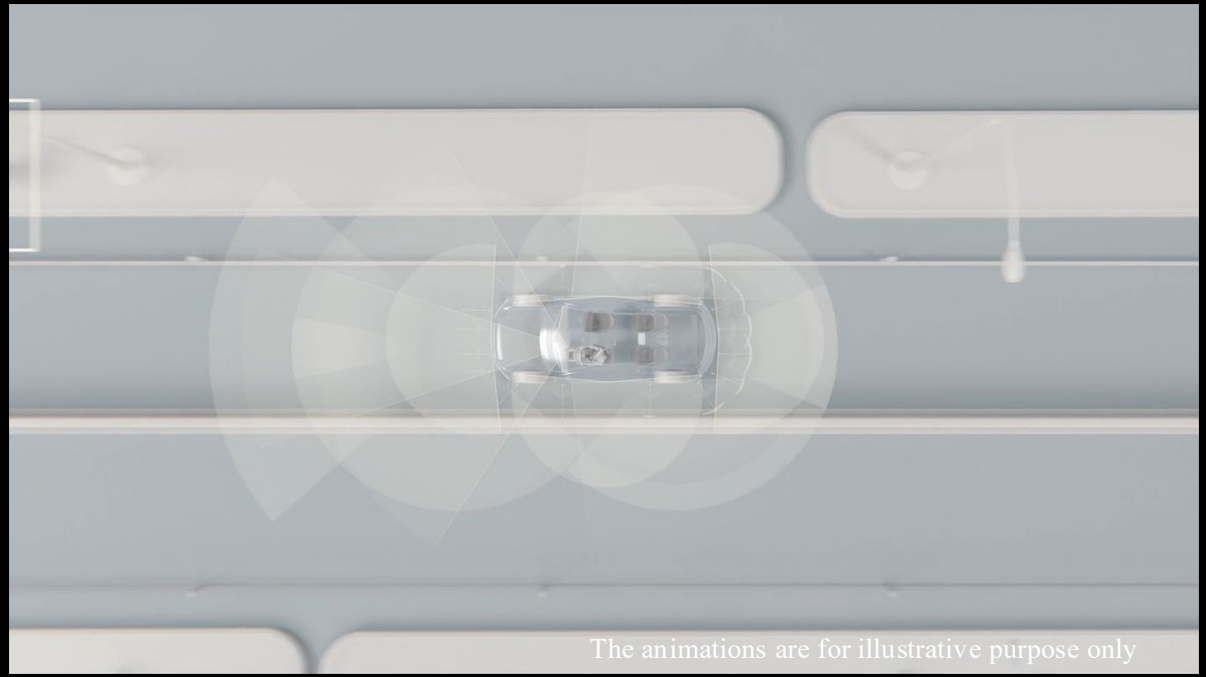
AI-enabled DevSafeOps for Autonomous Driving Software

