2024 2nd International Conference on Foundation and Large Language Models (FLLM) | 979-8-3503-5479-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/FLLM63129.2024.10852505

# LLM-Driven Testing for Autonomous Driving Scenarios

Nenad Petrovic*, Krzysztof Lebioda*, Vahid Zolfaghari*, André Schamschurko*, Sven Kirchner*, Nils Purschke*, Fengjunjie Pan*, Alois Knoll*
*Technical University of Munich, Chair of Robotics, Artificial Intelligence and Real-time Systems, Munich, Germany
Email: nenad.petrovic@tum.de, krzysztof.lebioda@tum.de, v.zolfaghari@tum.de, andre.schamschurko@tum.de, sven.kirchner@tum.de, nils.purschke@tum.de, f.pan@tum.de, k@tum.de

***Abstract*—In this paper, we explore the potential of leveraging Large Language Models (LLMs) for automated test generation based on free-form textual descriptions in area of automotive. As outcome, we implement a prototype and evaluate the proposed approach on autonomous driving feature scenarios in CARLA open-source simulation environment. Two pre-trained LLMs are taken into account for comparative evaluation: GPT-4 and Llama3. According to the achieved results, GPT-4 outperforms Llama3, while the presented approach speeds-up the process of testing (more than 10 times) and reduces cognitive load thanks to automated code generation and adoption of flexible simulation environment for quick evaluation.**

***Keywords—autonomous driving, CARLA, Generative Pre-Trained Transformer (GPT), Large Language Model (LLM), Llama3, Model-Driven Engineering (MDE)***

## I. Introduction

In recent years, intensive progress in deep learning research and related artificial intelligence areas has both led to improvement of the existing solutions and brought many novel use cases. Computer vision and deep learning-based sensor data analysis techniques have become much more accurate, even for real-time applications. On the other side, hardware is also becoming much cheaper and powerful at the same time.

However, apart for scientific purposes and prototyping, the adoption of such novel solutions is much slower when it comes to practical utilization in industry. When it comes to industry sectors such as automotive, integration of such solutions requires comprehensive testing together with strict standardisation and compliance-related activities [1]. These additional processes involve significant amount of additional effort, slowing down the progress and innovation in many areas, especially when it comes to automotive [1, 2]. It is identified that slow adoption of novelties and long re-engineering cycles are main bottlenecks in the European automotive industry [2].

As autonomous driving capabilities of state-of-art vehicles heavily rely on AI innovations, new developments, features and upgrades cannot be immediately delivered to the end users. Apart from that exhaustive standardisation process has to be passed, many of these novel solutions potentially require extension of the existing standards in alignment with current AI usage regulations. Due to the previously mentioned factors, in practice, utilization of novel hardware and software capabilities might take up to several years in the existing automotive software development workflow to be transferred from prototypes into end-consumer products [2].

However, apart from formal standardisation and compliance, the delivery of new features and capabilities in automotive, such as autonomous driving elements also require rigorous run-time testing. In this context, the elements of safety apart from pure functionality evaluation are also taken into account. Such tests might involve the utilization of road-legal test vehicles. Therefore, the whole process becomes longer and more expensive, considering the amount of resources required for such activities. Apart from additional engineering and testing efforts, this process might also involve test drivers, legal permissions and unexpected fees in case of damage.

On the other side, during the last two years, Large Language Models (LLMs), together with other approaches of the so-called generative artificial intelligence have drawn attention of numerous researchers and enthusiasts. Among the numerous proposed adoptions, LLMs have been considered to improvement of the existing software development processes as well. Usually, the following aspects are covered in area of software development aided by LLMs [1, 3]: 1) code generation – automatic generation of executable code starting from freeform textual descriptions or potentially involving other inputs as well; 2) debugging and code analysis – identifying security flaws, bugs and other potential problems within the provided code; 3) code explanation – LLMs are able to provide verbose textual descriptions of given code excerpts; 4) code correction – LLMs were also approved to be quite useful when it comes to feedback to the user, code corrections and other suggestions for improvement of the given code.

