9].

6.2.2 Test case running and evaluation

Test case runner

The system will automatically run all our base test cases using an ADS simulator, and collect data points to get a baseline. It will later also run the mutated LLM-enhanced versions of the base cases.

We have already ran the test cases in their base form. We will now run their improved versions in order to compare them to see what effect the LLM enhancement (see Section 6.2.1 → p. 25) has had.

For the reasons we have seen in Section 2.2.6 → p. 8, we want to run our test cases on Carla. It is the best offering as it is open source, under active development and has a feature rich Python API.

Test case improvement evaluation

We saw in Section 2.2.3 → p. 7 that there are several metrics for assessing ADSs. We will use these metrics when evaluating our improvements.

Test case result reporting

We will 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. Ideally with some automatic analysis of the results.

Having ran both the base test case and its enhanced counterpart, we have results. The results will be stored in Comma separated values (CSV) files, allowing (1) further analysis in Python/Jupyter, and (2) easy translation to LaTeXtables for the final report.

This is the final step of the envisioned pipeline. Where we have our result, and need to analyse them.

This last step has great opportunities for being scoped up to a fully integrated test suite which allows for both running test cases and analysing the results in a Graphical user interface (GUI). But we should focus on the prior steps for now, only creating a

[7] https://pypi.org/user/commonroad/

[8] https://smc.app.avl.com/validation

[9] Note for example how LGSVL[30] was shut down, preventing projects such as DeepScenario of Lu, Yue, and Ali to be further developed on the original platform.

GUI if there is sufficient time towards the end of the project to focus on such non-LLM related topics.

Initially, the results will consist of numerical comparison of the CSVs with regard to the relevant metrics outlined in Test case improvement evaluation → p. 26.

Chapter 7

Implementation architecture

Chapter 8

Implementation details 1 - Thor

Rest API - running test cases - Thor “Fast API”, Python POST a test case to the API. It will be ran on the ‘server’ RabbitMQ for listening for finished test cases? So that the client knows it can fetch the results? Will need UUID for test cases so the correct result can be fetched after it has been ran Need to store these somewhere. NoSQL database? This component should also accumulate results. Huge TODO: What metric are these results? Should be containerised (Docker/Podman)

Chapter 9

Implementation details 2 - Odin

Rest API - performing LLM enhancement - Odin “Fast API”, Python Take a base test case as body Have some prompt repository Apply prompts with LLMs Must integrate with LLM. Either locally (Ollama) or remote (some API) Look into good LLM agnostic transition layer. E.g. Aisuite https://github.com/andrewyng/aisuite Should use same UUIDs as outlined above, but suffixed with e.g. “pure” and “tainted” Containerized. Docker compose?

Chapter 10

Implementation details 3 - Loki

Client - orchestrating the process - Loki Fetch available test cases from Thor? Select what/which are to be used Store results clientside? Separate database for this?

Part III

Conclusion

Chapter 11

Experiment methodology

Chapter 12

Results

Chapter 13

Discussion

13.1 Environmental concerns

Cost/benefit with using LLMs. Refer back to General challenges with LLMs → p. 12.

Chapter 14

Further work

14.1 Other LLMs

14.2 Different promtping strategies

14.3 GUI visualisations

Maybe: Frontend client - web GUI - Ivar If Loki does its job effectively, we can create a web based frontend for doing the process. It could do the same as Loki, but with greater ease of use. Having a GUI allows for making neat visualisations. Motivate why our enhanced test cases are better by showing it.

Bibliography

[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.

[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.

[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.

[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.

[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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[12] Epic Games. *Unreal Engine*. Version 4.22.1. Apr. 25, 2019. URL: https://www.unrealengine.com.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

[19] 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.

[20] 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.

[21] 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/.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] Guodong Rong et al. “LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving”. In: *arXiv preprint arXiv:2005.03778* (2020).

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] Jason Wei et al. *Emergent Abilities of Large Language Models*. 2022. arXiv: 2206.07682 [cs.CL]. URL: https://arxiv.org/abs/2206.07682.

[38] 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.

[39] 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.

[40] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2025. arXiv: 2303.18223 [cs.CL]. URL: https://arxiv.org/abs/2303.18223.