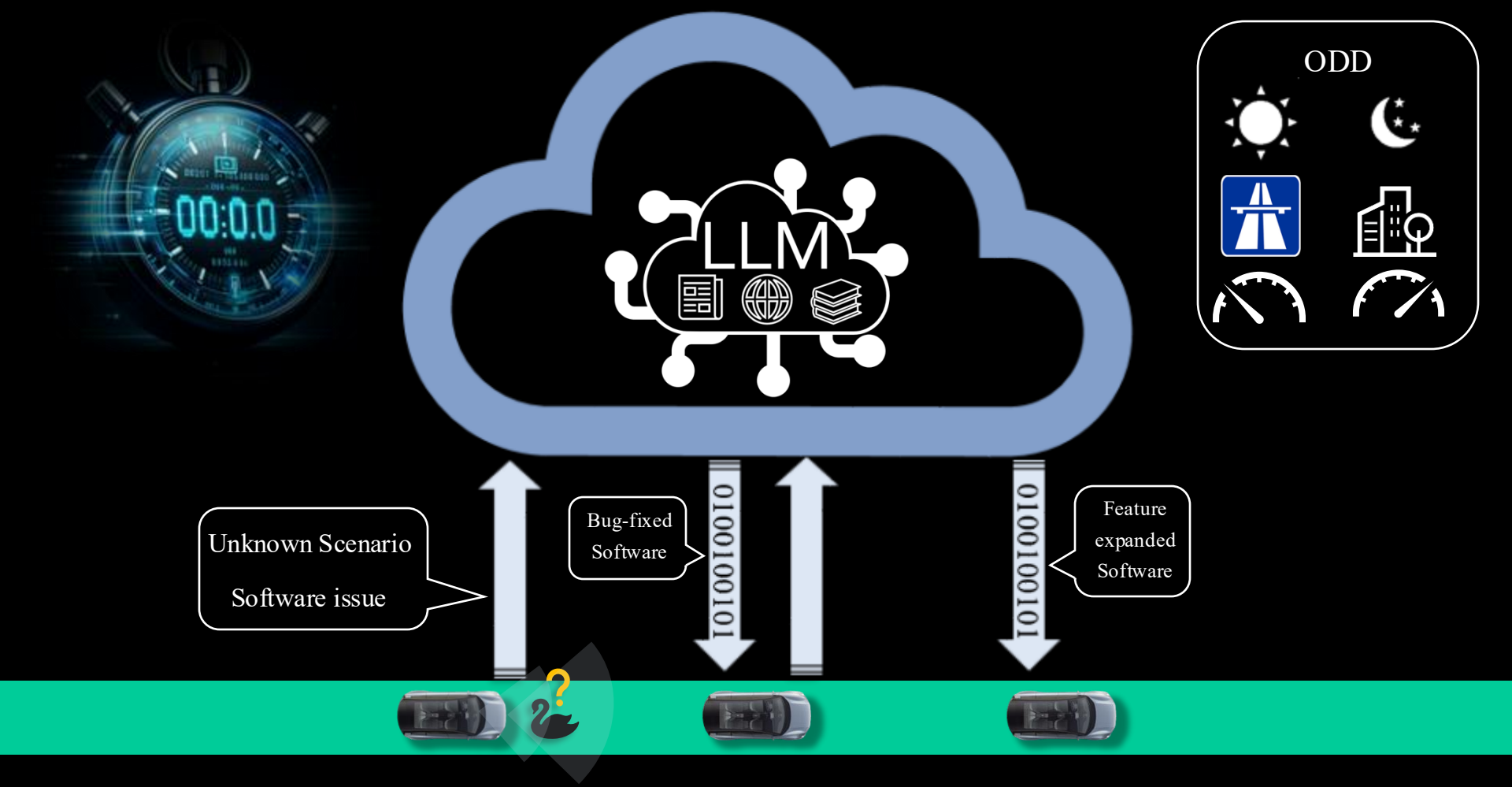
Ali Nouri

ASSERTED - Assuring Safety for Rapid and Continuous Deployment for AD

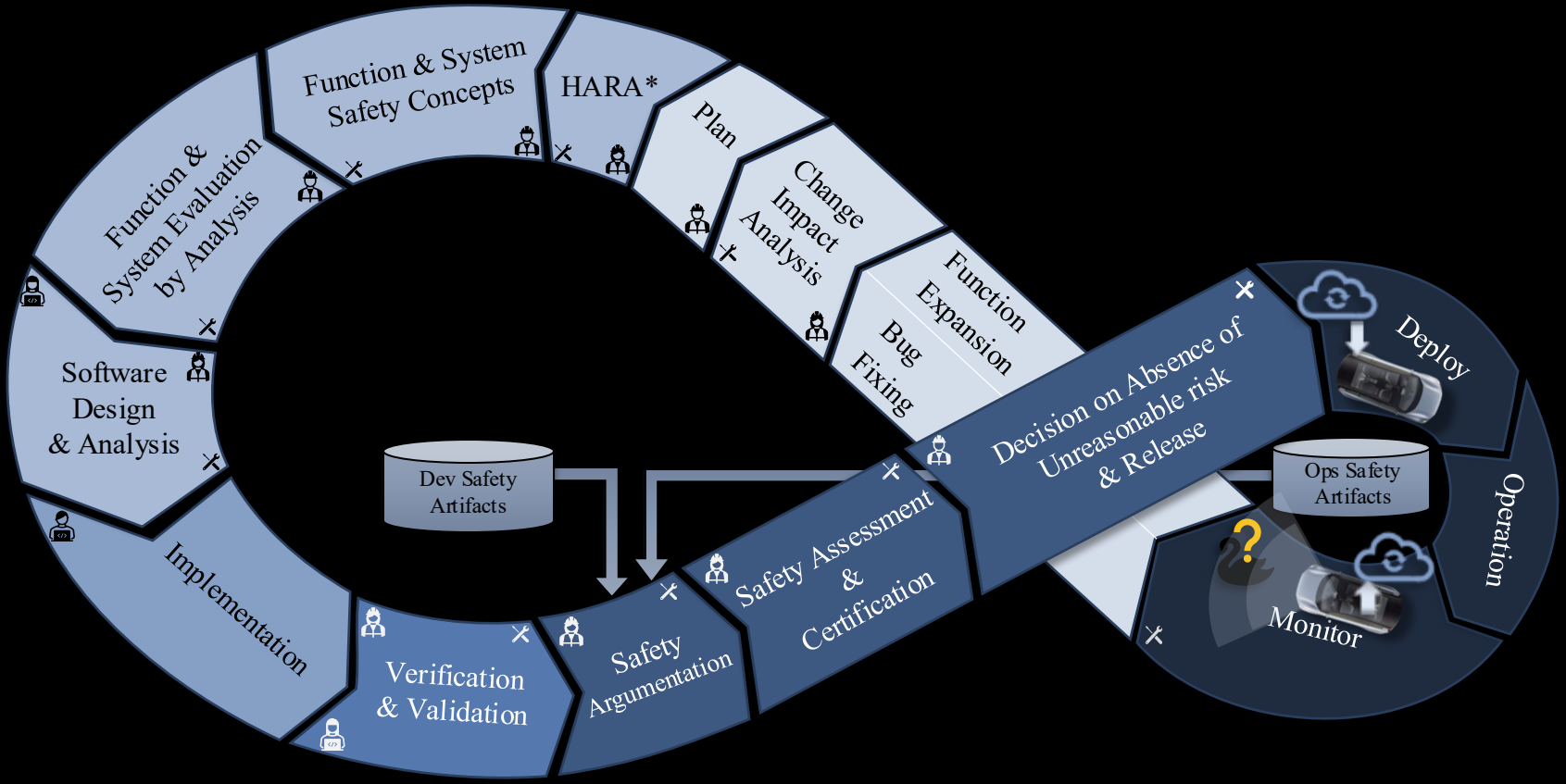


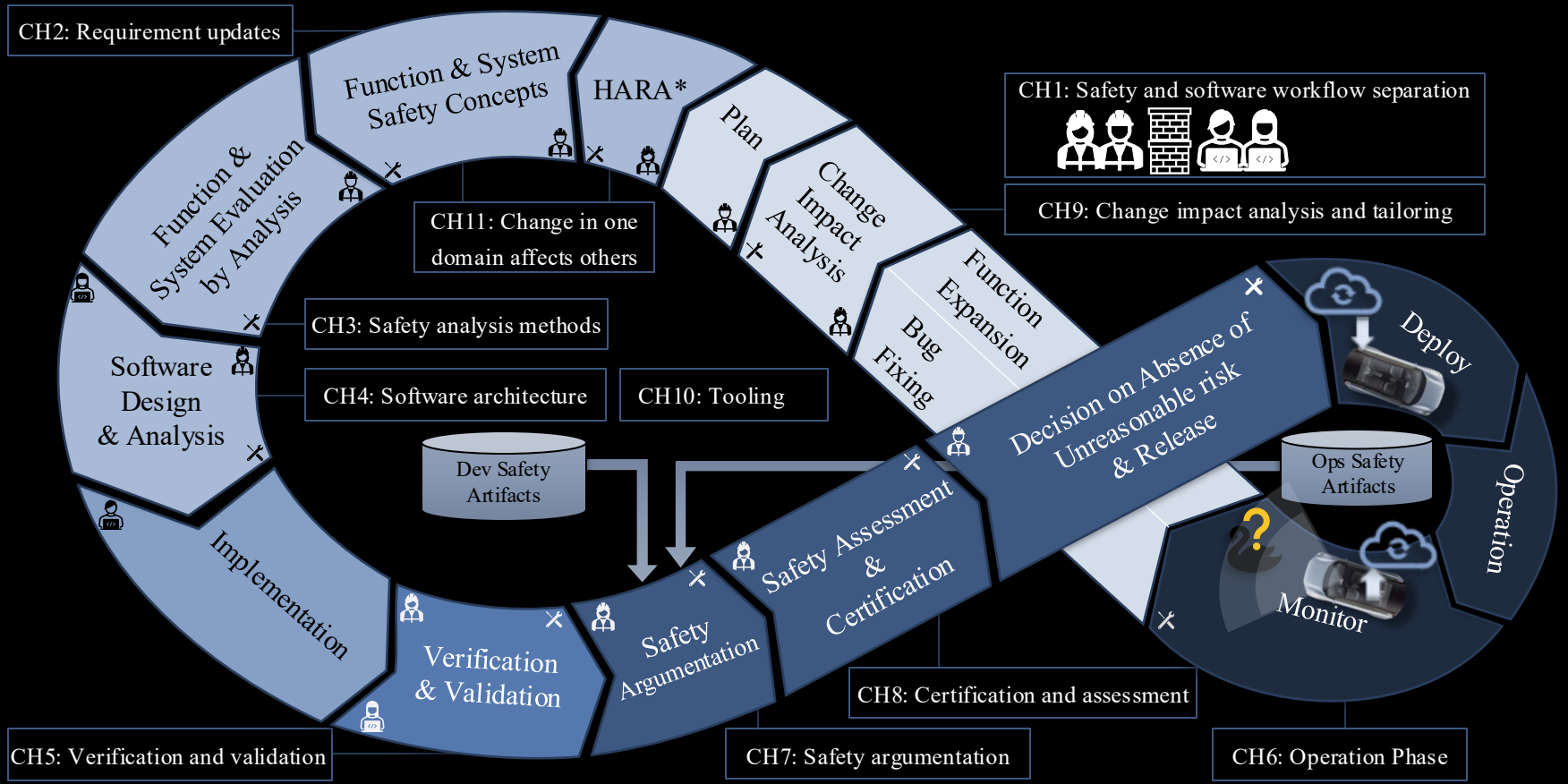
Diarienummer: 2021-02585

Goal


ODD
LLM
Unknown Scenario
Software issue
Bug-fixed
Software
Feature
expanded
Software
01001001010
01001001010

Safe


Function & System Safety Concepts
HARA*
Plan
Change Impact Analysis
Function Expansion
Bug Fixing
Decision on Absence of Unreasonable risk & Release
Deploy
Operation
Monitor
Ops Safety Artifacts
Safety Assessment & Certification
Safety Argumentation
Verification & Validation
Implementation
Software Design & Analysis
Function & System Evaluation by Analysis
Dev Safety Artifacts


CH2: Requirement updates
Function & System Safety Concepts
HARA*
Plan
Change Impact Analysis
Function Expansion
Bug Fixing
CH1: Safety and software workflow separation
CH9: Change impact analysis and tailoring
CH11: Change in one domain affects others
Function & System Evaluation by Analysis
CH3: Safety analysis methods
Software Design & Analysis
CH4: Software architecture
CH10: Tooling
Dev Safety Artifacts
Decision on Absence of Unreasonable risk & Release
Deploy
Ops Safety Artifacts
Operation
Monitor
Safety Assessment & Certification
Safety Argumentation
Verification & Validation
Implementation
CH8: Certification and assessment
CH5: Verification and validation
CH7: Safety argumentation
CH6: Operation Phase

CH2: Require

The Journal of Systems and Software 230 (2025) 112555

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

The DevSafeOps dilemma: A systematic literature review on rapidity in safe autonomous driving development and operation ☆

Ali Nouri [a,c,*], Beatriz Cabrero-Daniel [a,b], Fredrik Törner [c], Christian Berger [a,b]

[a] Chalmers University of Technology, Department of Computer Science, Gothenburg, Sweden
[b] University of Gothenburg, Department of Computer Science, Gothenburg, Sweden
[c] Volvo Cars, Gothenburg, Sweden

ARTICLE INFO

Keywords:
Continuous development
Safety-related function
Autonomous driving
Safety of the intended function (SOTIF)
DevOps
DevSafeOps

ABSTRACT

Developing autonomous driving (AD) systems is challenging due to the complexity of the systems and the need to assure their safe and reliable operation. The widely adopted approach of DevOps seems promising to support the continuous technological progress in AI and the demand for fast reaction to incidents, which necessitate continuous development, deployment, and monitoring. We present a systematic literature review meant to identify, analyse, and synthesise a broad range of existing literature related to usage of DevOps in autonomous driving development. Our results provide a structured overview of challenges and solutions, arising from applying DevOps to safety-related AI-enabled functions. Our results indicate that there are still several open topics to be addressed to enable safe DevOps for the development of safe AD.

1. Introduction

Artificial Intelligence (AI) has seen widespread adoption in various fields over the past decade, including the automotive industry. Autonomous Driving (AD) and Advanced Driver Assistance Systems (ADAS) are currently two prominent scenarios to apply AI. Ensuring the robustness and safety of such AI-based systems in a dynamic environment is a complex and multifaceted process that requires continuous monitoring and fast yet systematic reactions to the possible identified hazards.

AD development started back in the 1980s (Gudla et al., 2022) and there have been local experiments; however recent loss events highlight its ongoing safety challenges. For instance, in a recent mishap involving a robotaxi, a pedestrian was severely injured (Lawyer, 2024; Koopman, 2024). According to the investigation report (Lawyer, 2024), the cause was neither hardware nor software failure. The AD perception detected both the pedestrian and the adjacent vehicle. Weak recognition and response to nearby incidents, along with an inaccurate post-crash world model, are some of the technical issues and challenges highlighted by Koopman (Koopman, 2024). Moreover, a human safety driver is also considered a potential solution for handling these unforeseen complex scenarios (Koopman, 2024). This incident highlight the need for novel approaches in the design of ADS that enable the system to adapt its behaviour to similar unforeseen, complex, and yet numerous events. Releasing the system without sufficient confidence and necessary fallback strategies not only leads to safety risks but also poses delays in deployment or termination of the project. For instance, the aforementioned example led to the immediate revocation of ADS deployment and testing permits by the California Department of Motor Vehicles (DMV).[1]

The innovation rapidity in the automotive industry necessitates companies to adopt continuous development and integration approaches to remain competitive. Continuous approaches, such as DevOps, aim at function growth and refinement to lead to better customer experience after each design iteration (Google, 2024b). Hence, DevOps has the potential to enable continuous loops of monitoring and software adaptation for AD. It facilitates the expansion of the Operational Design Domain (ODD). Additionally, DevOps is crucial for maintaining the safety of the system against detected anomalies, and improve the adaptation speed to context/environmental/regulation updates (Zeller et al., 2024; Weiss et al., 2024). Hence, the safety community aims to integrate some aspects of DevOps into standards such as ISO/PAS 8800 (Safety and Artificial Intelligence) and ISO/TS 5083 (Safety for Automated Driving Systems).

However, the safety process required by automotive standards such as ISO 26262 ISO 26262:2018, 2018 or ISO 21448 (2022), and regulations such as UNECE R157 (ALKS) (UNECE, 2021) can make it

☆ Editor: Dr. Dario Di Nucci.
* Corresponding author at: Chalmers University of Technology, Department of Computer Science, Gothenburg, Sweden.
E-mail address: ali.nouri@volvocars.com (A. Nouri).
[1] DMV STATEMENT ON CRUISE LLC SUSPENSION, accessed February 28, 2025, https://www.dmv.ca.gov/portal/news-and-media/dmv-statement-on-cruise-llc-suspension/.

https://doi.org/10.1016/j.jss.2025.112555
Received 7 September 2024; Received in revised form 15 April 2025; Accepted 26 June 2025
Available online 11 July 2025

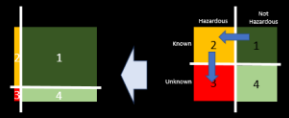
separation

tailoring

CH5: Verifica

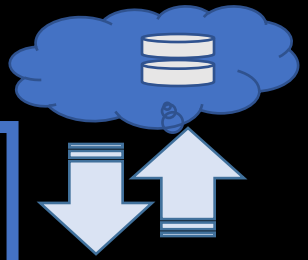
Safety argumentation

CH6: Operation Phase



ODD Expansion

Field Monitoring



Specification & Design

Conventional SW Dev.

HW Dev.

ML-based SW Dev.

Test

Test

Test

Test

No

All steps Done Correctly?