In this paper, we explore the potential of the emerging LLMs with aim to make the testing workflow in case of future vehicle development faster, more efficient, cost-effective and convenient. We focus on generation of testing code for autonomous driving scenarios based on freeform textual scenario descriptions provided as input, including both the constraints coming from standardisation documents and user-defined requirements. When it comes to implementation, we perform the experiments leveraging the latest GPT and openly available Llama3 [4] as the underlying LLMs. On the other side, for simulation aspects we make use of graphically rich, open-source CARLA platform [5] based on Unreal Engine as simulation environment. CARLA offers solid support for variety of sensors and actuators often leveraged for autonomous driving features. In our previous work from [1], we presented initial prototype based on GPT, while Llama3-based fine-tuned solution for Object Constraint Language (OCL) rule construction from free-form text was shown in [6].

One of the advantages of the proposed approach is the fact that it takes into account both the design- and run-time aspects.

The rest of the paper is organized as follows. In the next section, we provide overview of relevant existing works in area of automotive test generation, together with works relying on LLMs even in other domains. The third section gives high-level overview of the approach implementation, including the crucial prompts and description of the underlying metamodel used as one of the prompt inputs. The fourth section describes the emergency brake case study which was used as reference for automated test generation. Moreover, the fifth section gives comparative overview of GPT-4 and Llama3 models for distinct steps, Finally, conclusion summarizes the main outcomes and considers potential future research directions.

## II. Background and Related Works

### A. Automated test generation in automotive industry

In this section, we identify the publications which tackle the topic of automated testing in automotive area, with main focus on autonomous and assisted driving scenarios.

Table I gives an insight into the identified works, summarizing their main characteristics: 1) description – general overview of the solution 2) approach – which kind of methods and techniques have been adopted in order to perform the automation of testing workflow, such as model-driven engineering, template-based code generation 3) scenario – which particular automotive use cases were targeted by the solution 4) target – which specific execution platform is targeted, is it a simulation environment or physical vehicle.

Based on the existing works shown in Table I, we can draw conclusion that there are several different approaches to automation of test workflows in automotive. Most of them incorporate some kind of model-based notation in order to define the test scenarios. Moreover, it can be identified that simulation environments and scenario-based test generation are successfully adopted as promising direction. Based on the considered works, we can identify that their main goal is to reduce the costs and complexity of autonomous driving functionality verification. However, it can be also noticed that LLMs are still not exploited enough when it comes to adoption of test generation in field of automotive.

TABLE I. Automated test generation for automotive

| Ref | Description | Approach | Scenario | Target |
|---|---|---|---|---|
| MD-AS1 [7] | Mathematical model describing scenarios as a set of parameters to construct a test scenario for simulation environment, while two component categories are identified: static components and dynamic objects. | Model-driven | Pedestrian detection | ASAM-Open Scenario<br>CARLA |
| MG [8] | Automated approach to generation of mock classes and unit tests for automotive scenarios based on metamodel specification. | Model-driven | Emergency brake | C++ with GMock/GTest |
| MTS [9] | Traffic behavior scenarios from perspective of ego vehicle are generated based on road netwrok, OD (Origin-Destination) matrix, Car-Following and Lane-Changing models. | Microscopic traffic simulation | U-turn<br>Parallel driving<br>Collision accident | SUMO – Simulation of Urban Mobility, OSM, Matlab |

### B. LLM adoption for test generation

On the other side, this section provides an overview of the existing scientific work that make use of LLMs for automated test generation. For each of the considered references, the following aspects are considered: 1) description – brief description of the approach, its main goals and purpose; 2) test scope – which level of testing granularity is targeted within the work, such as unit testing, integration testing, non-functional aspect evaluation or generation of complex scenarios for acceptance testing; 3) target – which kind of output is aimed to be produced by LLM in the considered scenario, such as Python test scripts relying on Pytest; Java code using JUnit or some other dependencies 4) model – which LLM(s) were used in the considered work.

Table II provides an overview and summarizes the relevant works on LLM-enabled test generation topic, including their main characteristics and usage.