Yes

Define & Specify

Develop

Component Verification

Integrate

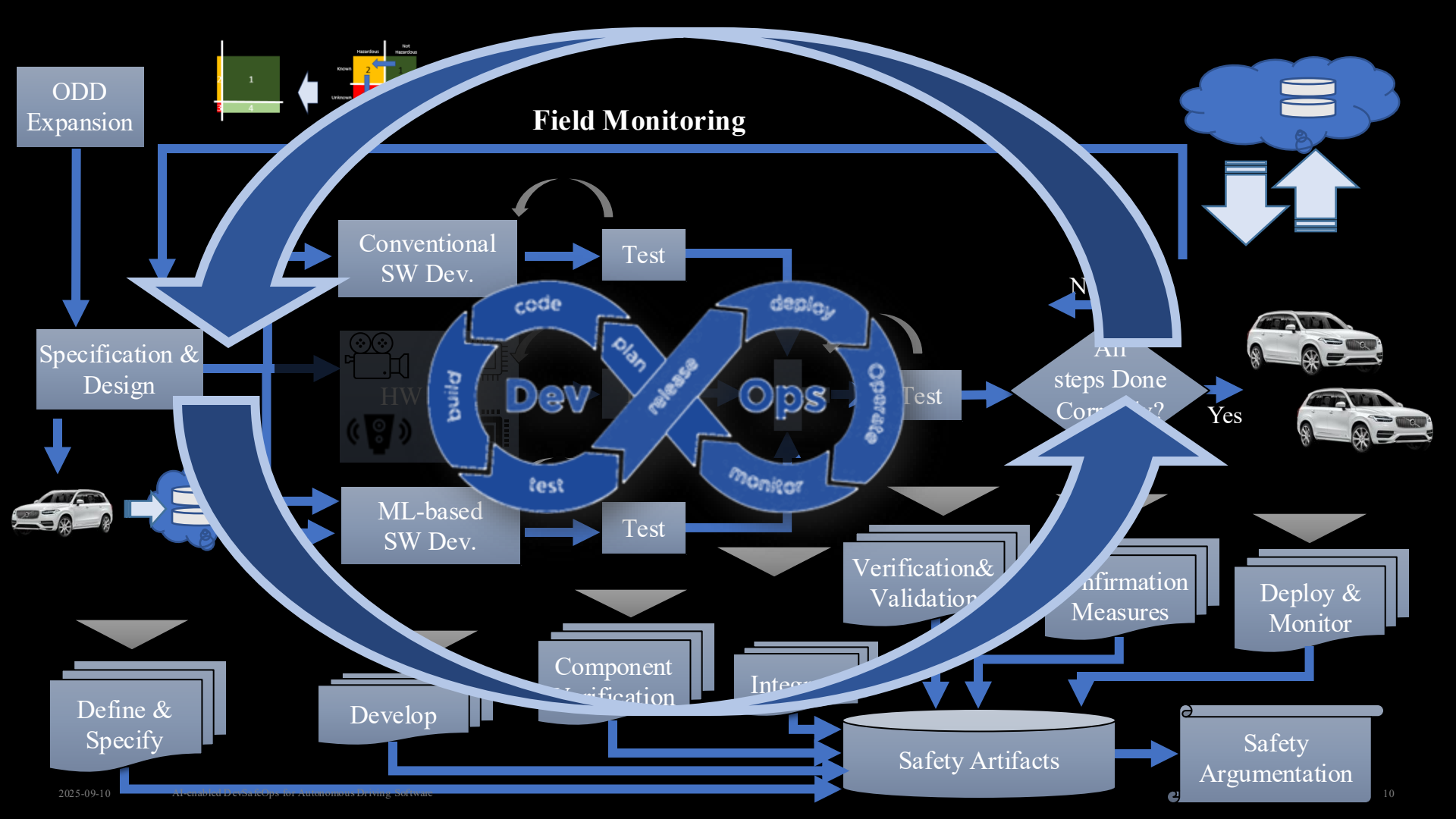
Verification& Validation

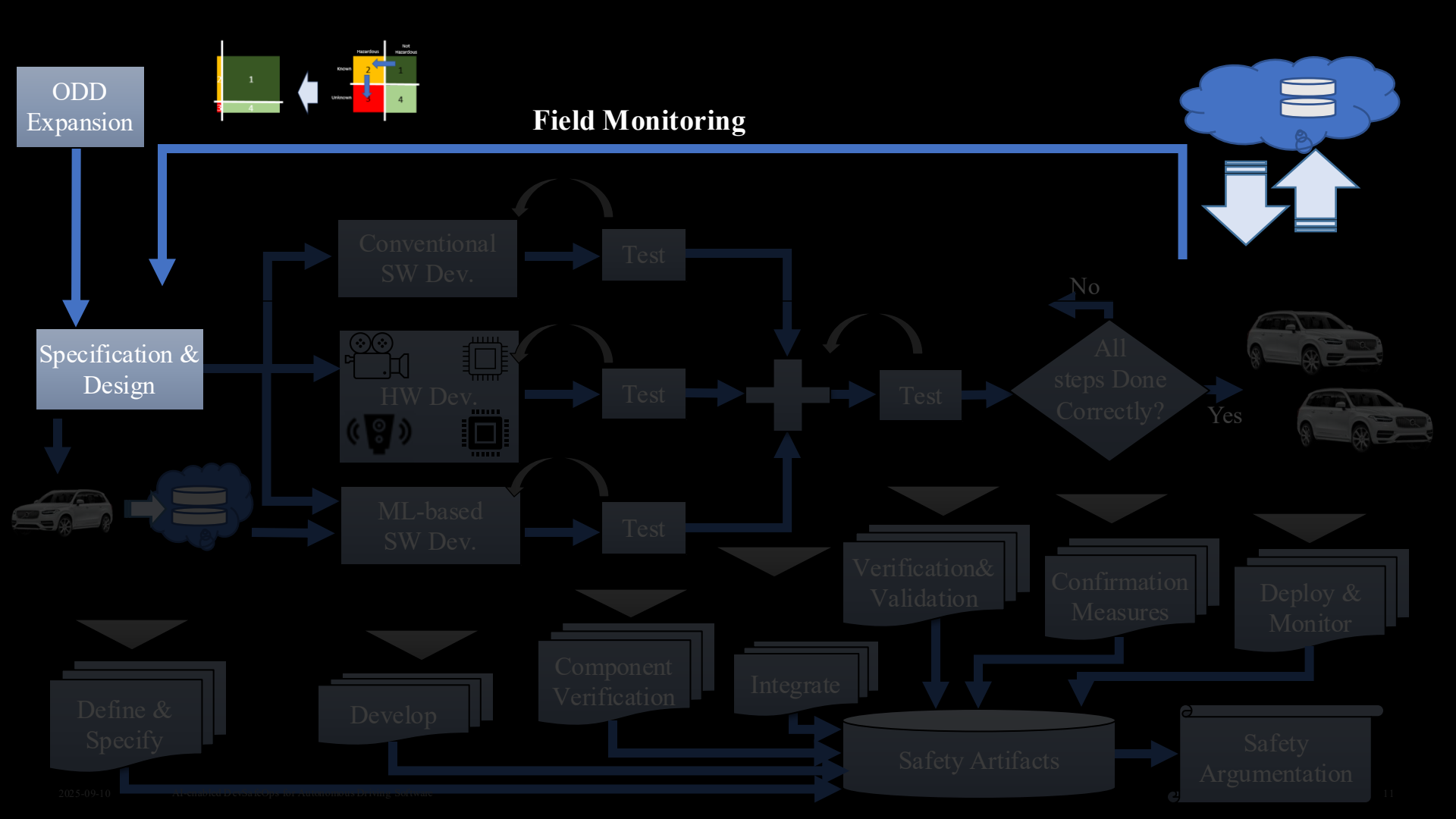
Confirmation Measures

Deploy & Monitor

Safety Artifacts

Safety Argumentation


ODD Expansion
Field Monitoring
Conventional SW Dev.
Test
Specification & Design
HW
code
plan
release
deploy
operate
monitor
test
build
Dev
Ops
Test
All steps Done Correctly?
Yes
ML-based SW Dev.
Test
Verification& Validation
Confirmation Measures
Deploy & Monitor
Define & Specify
Develop
Component
Safety Artifacts
Safety Argumentation


ODD Expansion
Field Monitoring
Specification & Design
Conventional SW Dev.
HW Dev.
ML-based SW Dev.
Test
Test
Test
Test
No
All steps Done Correctly?
Yes
Verification& Validation
Confirmation Measures
Deploy & Monitor
Define & Specify
Develop
Component Verification
Integrate
Safety Artifacts
Safety Argumentation

Welcome Your New AI Teammate: On Safety Analysis by Leashing Large Language Models

Ali Nouri
Volvo Cars &
Chalmers University of Technology
Gothenburg, Sweden
ali.nouri@volvocars.com

Beatriz Cabrero-Daniel
University of Gothenburg,
Department of Computer Science
Gothenburg, Sweden
beatriz.cabrero-daniel@gu.se

Fredrik Törner
Volvo Cars,
Gothenburg, Sweden
fredrik.torner@volvocars.com

Håkan Sivencrona
Zenseact
Gothenburg, Sweden
hakan.sivencrona@zenseact.com

Christian Berger
University of Gothenburg,
Department of Computer Science
Gothenburg, Sweden
christian.berger@gu.se

Abstract

DevOps is a necessity in many industries, including the development of Autonomous Vehicles. In those settings, there are iterative activities that reduce the speed of SafetyOps cycles. One of these activities is "Hazard Analysis & Risk Assessment" (HARA), which is an essential step to start the safety requirements specification. As a potential approach to increase the speed of this step in SafetyOps, we have delved into the capabilities of Large Language Models (LLMs). Our objective is to systematically assess their potential for application in the field of safety engineering. To that end, we propose a framework to support a higher degree of automation of HARA with LLMs. Despite our endeavors to automate as much of the process as possible, expert review remains crucial to ensure the validity and correctness of the analysis results, with necessary modifications made accordingly.

CCS Concepts: • **Software and its engineering → Software verification and validation**; • **General and reference → Verification**; • **Computing methodologies → Natural language processing**; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks.*

CAIN 2024, April 14–15, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0591-5/24/04…$15.00
https://doi.org/10.1145/3644815.3644953

Keywords: Hazard Analysis Risk Assessment, Autonomous Vehicles, DevOps, Safety, Large Language Model, Prompt Engineering, LLM, ChatGPT

ACM Reference Format:
Ali Nouri, Beatriz Cabrero-Daniel, Fredrik Törner, Håkan Sivencrona, and Christian Berger. 2024. Welcome Your New AI Teammate: On Safety Analysis by Leashing Large Language Models . In *Conference on AI Engineering Software Engineering for AI (CAIN 2024), April 14–15, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3644815.3644953

1 Introduction

The safety analysis of Autonomous Driving (AD) functions is crucial for engineers to identify hazardous events, assess their risks, and determine their root causes. Ensuring the safety of such functions often relies on iterative natural language (NL)-based activities, one of which is safety analysis. Artificial Intelligence (AI)-based tools capable of processing NL can be used to increase the efficiency and speed of these activities. One of the most promising tools is Large Language Model (LLM).

Safety analysis is, however, not trivial as it consists of various activities such as identification of failure modes, and their effect in specific situations, which aim to mitigate or avoid unreasonable risks. Standards such as ISO 26262 [1] or ISO 21448 [2], and regulations like UNECE R157 (ALKS) [3] often propose or mandate activities such as Hazard Analysis Risk Assessment (HARA) and System Theoretic Process Analysis (STPA) [4]. These guides are used when specifying the safety requirements for mitigation or avoidance strategies.

HARA is a well-known and usually required safety analysis for automotive functions, such as AD. The aim of this activity is to identify the hazardous events, categorize them, and to specify safety goals to prevent or mitigate them. Safety goals are top-level (i.e., vehicle-level) safety requirements that are then used in other safety activities. Together, these

Engineering Safety Requirements for Autonomous Driving with Large Language Models

Ali Nouri
Volvo Cars &
Chalmers University of Technology
Gothenburg, Sweden
ali.nouri@volvocars.com

Beatriz Cabrero-Daniel
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden
beatriz.cabrero-daniel@gu.se

Fredrik Törner
Volvo Cars
Gothenburg, Sweden
fredrik.torner@volvocars.com

Håkan Sivencrona
Zenseact
Gothenburg, Sweden
hakan.sivencrona@zenseact.com

Christian Berger
University of Gothenburg
Department of Computer Science and Engineering
Gothenburg, Sweden
christian.berger@gu.se

***Abstract*—Changes and updates in the requirement artifacts, which can be frequent in the automotive domain, are a challenge for SafetyOps. Large Language Models (LLMs), with their impressive natural language understanding and generating capabilities, can play a key role in automatically refining and decomposing requirements after each update. In this study, we propose a prototype of a pipeline of prompts and LLMs that receives an item definition and outputs solutions in the form of safety requirements. This pipeline also performs a review of the requirement dataset and identifies redundant or contradictory requirements. We first identified the necessary characteristics for performing HARA and then defined tests to assess an LLM's capability in meeting these criteria. We used design science with multiple iterations and let experts from different companies evaluate each cycle quantitatively and qualitatively. Finally, the prototype was implemented at a case company and the responsible team evaluated its efficiency.**

***Index Terms*—Requirement Engineering, Hazard Analysis Risk Assessment, Autonomous Vehicles, DevOps, Safety, Large Language Model, Prompt Engineering, LLM, ChatGPT**

I. Introduction

Software for Autonomous Driving (AD) is complex and ensuring its safety is critical. It must be assessed throughout the many sub-systems and sub-components that make up the desired AD behaviour, rendering it a difficult and complex task itself. Moreover, the complexity of the environment in which AD systems operate, and the possible malfunctions when interacting with other traffic agents lead to an almost infinite exploration space for potential issues.

Techniques to engineer and maintain requirements for such complex systems are commonplace in industrial setups. An example is Hazard Analysis and Risk Assessment (HARA), based on standards like ISO 26262 [1] and ISO 21448 [2], is an example to mitigate such issues to identify possible hazardous events and to assess their risk in a systematic way. Various strategies are used to specify safety requirements for events associated with a high risk that will be verified and validated at different stages of the project.

However, function descriptions, operational environments, and regulations in the automotive domain rapidly change. Hence, a company-specific DevOps cycle [3] including HARA needs to be repeated iteratively each time, when a new hazard or scenario is identified to potentially specify new relevant safety requirements [4]. An important ingredient for conducting HARA is brainstorming about possible hazards, which requires imagination and creativity. Recent technological successes in AI such as LLMs might be able to assist engineers when brainstorming.

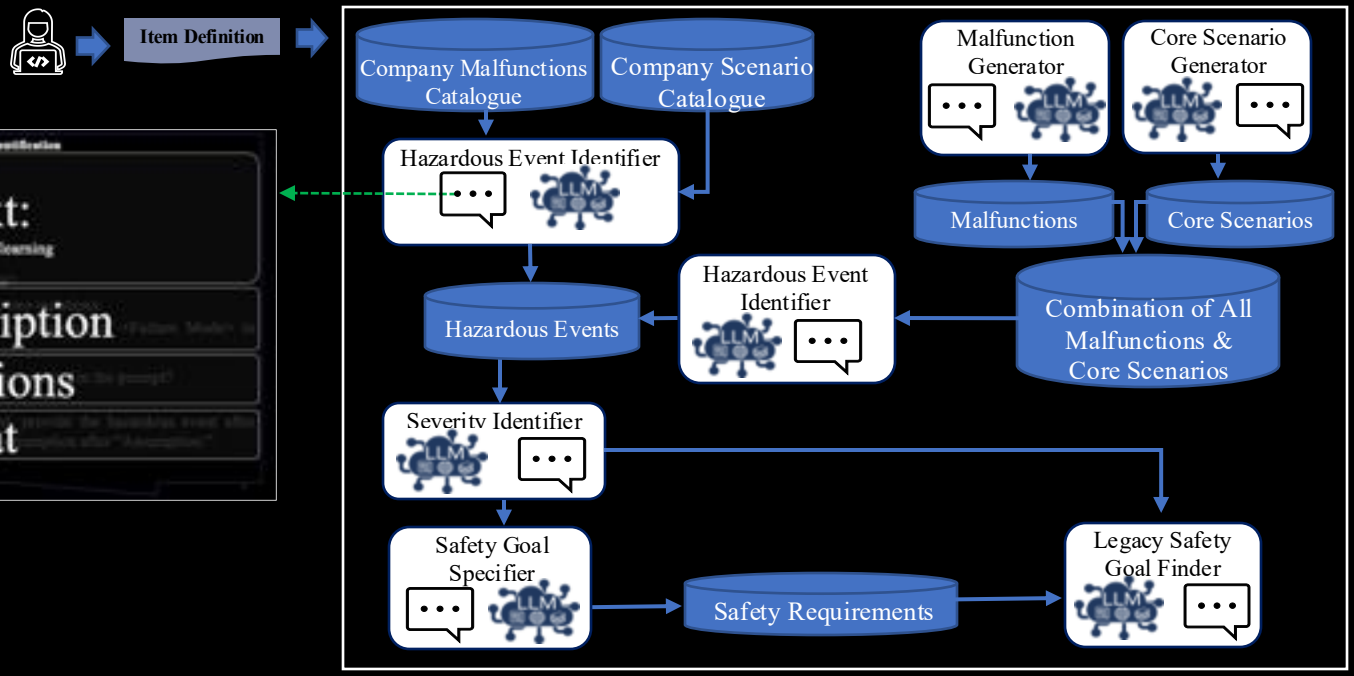
Our research goal is to design an LLM-based prototype capable of effectively supporting human engineers to specify safety requirements as needed for HARA in the context of complex automotive functions like AD. The design of the prototype was done in cycles: Firstly, identifying the LLM's limitations, followed by focusing on the task breakdown and prompt engineering, and finally evaluating the results in a real-world industrial context. We aim at answering the following research questions:

- RQ1 What are the limitations of using LLMs for specifying safety requirements for AD functions?
- RQ2 What is the task breakdown to enhance the LLMs' performance in specifying safety requirements using HARA?
- RQ3 How can prompt engineering enhance the LLMs' performance in specifying safety requirements for AD functions?

Our observations indicate that LLMs have the potential to effectively and efficiently specify safety requirements for AD functions.The remainder of the paper is structured as follows: Sec. III presents the methodology used to iteratively improve the LLM-based prototype. Sec. IV to VI discuss and justify the main design changes based on the evaluations of the generated artifacts. Sec. VII provides an overview of the main design

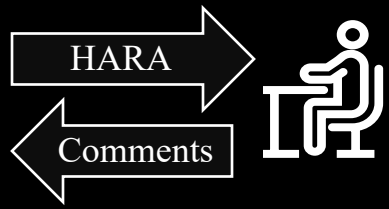
Funded by Sweden's Innovation Agency, Diarienummer: 2021-02585

Designed Pipeline of Prompts



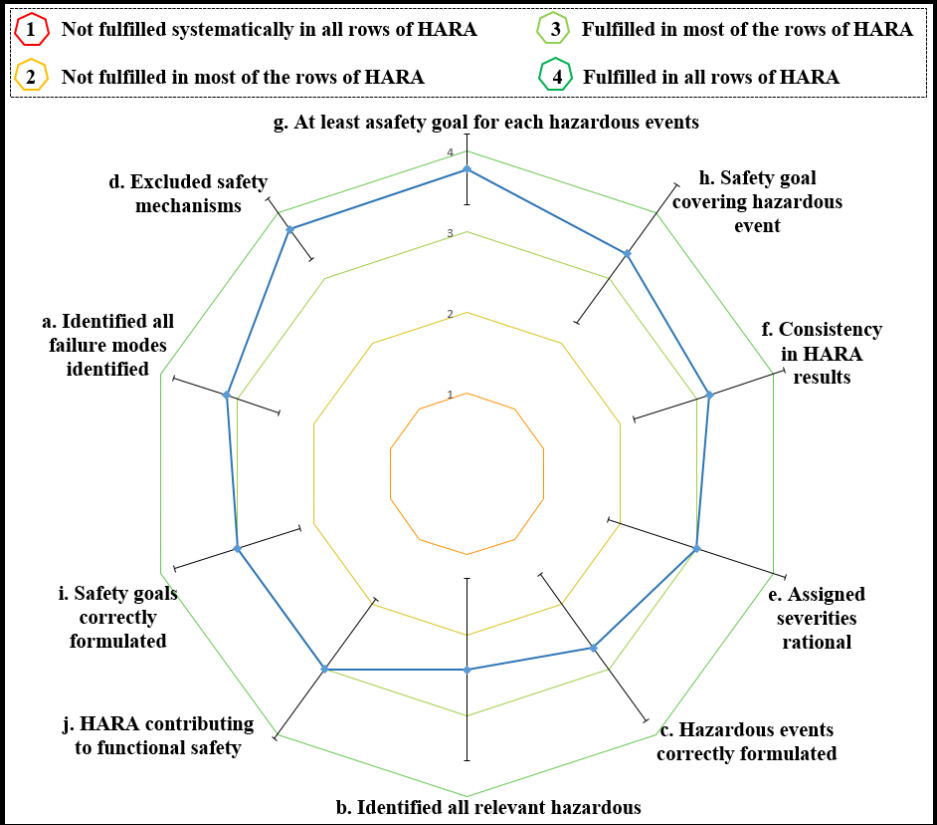
HARA Column:	Guideword	Hazard	Core Scenario	Detailed scenario	Hazardous event	Severity	Severity rationale	Exposure	Controllability	ASIL	New Four Safety Goals	Redundant Safety Goals
Relation to Prompts:	Output of Prompt 1 Input to Prompt 3, and 5		Output of Prompt 2 Input to Prompt 3	Output of Prompt 3 Internal Input to Prompt 3	Output of Prompt 3 Input to Prompt 4, 5, 6, and 7	Output of Prompt 4 Input to Prompt 5		Not in the scope			Output of Prompt 5	Output of Prompt 6