TABLE II. LLMs for Automated Test Generation

| Ref | Description | Scope | Target | Model |
|---|---|---|---|---|
| SYS-NF [10] | LLM fine-tuning using reinforcement learning to generate system-level tests for embedded systems | System-level tests<br>Non-functional | Python<br>C | Code Llama |
| Chat Unit Test [11] | Unit test generation framework, incorporating adaptive focal context mechanism for generation-validation-repair workflow. | Unit tests | Java with JUnit | Code Llama |
| GeneUS [12] | Test case specification and user story generation based on requirements documents | Test case specification | JSON | GPT-4.0 |
| LIBRO [13] | Test generation based on bug reports as input | Unit tests | Java | GPT-3.0 based Codex |

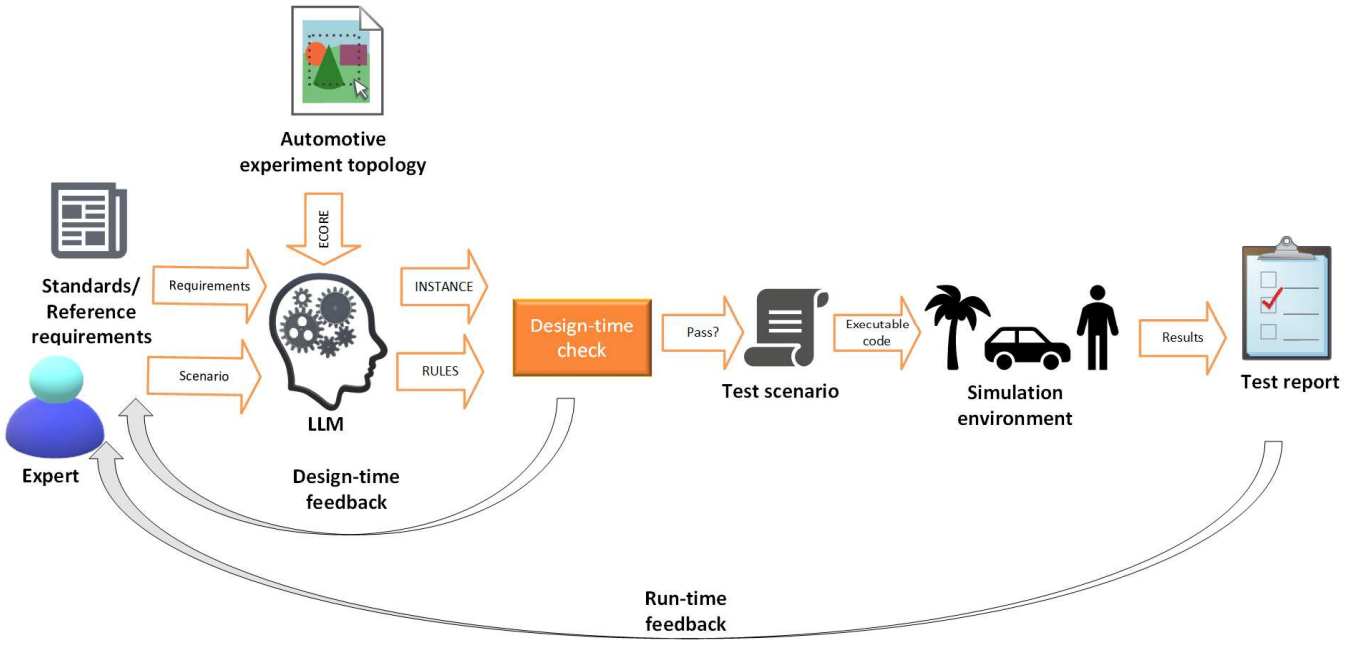


Fig. 1. Flexible LLM-enabled testing for automotive scenarios.

| Ref | Description | Scope | Target | Model |
|---|---|---|---|---|
| Test Gen-LLM [14] | Extend the coverage of the existing unit tests for corner cases | Unit tests | Kotlin | Proprietary (Meta) |

Based on the current works and more comprehensive literature reviews [15], it can be concluded that LLMs show quite strong potential in area of automated test generation. However, it can be also noticed that most of the existing solutions are considering unit testing, while none of them is specialized for automotive industry. However, there are only few solutions trying to achieve system-level and integration testing relying on LLMs. Therefore, we identify the research gap on this topic, as there are not many notable works focused on automotive domain and complex scenario-based testing. When it comes to target programming languages, it can be noticed that most of the widely used ones are covered – Python, Java and C. In this paper, we will focus on generation on test code generation aiming autonomous driving simulation platform CARLA.