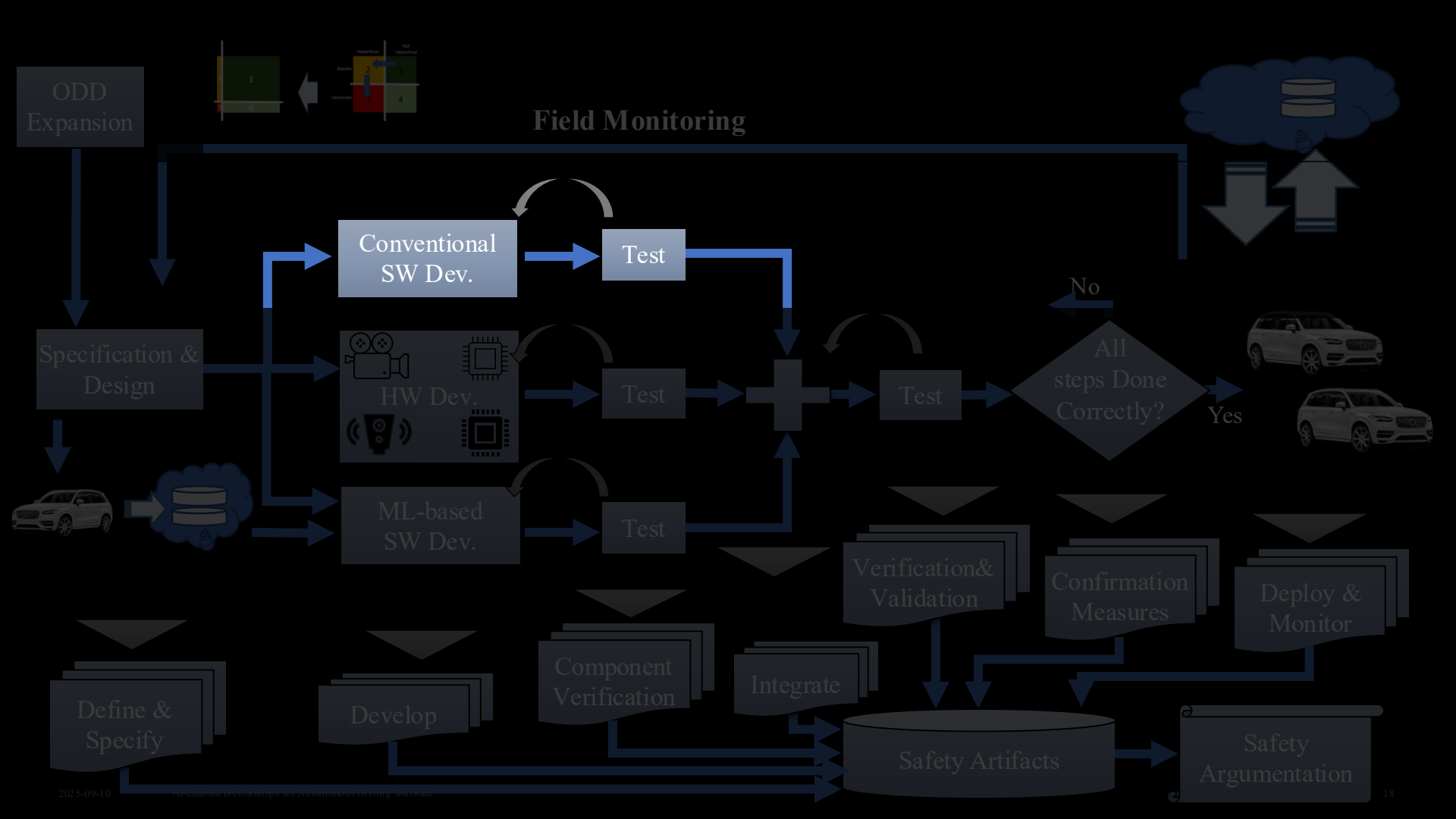
Evaluation: Experts Review Results



Root cause	Sample review comment
Scenarios	CR3: "The detailed scenarios are too detailed, which results in very specialized scenarios while excluding many other scenarios and also a risk of artificially lowering of E." CR3: (ID 50) "Ensure consistency: In detailed scenario there is a "truck approaching from the left" and in the severity rationale the truck is "a large stationary object", this is inconsistent." VR2: (ID 22) "Better to use VRU instead of Pedestrian in order to cover wider range of unprotected road users" CR2: (ID 51, 77, 91) "The scenario is too unclear to be able to formulate a valid hazardous event"
Hazardous event	CR2: (ID 19, 38, 25, 111) "The Hazardous event does not correlate with the malfunctioning behaviour in the described scenario"
Scenario Completeness	VR5: " ... difficult to determine completeness. Was a systematic approach applied? ... " VR3: "Have we covered the sharp turns, when commission has happened. That might lead to lateral instability."
Severity Identification	CR3: "Not enough rationale provided for the stated S. Many assumptions made without proper rationales." VR5: (ID 17) "CAEM doesn't seem to be limited in speed. How was S2 determined, vehicle speed could have been 130 kph? ..." CR3: (ID 77, and 196) " In severity rationale it is stated 'max allowed speed' in ID 77 and 'maximum allowed speed' in ID 196. This is not defined and if there is an upper limit of the host vehicle speed for CAEM this could be a safety mechanism."
Safety Goals Formulation	VR8: "Some safety goals have large overlap, ... Consider generalizing ...." VR5: "Safety goals do not need to explain why they exist, like "... to prevent unnecessary lane changes". ... specify the goal, such as "CAEM shall not cause lane departure unless to avoid collision". " CR3: "Ensure unambiguous safety goals: The safety goals contains a lot of undefined parts." VR6: "Vicinity need to be precise. The invitation shall be in in case the collision is imminent in-front. ..." VR1: (ID 22, 23, and 25) "Safety Goal 23 is more general and it includes Safety Goal 22. Safety Goal 25 is similar to Safety Goal 23", "Many safety goals are referring to same thing but different phrasing. " VR2: (ID 22) "when necessary" is vague and ambiguous."

Evaluation: Interview Results




ODD Expansion
Field Monitoring
Conventional SW Dev.
Test
Specification & Design
HW Dev.
Test
Test
No
All steps Done Correctly?
Yes
ML-based SW Dev.
Test
Verification& Validation
Confirmation Measures
Deploy & Monitor
Define & Specify
Develop
Component Verification
Integrate
Safety Artifacts
Safety Argumentation

On Simulation-Guided LLM-based Code Generation for Safe Autonomous Driving Software

Ali Nouri [1,2], Johan Andersson [2], Kailash De Jesus Hornig [2], Zhennan Fei[1,2]
Emil Knabe [1], Håkan Sivencrona [1], Beatriz Cabrero-Daniel [2,3], Christian Berger [2,3]
[1] *Volvo Cars, Gothenburg, Sweden*
[2] *Chalmers University of Technology, Department of Computer Science and Engineering,*
[3] *University of Gothenburg, Department of Computer Science and Engineering,*
{ali.nouri, zhennan.fei, hakan.sivencrona} @volvocars.com,
{beatriz.cabrero-daniel, christian.berger} @gu.se

Abstract

Automated Driving System (ADS) is a safety-critical software system responsible for the interpretation of the vehicle's environment and making decisions accordingly. The unbounded complexity of the driving context, including unforeseeable events, necessitate continuous improvement, often achieved through iterative DevOps processes. However, DevOps processes are themselves complex, making these improvements both time- and resource-intensive. Automation in code generation for ADS using Large Language Models (LLM) is one potential approach to address this challenge. Nevertheless, the development of ADS requires rigorous processes to verify, validate, assess, and qualify the code before it can be deployed in the vehicle and used. In this study, we developed and evaluated a prototype for automatic code generation and assessment using a designed pipeline of a LLM-based agent, simulation model, and rule-based feedback generator in an industrial setup. The LLM-generated code is evaluated automatically in a simulation model against multiple critical traffic scenarios, and an assessment report is provided as feedback to the LLM for modification or bug fixing. We report about the experimental results of the prototype employing Codellama:34b, DeepSeek (r1:32b and Coder:33b), CodeGemma:7b, Mistral:7b, and GPT4 for Adaptive Cruise Control (ACC) and Unsupervised Collision Avoidance by Evasive Manoeuvre (CAEM). We finally assessed the tool with 11 experts at two Original Equipment Manufacturers (OEMs) by conducting an interview study.

CCS Concepts

• **Software and its engineering** → **Software verification and validation**; • **General and reference** → **Verification**; • **Computing methodologies** → **Natural language processing**; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks.*

ACM ISBN 979-8-4007-1385-9/25/06
https://doi.org/10.1145/3756681.3756987

Keywords

DevOps, Autonomous Driving System, automated Software Generation, Large Language Model, Verification, Simulation

ACM Reference Format:
Ali Nouri [1,2], Johan Andersson [2], Kailash De Jesus Hornig [2], Zhennan Fei[1,2], Emil Knabe [1], Håkan Sivencrona [1], Beatriz Cabrero-Daniel [2,3], Christian Berger [2,3], [1] *Volvo Cars, Gothenburg, Sweden,* [2] *Chalmers University of Technology, Department of Computer Science and Engineering,*, [3] *University of Gothenburg, Department of Computer Science and Engineering,*. {ali.nouri, zhennan.fei, hakan.sivencrona} @volvocars.com,, {beatriz.cabrero-daniel, christian.berger} @gu.se . 2025. On Simulation-Guided LLM-based Code Generation for Safe Autonomous Driving Software . In *Proceedings of the 2025 Evaluation and Assessment in Software Engineering (EASE '25), June 17–20, 2025, Istanbul, Turkiye.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3756681.3756987

1 Introduction

To facilitate continuous improvement of Autonomous Driving Systems (ADS) and expansion of its Operational Design Domain (ODD) [1], software-defined vehicles rely heavily on software, while the hardware remains largely unchanged. Over-the-air (OTA) updates and centralized compute units enable the ongoing enhancement of functionalities during operation, leveraging DevOps. Rapid software updates are not only crucial for guaranteeing the safety of the system against new, unknown hazardous situations but also for improving customer experience. Automation can serve as a solution for rapid and efficient software implementation.

Natural language serves as the main input in various stages of the software engineering process, including function descriptions, requirements engineering [2], and scenario descriptions [3]. As LLMs have demonstrated their capability in tasks involving natural language and code generation, their application in automating code generation is promising. However, their capabilities have been examined in simple coding tasks [4–6] and not in safety-related complex applications. Moreover, due to LLMs' known weaknesses and the safety-related nature of ADS, the generated code must follow stringent processes prescribed in ISO 26262 [7] and ISO 21448 [1] such as code reviews, verification, and validation. Software in the Loop (SiL) and simulation [8] environments can be used to verify the code in a closed loop before it is integrated into hardware and reviewed by engineers. This serves as a preliminary validations to increase efficiency and improve code quality before other resource-demanding steps, such as code reviews.

SCAN ME

SCAN ME

Large Language Models in Code Co-generation for Safe Autonomous Vehicles

Ali Nouri[1,3(✉)], Beatriz Cabrero-Daniel[2,3], Zhennan Fei[1,3], Krishna Ronanki[2,3], Håkan Sivencrona[1], and Christian Berger[2,3]

[1] Volvo Cars, Gothenburg, Sweden
{ali.nouri, zhennan.fei, hakan.sivencrona}@volvocars.com
[2] University of Gothenburg, Sweden
{beatriz.cabrero-daniel, ronanki, christian.berger}@gu.se
[3] Chalmers University of Technology, Gothenburg, Sweden

Abstract. Software engineers in various industrial domains are already using Large Language Models (LLMs) to accelerate the process of implementing parts of software systems. When considering its potential use for ADAS or AD systems in the automotive context, there is a need to systematically assess this new setup: LLMs entail a well-documented set of risks for safety-related systems' development due to their stochastic nature. To reduce the effort for code reviewers to evaluate LLM-generated code, we propose an evaluation pipeline to conduct sanity-checks on the generated code. We compare the performance of six state-of-the-art LLMs (CodeLlama, CodeGemma, DeepSeek-r1, DeepSeek-Coders, Mistral, and GPT-4) on four safety-related programming tasks. Additionally, we qualitatively analyse the most frequent faults generated by these LLMs, creating a failure-mode catalogue to support human reviewers. Finally, the limitations and capabilities of LLMs in code generation, and the use of the proposed pipeline in the existing process, are discussed.

Keywords: DevOps, Autonomous Driving System, Automated Code Generation, Large Language Model, Verification, Simulation

1 Introduction

Function realisations and improvement in software-defined vehicles require continuous software updates; hence, rapid, efficient, and continuous software development is crucial to maintaining competitiveness and user satisfaction. LLMs can be seen as a potential element in the software development pipeline, as their capability in code generation has been demonstrated [10]. However, the limitations and capabilities of LLMs are under-explored, as they are examined primarily for simple coding tasks and less for complex and novel tasks that require creativity.

Generating code with LLMs might require multiple tries, given the LLMs' stochastic behaviour [2] and the complexity of the task. Moreover, as an LLM does not possess a proper understanding of the real world, it might fail to propose an appropriate strategy in the generated code, that may not be easily detected

SCAN ME

Large Language Models in Code Co-generation for Safe Autonomous Vehicles

Ali Nouri[1,3(✉)], Beatriz Cabrero-Daniel[2,3], Zhennan Fei[1,3], Krishna Ronanki[2,3], Håkan Sivencrona[1], and Christian Berger[2,3]

[1] Volvo Cars, Gothenburg, Sweden
{ali.nouri, zhennan.fei, hakan.sivencrona}@volvocars.com
[2] University of Gothenburg, Sweden
{beatriz.cabrero-daniel, ronanki, christian.berger}@gu.se
[3] Chalmers University of Technology, Gothenburg, Sweden

Abstract. Software engineers in various industrial domains are already using Large Language Models (LLMs) to accelerate the process of implementing parts of software systems. When considering its potential use for ADAS or AD systems in the automotive context, there is a need to systematically assess this new setup: LLMs entail a well-documented set of risks for safety-related systems' development due to their stochastic nature. To reduce the effort for code reviewers to evaluate LLM-generated code, we propose an evaluation pipeline to conduct sanity-checks on the generated code. We compare the performance of six state-of-the-art LLMs (CodeLlama, CodeGemma, DeepSeek-r1, DeepSeek-Coders, Mistral, and GPT-4) on four safety-related programming tasks. Additionally, we qualitatively analyse the most frequent faults generated by these LLMs, creating a failure-mode catalogue to support human reviewers. Finally, the limitations and capabilities of LLMs in code generation, and the use of the proposed pipeline in the existing process, are discussed.

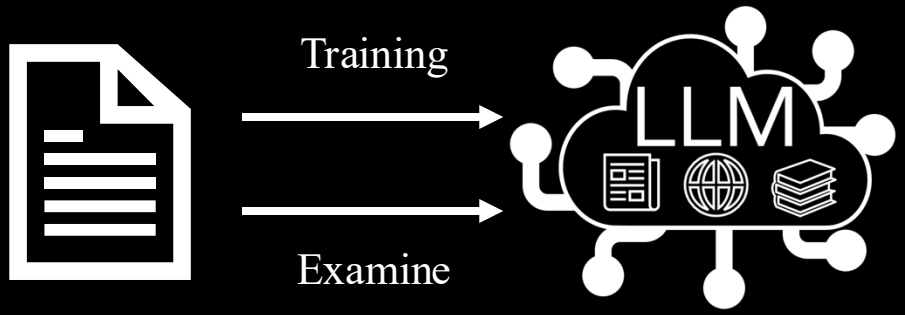
Keywords: DevOps, Autonomous Driving System, Automated Code Generation, Large Language Model, Verification, Simulation

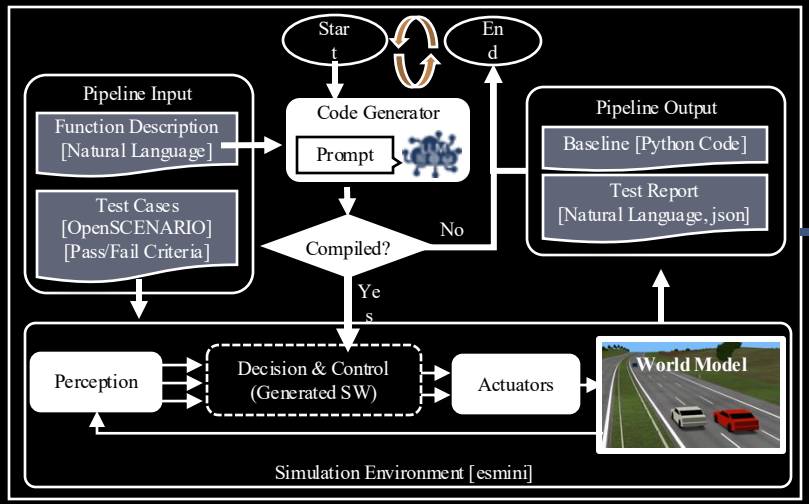
1 Introduction

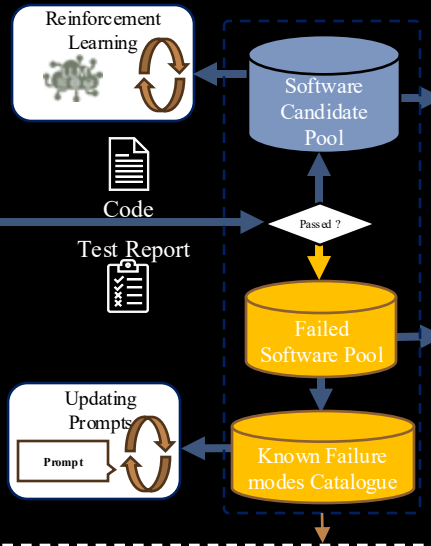
Function realisations and improvement in software-defined vehicles require continuous software updates; hence, rapid, efficient, and continuous software development is crucial to maintaining competitiveness and user satisfaction. LLMs can be seen as a potential element in the software development pipeline, as their capability in code generation has been demonstrated [10]. However, the limitations and capabilities of LLMs are under-explored, as they are examined primarily for simple coding tasks and less for complex and novel tasks that require creativity.

Generating code with LLMs might require multiple tries, given the LLMs' stochastic behaviour [2] and the complexity of the task. Moreover, as an LLM does not possess a proper understanding of the real world, it might fail to propose an appropriate strategy in the generated code, that may not be easily detected

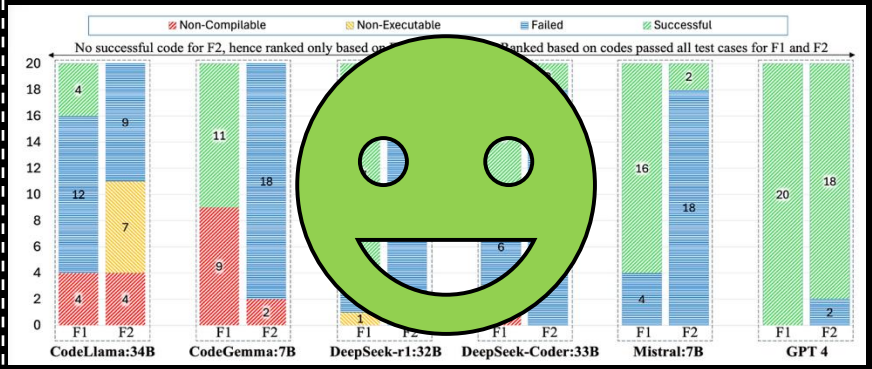
Benchmark leakage



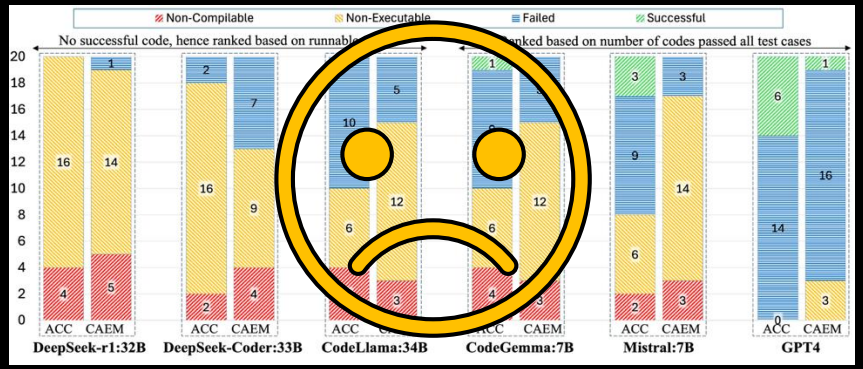




Evaluation results across four functions with varying levels of complexity and benchmark leakage risks



Low Complexity & High Benchmark Leakage



High Complexity & Low Benchmark Leakage

On Simulation-Guided LLM-based Code Generation for Safe Autonomous Driving Software

Ali Nouri [1,2], Johan Andersson [2], Kailash De Jesus Hornig [2], Zhennan Fei[1,2]
Emil Knabe [1], Håkan Sivencrona [1], Beatriz Cabrero-Daniel [2,3], Christian Berger [2,3]
[1] *Volvo Cars, Gothenburg, Sweden*
[2] *Chalmers University of Technology, Department of Computer Science and Engineering,*
[3] *University of Gothenburg, Department of Computer Science and Engineering,*
{ali.nouri, zhennan.fei, hakan.sivencrona} @volvocars.com,
{beatriz.cabrero-daniel, christian.berger} @gu.se

Abstract

Automated Driving System (ADS) is a safety-critical software system responsible for the interpretation of the vehicle's environment and making decisions accordingly. The unbounded complexity of the driving context, including unforeseeable events, necessitate continuous improvement, often achieved through iterative DevOps processes. However, DevOps processes are themselves complex, making these improvements both time- and resource-intensive. Automation in code generation for ADS using Large Language Models (LLM) is one potential approach to address this challenge. Nevertheless, the development of ADS requires rigorous processes to verify, validate, assess, and qualify the code before it can be deployed in the vehicle and used. In this study, we developed and evaluated a prototype for automatic code generation and assessment using a designed pipeline of a LLM-based agent, simulation model, and rule-based feedback generator in an industrial setup. The LLM-generated code is evaluated automatically in a simulation model against multiple critical traffic scenarios, and an assessment report is provided as feedback to the LLM for modification or bug fixing. We report about the experimental results of the prototype employing Codellama:34b, DeepSeek (r1:32b and Coder:33b), CodeGemma:7b, Mistral:7b, and GPT4 for Adaptive Cruise Control (ACC) and Unsupervised Collision Avoidance by Evasive Manoeuvre (CAEM). We finally assessed the tool with 11 experts at two Original Equipment Manufacturers (OEMs) by conducting an interview study.

CCS Concepts

• **Software and its engineering** → **Software verification and validation**; • **General and reference** → **Verification**; • **Computing methodologies** → **Natural language processing**; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks.*

EASE '25, June 17–20, 2025, Istanbul, Turkiye

ACM ISBN 979-8-4007-1385-9/25/06
https://doi.org/10.1145/3756681.3756987

Keywords

DevOps, Autonomous Driving System, automated Software Generation, Large Language Model, Verification, Simulation

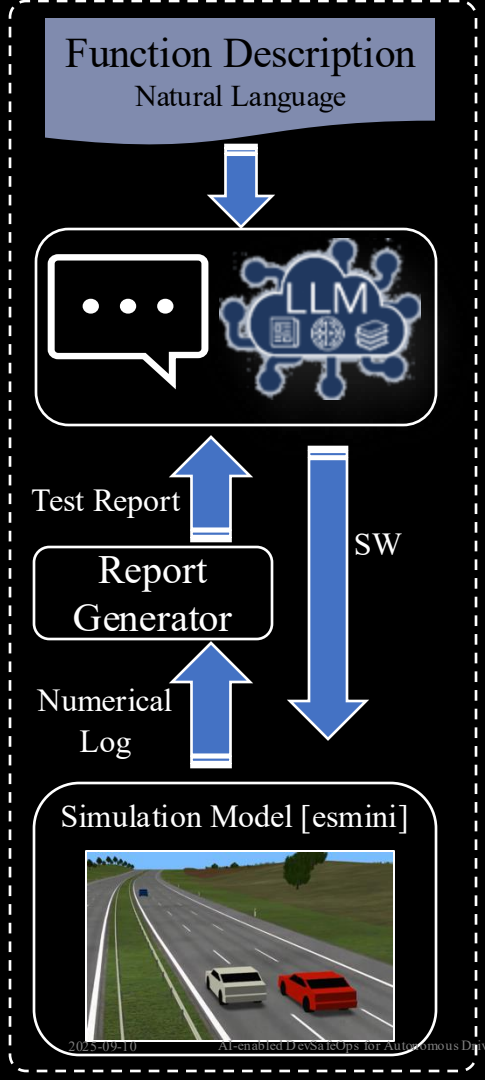
ACM Reference Format:
Ali Nouri [1,2], Johan Andersson [2], Kailash De Jesus Hornig [2], Zhennan Fei[1,2], Emil Knabe [1], Håkan Sivencrona [1], Beatriz Cabrero-Daniel [2,3], Christian Berger [2,3]. [1] *Volvo Cars, Gothenburg, Sweden,* [2] *Chalmers University of Technology, Department of Computer Science and Engineering,*. [3] *University of Gothenburg, Department of Computer Science and Engineering,*. {ali.nouri, zhennan.fei, hakan.sivencrona} @volvocars.com,. {beatriz.cabrero-daniel, christian.berger} @gu.se . 2025. On Simulation-Guided LLM-based Code Generation for Safe Autonomous Driving Software . In *Proceedings of the 2025 Evaluation and Assessment in Software Engineering (EASE '25), June 17–20, 2025, Istanbul, Turkiye.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3756681.3756987

1 Introduction

To facilitate continuous improvement of Autonomous Driving Systems (ADS) and expansion of its Operational Design Domain (ODD) [1], software-defined vehicles rely heavily on software, while the hardware remains largely unchanged. Over-the-air (OTA) updates and centralized compute units enable the ongoing enhancement of functionalities during operation, leveraging DevOps. Rapid software updates are not only crucial for guaranteeing the safety of the system against new, unknown hazardous situations but also for improving customer experience. Automation can serve as a solution for rapid and efficient software implementation.

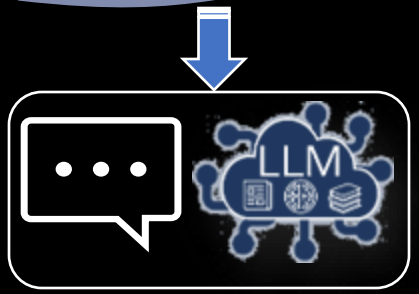
Natural language serves as the main input in various stages of the software engineering process, including function descriptions, requirements engineering [2], and scenario descriptions [3]. As LLMs have demonstrated their capability in tasks involving natural language and code generation, their application in automating code generation is promising. However, their capabilities have been examined in simple coding tasks [4–6] and not in safety-related complex applications. Moreover, due to LLMs' known weaknesses and the safety-related nature of ADS, the generated code must follow stringent processes prescribed in ISO 26262 [7] and ISO 21448 [1] such as code reviews, verification, and validation. Software in the Loop (SiL) and simulation [8] environments can be used to verify the code in a closed loop before it is integrated into hardware and reviewed by engineers. This serves as a preliminary validations to increase efficiency and improve code quality before other resource-demanding steps, such as code reviews.