## III. Implementation Overview

### A. LLM-enabled test generation workflow

Fig. 1 depicts the proposed workflow for automated testing of autonomous driving capabilities relying on LLM. As it can be seen, prompt to LLM which produces test scenario code as output considers three types of inputs: 1) scenario – user-provided textual description of test scenario; 2) requirements – standardization documents containing the reference values and specifications for particular scenarios related to autonomous driving; 3) experiment topology – metamodel describing both the vehicle and the environment, including the other vehicles, pedestrians, obstacles and positions of these objects 4) experiment template – JSON example illustrating the structure of the target code interpreted by our custom experimentation engine on top of CARLA. Therefore, in the first step, user (domain expert) specifies the free-form textual description of the scenario which is about the be tested. After that, in the next step, based on the reference requirements and provided experiment topology, a set of prompts to LLM is constructed and executed.

First, the reference requirements are transformed to Object Constraint Language (OCL) rules, using the prompt:

*Prompt 1: Generate OCL rules based on requirements {requirements} with respect to Ecore metamodel {experiment topology}* (1)

After that, the second prompt is executed against the scenario description in order to generate the Ecore model instance which is verifiable in design-time, before the test execution:

*Prompt 2: Generate Ecore model instance based on specification {scenario} with respect to Ecore metamodel {experiment topology}* (2)

After that, once the model instance and OCL rules are generated, verification of compliance whether the constraints are satisfied is performed. For that purpose, we implement engine in Java which takes metamodel, model instance and constraint rules as input, while the output is the list of unsatisfied constraints.

In case that some of the rules do not hold within the model instance, additional prompt is executed in order to generate feedback to the user which provides hints what should be corrected within the model instance:

*Prompt 3: What should be corrected in {model instance} in case that following OCL rules are not satisfied {failing rules} (3)*

Once the model instance passes all the checks in design-time, the following prompt is executed in order to generate the JSON-based experiment specification starting from model instance and JSON template for our engine on top of CARLA:

*Prompt 4: Based on {model instance} generate JSON file with respect to template {experiment template} (4)*

As outcome of this prompt execution, a JSON file is generated, which is further interpreted, as shown in the