SCAN ME


Function Description
Natural Language
LLM
Test Report
Report Generator
Numerical Log
SW
Simulation Model [esmini]

Function Description

Natural Language



Test Report

Report Generator

Numerical Log

SW

Simulation Model [esmini]

Prompt

You are tasked with writing Python files to control autonomous cars within a simulated environment. These controller files receive information about the simulation through a `State` class object, allowing commands to be issued to a vehicle referred to as the Ego car.

State Class Documentation: The `CustomController` interacts with the simulation via the `State` class, which provides the following functionalities:

```
def __init__(self, simulator):
    ...
    # A list of all the cars in the simulation. The "ego" car is at index 0.
    self.vehicles = []

def switch_lane(self, lane_id):
    # Changes the ego car's lane. 1 for left and -1 for right.
```

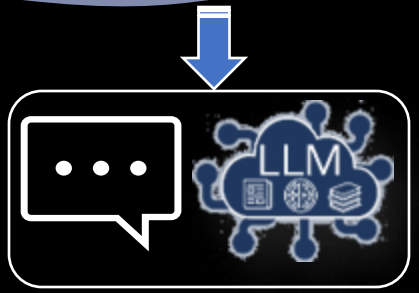
Vehicle Class:
Each vehicle in the simulation, including the Ego car, is represented as follows:

```
def __init__(self, identity, position, speed, lane_id, s, t):
    self.id = identity  # Integer
```

Your task is to generate a custom_controller.py file that controlls
<Add function description>

Function Description

Natural Language



Test Report

Report Generator

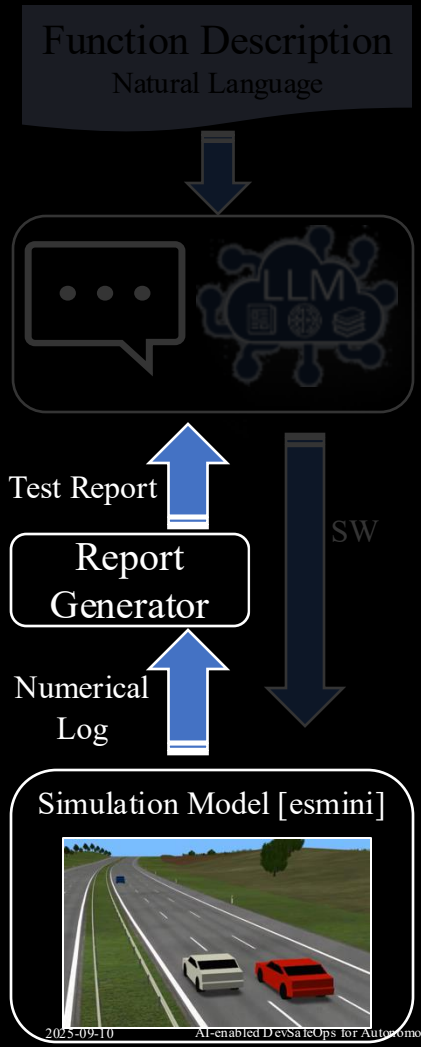
Numerical Log

SW

LLM-based Generated Code(Version 1)

Simulation Model [esmini]

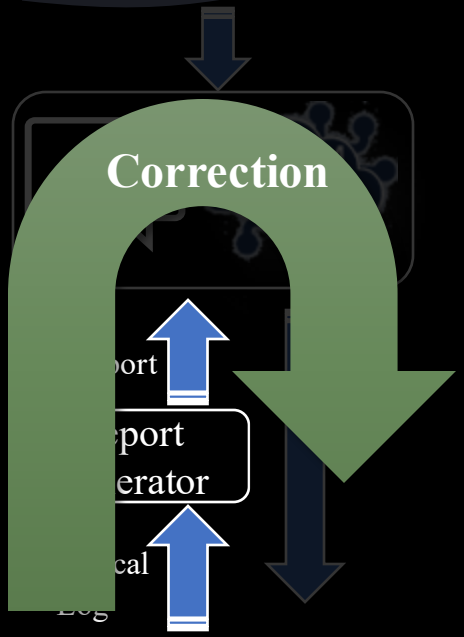
```python
class CustomController:
    def __init__(self, state):
        self.state = state

    def step(self):
        ego = self.state.vehicles[0]
        current_lane = ego.lane_id
        lanes = [-2, -3, -4]  # Valid lanes on the road
        imminent_collision = False
        safe_distance = 20  # Minimum safe distance to consider a lane free (both in front and behind ego)

        # Check if there's a risk of collision in the current lane
        for vehicle in self.state.vehicles:
            if vehicle.lane_id == current_lane and vehicle.id != ego.id:
                if vehicle.s > ego.s and vehicle.s - ego.s < safe_distance:
                    imminent_collision = True
                    break

        if imminent_collision:
            # Attempt to change to a safer lane
            for lane in lanes:
                if lane == current_lane:
                    continue
                if self.is_lane_safe(lane, ego, safe_distance):
                    self.state.switch_lane(lane - current_lane)
                    return

    def is_lane_safe(self, lane_id, ego, safe_distance):
        """ Checks if the given lane is safe for the ego vehicle to switch into. """
        for vehicle in self.state.vehicles:
            if vehicle.lane_id == lane_id:
                # Check if the vehicle is within a dangerous range
                distance = abs(vehicle.s - ego.s)
                if distance < safe_distance:
                    return False
        return True
```



Watch the demo here

SCAN ME

Function Description
Natural Language



Simulation Model [esmini]

Failed

Passed

Passed

Failed

Failed

Passed

Passed

```
Iteration 0 reports:
Log based report for scenario: cut-in_high.xosc:
Fail: Ego was involved in a collision at time: 13.3 s with a speed of 33.333333 m/s, colliding with: OverTaker.
Fail: Closest distance Ego comes to any vehicle is 5.63 m to vehicle #2 at time: 13.2 s, which is closer than the allowed minimum of 7 m.
Pass: Greatest absolute lane offset of Ego: 8.00 m at time: 0.0 s, within the allowed maximum of 12.575 m.
Pass: Smallest absolute lane offset of Ego: 4.42 m at time: 14.8 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_middle.xosc:
Pass: No collisions were detected.
Pass: Closest distance Ego comes to any vehicle is 16.26 m to vehicle #2 at time: 9.9 s, respecting the minimum allowed distance of 7 m.
Pass: Greatest absolute lane offset of Ego: 8.00 m at time: 0.0 s, within the allowed maximum of 12.575 m.
Pass: Smallest absolute lane offset of Ego: 4.42 m at time: 13.0 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_low.xosc:
Pass: No collisions were detected.
Pass: Closest distance Ego comes to any vehicle is 7.99 m to vehicle #2 at time: 15.5 s, respecting the minimum allowed distance of 7 m.
Pass: Greatest absolute lane offset of Ego: 8.00 m at time: 0.0 s, within the allowed maximum of 12.575 m.
Pass: Smallest absolute lane offset of Ego: 4.42 m at time: 18.6 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_double_EM.xosc:
Fail: Ego was involved in a collision at time: 16.7 s with a speed of 30.0 m/s, colliding with: OverTaker2.
Fail: Closest distance Ego comes to any vehicle is 1.68 m to vehicle #3 at time: 16.8 s, which is closer than the allowed minimum of 7 m.
Pass: Greatest absolute lane offset of Ego: 8.00 m at time: 0.0 s, within the allowed maximum of 12.575 m.
Pass: Smallest absolute lane offset of Ego: 4.42 m at time: 14.6 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_block_EM.xosc:
Pass: No collisions were detected.
Fail: Closest distance Ego comes to any vehicle is 6.90 m to vehicle #2 at time: 11.5 s, which is closer than the allowed minimum of 7 m.
Pass: Greatest absolute lane offset of Ego: 11.70 m at time: 13.1 s, within the allowed maximum of 12.7 m.
Pass: Smallest absolute lane offset of Ego: 8.00 m at time: 0.0 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_empty_commission.xosc:
Pass: Minimum speed of Ego: 30.00 m/s at time: 0.0 s, above the minimum limit of 28 m/s.
Pass: Greatest absolute lane offset of Ego: 8.00 m at time: 0.0 s, within the allowed maximum of 12.575 m.
Pass: Smallest absolute lane offset of Ego: 8.00 m at time: 0.0 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_meeting_commission.xosc:
Pass: No collisions were detected.
Pass: Minimum speed of Ego: 30.00 m/s at time: 0.0 s, above the minimum limit of 28 m/s.
Pass: Greatest absolute lane offset of Ego: 4.42 m at time: 0.0 s, within the allowed maximum of 6.7 m.
Pass: Smallest absolute lane offset of Ego: 4.42 m at time: 0.0 s, above the allowed minimum of 3.425 m.
```

Function Description
Natural Language

Correction

port

eport
erator

cal

Simulation Model [esmini]

Correction Prompt

You are tasked with writing Python files to control autonomous cars within a simulated environment. These controller files receive information about the simulation through a `State` class object, allowing commands to be issued to a vehicle referred to as the Ego car.

State Class Documentation: The `CustomController` interacts with the simulation via the `State` class, which provides the following functionalities:

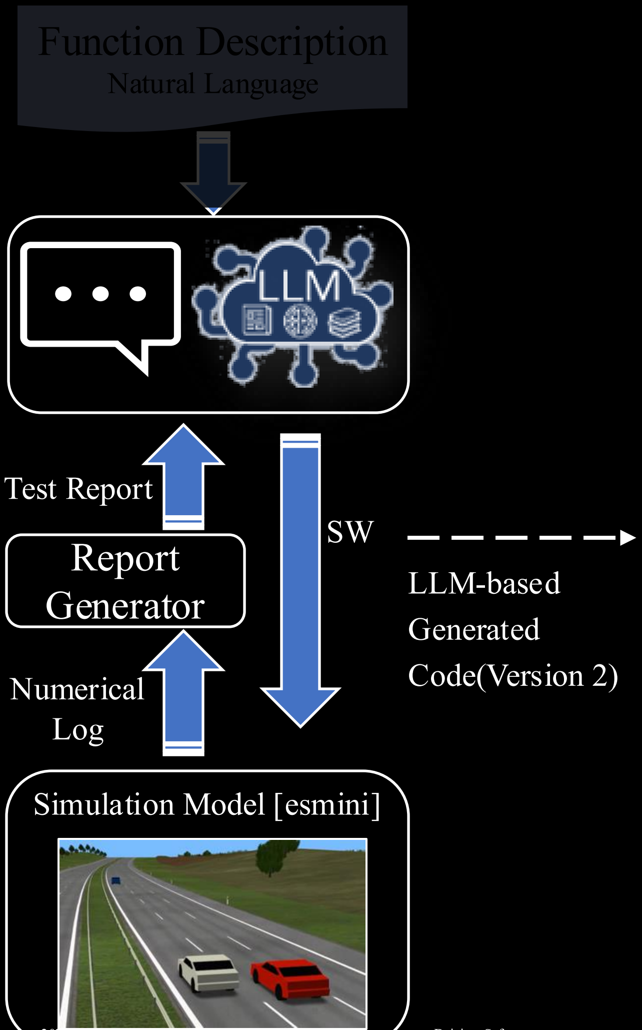
```
def __init__(self, simulator):
    ...
    # A list of all the cars in the simulation. The "ego" car is at index 0.
    self.vehicles = []

def switch_lane(self, lane_id):
    # Changes the ego car's lane. 1 for left and -1 for right.
```