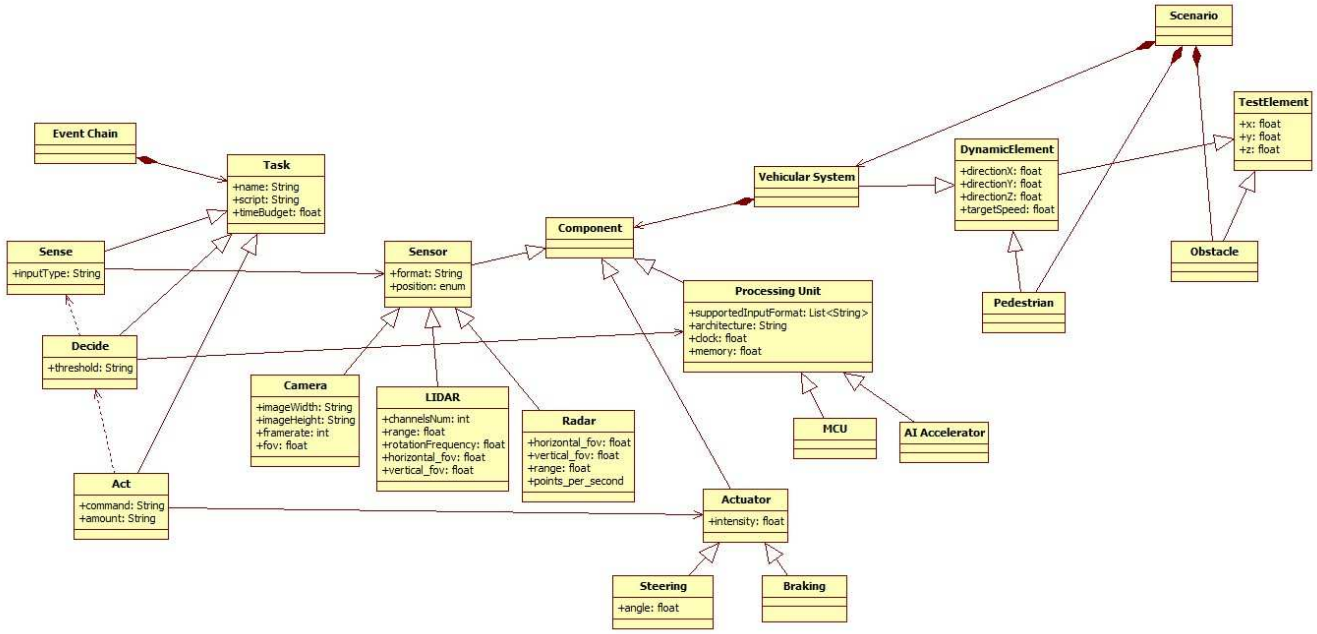


Fig. 3. Automotive experiment topology (AET) metamodel.

experiment workflow, depicted in Fig. 2. The aim of this engine is to further simplify the target code which is generated by LLM based on textual requirements, so the possibility of syntax and other errors can be reduced.

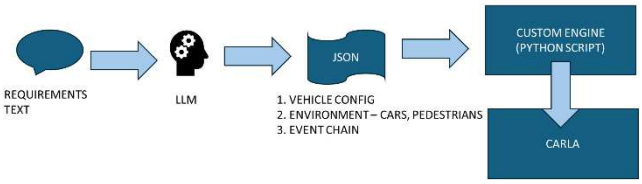


Fig. 2. Experimentation workflow based on CARLA engine.

Finally, based on test execution results and logs produced in experiment run-time, another prompt is executed to LLM in order to construct verbose feedback to the user in case that test conditions are failing. In what follows, an example of such prompt for generation of verbose feedback to the user based on test results in run-time is given.

*Prompt 5: In case of {test scenario} with outcomes {test report} what should be corrected? (5)*

Taking into account the generated feedback in run-time, user is able to re-consider the system aspects – both functional and non-functional, scenario definition and augment the provided specification in order to correct the missing aspects or update the insufficient ones.

### *B. Automotive experiment metamodel*

In this subsection, we introduce Automotive Experiment Topology (AET) metamodel which is used for specification of automotive experiments and used as one of the inputs for automated test scenario code generation. Three main aspects are covered by the metamodel: 1) environment configuration – positioning of vehicle, obstacles and pedestrians within the virtual driving environment 2) vehicle configuration – covers capabilities, functional and non-functional aspects of the vehicle which is the subject of testing within the generated experiment, including sensors, actuators and processing hardware 3) event chain – definition of processing tasks which are expected to be performed on the vehicle during the virtual drive.

The highest-level entity is test scenario. Test scenario consists of test elements, whose common properties are position coordinates *(x,y,z)* within the simulation. Furthermore, there are two categories of test elements: 1) *fixed* - static elements such as obstacles within the environment 2) *movable* – elements which are able to change their position over time, containing additional properties, such as destination coordinates and target movement speed. Additionally, there are different categories of movable elements, such as pedestrian and ego vehicle. Special category of movable elements is ego vehicle, which is unique per test scenario and represents controllable vehicle which from whose perspective the testing scenario is executed. On the other side, scenario could contain multiple obstacles and pedestrians.

When it comes to ego vehicle, our metamodel covers the following relevant aspects: 1) *sensors* – additional devices responsible for environment data acquisition, such as camera, LIDAR and radar 2) *actuators* – components which are used to change the current state of the vehicle with respect to environment, such as braking and steering 3) *processing units* – general purpose processors, microcontroller units or custom accelerators aiming specialized tasks such as machine learning capabilities. When it comes to sensors, for each of them we take into account relevant characteristics, such as camera resolution (image width and height), camera field of view (FOV), number of LIDAR channels, range and others. Additionally, positioning for each of them can be taken into account as well, so vehicle could have front, side or back sensors. On the other side, relevant target parameters are considered for actuators, together intensity of actuator reaction. For processing units, their capabilities such as clock, architecture and memory limitations are taken into account.