Vehicle Class:
Each vehicle in the simulation, including the Ego car, is represented as follows:

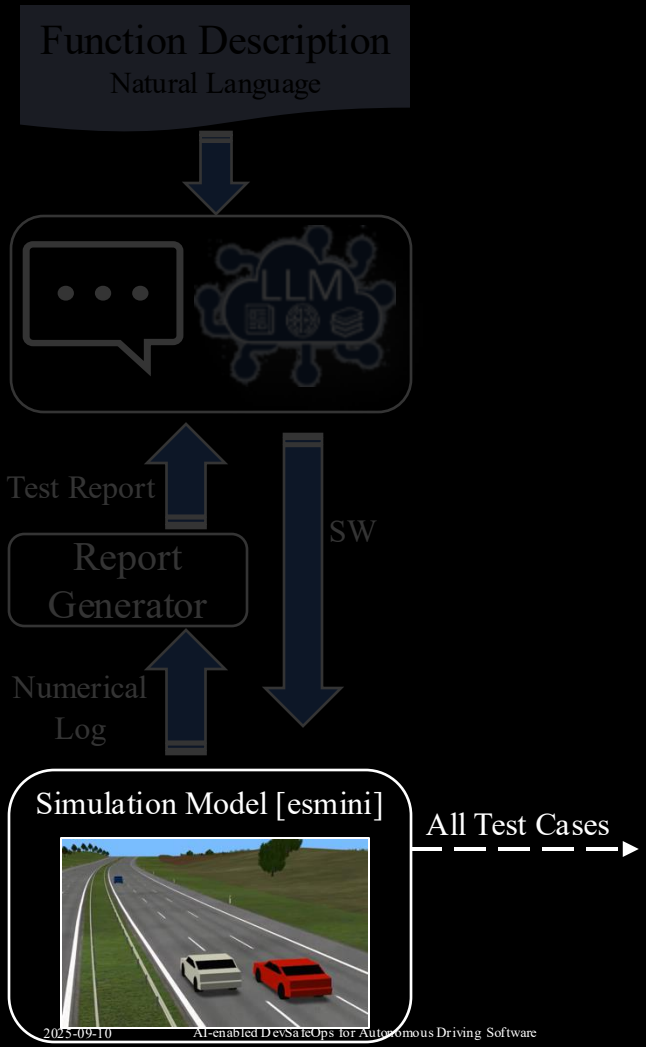
```
def __init__(self, identity, position, speed, lane_id, s, t):
    self.id = identity  # Integer
```

Here is a code which failed in some of its tested and its report:
<Add the scenario descriptions>
< Add the last version of Code>
<Add the Test Results>

Your task is to generate a custom_controller.py file that controlls <Add function description>



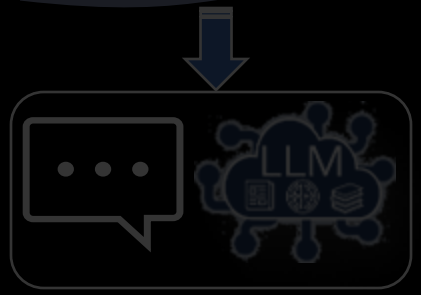

```python
# version 1

class CustomController:
    def __init__(self, state):
        self.state = state
        self.reaction_time = 1.5  # seconds
        self.braking_capacity = -5  # m/s^2

    def step(self):
        ego = self.state.vehicles[0]
        current_lane = ego.lane_id
        lanes = [-2, -3, -4]  # Valid lanes on the road

        for vehicle in self.state.vehicles:
            if vehicle.lane_id == current_lane and vehicle.id != ego.id:
                relative_speed = ego.speed - vehicle.speed
                time_to_collision = (vehicle.s - ego.s) / relative_speed if relative_speed != 0 else float('inf')

                if time_to_collision < self.reaction_time:
                    if not self.attempt_lane_change(ego):
                        # Adjust speed to prevent collision
                        self.adjust_speed(ego, vehicle, relative_speed)

    def attempt_lane_change(self, ego):
        current_lane = ego.lane_id
        for lane in [-2, -3, -4]:
            if lane != current_lane and self.is_lane_safe(lane, ego, self.calculate_safe_distance(ego.speed)):
                self.state.switch_lane(lane - current_lane)
                return True
        return False

    def is_lane_safe(self, lane_id, ego, safe_distance):
        for vehicle in self.state.vehicles:
            if vehicle.lane_id == lane_id:
                distance = abs(vehicle.s - ego.s)
                if distance < safe_distance:
                    return False
        return True

    def calculate_safe_distance(self, speed):
        return speed * self.reaction_time + (speed ** 2) / (2 * abs(self.braking_capacity))

    def adjust_speed(self, ego, vehicle, relative_speed):
        if relative_speed < 0:  # Ego is faster than the vehicle ahead
            new_speed = max(0, vehicle.speed - 5)
            ego.speed = new_speed
        else:
            ego.speed += 5  # Try to increase speed to avoid collision
```



Watch the demo here

SCAN ME

Function Description
Natural Language



Test Report

Report Generator

Generated Report for Test Cases

Numerical Log

Simulation Model [esmini]

```
Iteration 1 reports:
Log based report for scenario: cut-in_high.xosc:
Pass: No collisions were detected.
Pass: Closest distance Ego comes to any vehicle is 24.27 m to vehicle #2 at time: 8.0 s, respecting the minimum allowed distance of 7 m.
Pass: Greatest absolute lane offset of Ego: 8.00 m at time: 0.0 s, within the allowed maximum of 12.575 m.
Pass: Smallest absolute lane offset of Ego: 4.42 m at time: 11.1 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_middle.xosc:
Pass: No collisions were detected.
Pass: Closest distance Ego comes to any vehicle is 16.26 m to vehicle #2 at time: 9.9 s, respecting the minimum allowed distance of 7 m.
Pass: Greatest absolute lane offset of Ego: 8.00 m at time: 0.0 s, within the allowed maximum of 12.575 m.
Pass: Smallest absolute lane offset of Ego: 4.42 m at time: 13.0 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_low.xosc:
Pass: No collisions were detected.
Pass: Closest distance Ego comes to any vehicle is 7.99 m to vehicle #2 at time: 15.5 s, respecting the minimum allowed distance of 7 m.
Pass: Greatest absolute lane offset of Ego: 8.00 m at time: 0.0 s, within the allowed maximum of 12.575 m.
Pass: Smallest absolute lane offset of Ego: 4.42 m at time: 18.6 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_double_EM.xosc:
Pass: No collisions were detected.
Pass: Closest distance Ego comes to any vehicle is 13.81 m to vehicle #3 at time: 16.4 s, respecting the minimum allowed distance of 7 m.
Pass: Greatest absolute lane offset of Ego: 11.70 m at time: 18.4 s, within the allowed maximum of 12.575 m.
Pass: Smallest absolute lane offset of Ego: 4.42 m at time: 11.5 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_block_EM.xosc:
Pass: No collisions were detected.
Pass: Closest distance Ego comes to any vehicle is 22.14 m to vehicle #2 at time: 6.8 s, respecting the minimum allowed distance of 7 m.
Pass: Greatest absolute lane offset of Ego: 11.70 m at time: 9.9 s, within the allowed maximum of 12.7 m.
Pass: Smallest absolute lane offset of Ego: 8.00 m at time: 0.0 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_empty_commission.xosc:
Pass: Minimum speed of Ego: 30.00 m/s at time: 0.0 s, above the minimum limit of 28 m/s.
Pass: Greatest absolute lane offset of Ego: 8.00 m at time: 0.0 s, within the allowed maximum of 12.575 m.
Pass: Smallest absolute lane offset of Ego: 8.00 m at time: 0.0 s, above the allowed minimum of 3.425 m.

Log based report for scenario: cut-in_meeting_commission.xosc:
Pass: No collisions were detected.
Pass: Minimum speed of Ego: 30.00 m/s at time: 0.0 s, above the minimum limit of 28 m/s.
Pass: Greatest absolute lane offset of Ego: 4.42 m at time: 0.0 s, within the allowed maximum of 6.7 m.
Pass: Smallest absolute lane offset of Ego: 4.42 m at time: 0.0 s, above the allowed minimum of 3.425 m.

1 iterations of corrections were performed. The final controller was successful.
```

Passed

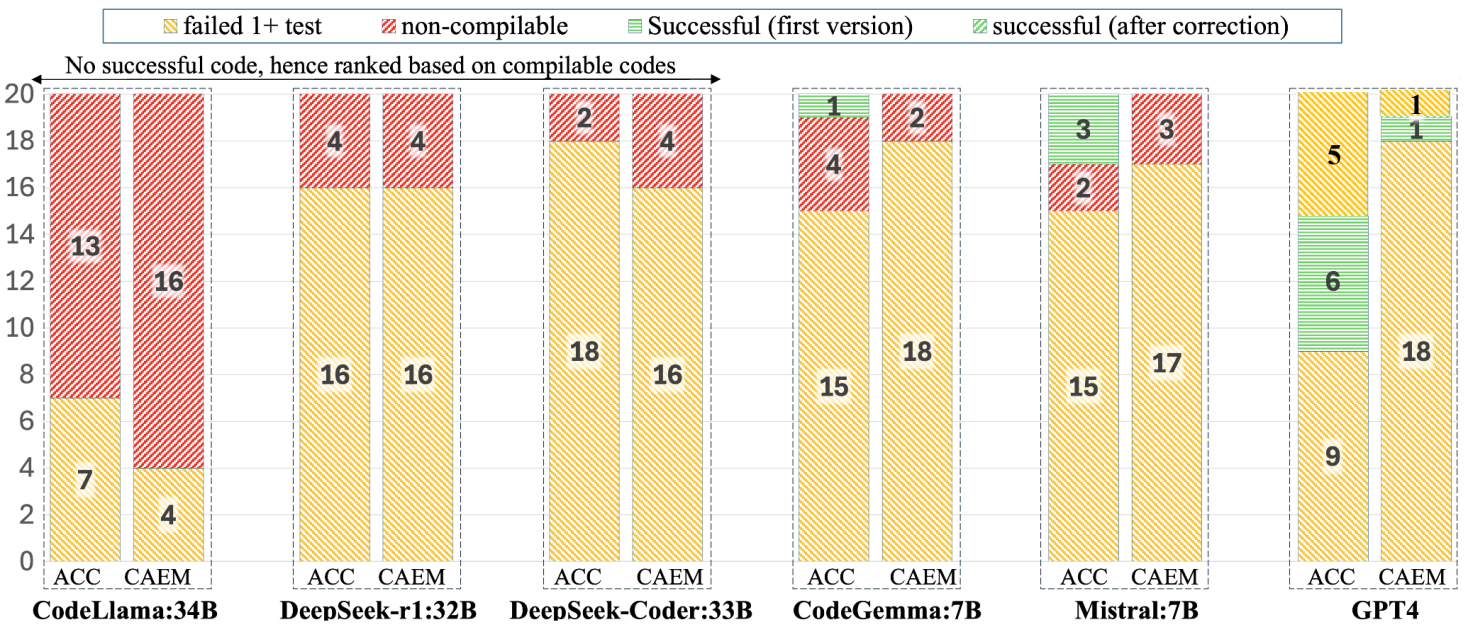
Passed

Passed

Passed

Passed

Passed


failed 1+ test
non-compilable
Successful (first version)
successful (after correction)
No successful code, hence ranked based on compilable codes
20
18
16
14
12
10
8
6
4
2
0
13
7
16
4
4
16
4
16
2
18
4
16
1
4
15
2
18
3
2
15
3
17
5
6
9
1
1
18
ACC
CAEM
CodeLlama:34B
DeepSeek-r1:32B
DeepSeek-Coder:33B
CodeGemma:7B
Mistral:7B
GPT4

Ali Nouri