Finally, our metamodel also covers the aspects of on-vehicle processing, in a form of event chain. The event chain consist of tasks, which are further split into three categories: 1) *sense* – recording of environmental data, such as camera

images 2) *decide* – corresponds to tasks whose goal is to extract useful information from raw sensor data, such as obstacle detection 3) *act* – activation of actuators, such as steering or brakes in case of obstacle detection. To each processing task, a mapping to corresponding Python module is done. On the other side, there is also a correspondence between tasks and vehicle components, so sense tasks depend on sensors, decide depends on processing units, while act depends on vehicle's actuator components. Finally, each of these tasks also has time budget parameter, which represents the longest allowed processing time, that can be further leveraged for insights into non-functional constraint satisfaction. Fig. 3 depicts the previously described metamodel. The implementation of this metamodel was done relying on Ecore which is a part of Eclipse Modeling Framework (EMF) [16].

## IV. Case Study

As a case study for test generation, we consider emergency brake scenario, based on regulations from [17] and publicly available reference requirements based on [18]. The test scenario is generated based on the following textual description:

*Ego vehicle is Tesla Model 3, the obstacle lead vehicle is Toyota Prius.*

*The initial position (transform) of the ego vehicle is x=-67.25, y=27.93.*

*The initial rotation (transform) of the ego vehicle is yaw=0.16.*

*The destination of the ego vehicle is 40 meters in front of its initial position.*

*The forward speed of the ego vehicle is 20m/s.*

*The ego vehicle should ignore other vehicles, but should not ignore traffic light.*

*The initial position (transform) of the lead vehicle is 20 meters in front of the ego vehicle.*

*The initial rotation (transform) of the lead vehicle is the same as ego vehicle.*

*The forward speed of the lead vehicle is 0m/s.*

*The ego vehicle has a front camera with resolution of 1920x1080, and field of vision 90.*

*The ego vehicle has a front LIDAR with horizontal FOV: 9.5° and vertical FOV: 2.1° – 7°.*

*There is one pedestrian with initial position is x=-35.00, y=27.96.*

*Pedestrian walks towards direction of x=0.0, y=-1.0.*

When it comes to constraints, it is checked whether vehicle contains at least one camera and one LIDAR.

The screenshots from the CARLA simulation environment while running the emergency brake scenario code based on our JSON configuration interpretation are shown in Fig. 4.

Fig. 4. Generated CARLA simulation experiment.

## V. Experiments and Evaluation

When it comes to evaluation, we compare two LLMs: GPT-4 and Llama3 8B instruct. When it comes to execution environment, the first one relies completely on OpenAI's cloud infrastructure due to high hardware demands in terms of GPU power. For the second one, we rely on Hugging Face's library in Python and it is deployed within free version of Google Colab's environment with 15GB VRAM T4 GPU. It is considered as more flexible solution, regarding its lower resource demand and open-source nature, making it locally deployable.

The following aspects of evaluation are taken into account for each of the main steps are taken into account: 1) execution time – how much time is needed to execute the step; 2) tokens – average number of consumed tokens for given step 3) error rate – percentage of wrong output generated by LLM based on 10 subsequent runs; 4) manual – the estimated time required to perform the step manually without relying on the proposed workflow. When it comes to hyperparameter values, *temperature* value 0.1 was used, while nucleus sampling parameter *top_p* was set to 0.9. The choice of such parameter values leads to almost deterministic results with small degree of variety, which is suitable due to nature of code generation problem itself (correct syntax and modeling rules applied).

TABLE I. Question Answering Accuracy and Execution Time Evaluation

| Step | GPT-4 / Llama 3 | Exec time [s] | Tokens | Error rate [%] | Manual [s] |
|---|---|---|---|---|---|
| OCL rule generation | | 0.11 | 760 | 10% | 300 |
| | | 13.75 | 842 | 30% | |
| Model instance creation | | 0.16 | 1462 | 20% | 1200 |
| | | 94.83 | 1528 | 30% | |
| JSON configuration generation | | 0.041 | 1081 | 10% | 2400 |
| | | 23.53 | 1122 | 20% | |
| Feedback | | 0.02 | 623 | 10% | 200 |
| | | 16.42 | 667 | 20% | |

Based on the obtained results, it can be noticed that GPT-based solution was faster, as expected, considering the fact that Llama3 was run as smaller variant suitable for local deployments on less demanding hardware, while GPT deployment relied on OpenAI's powerful cloud infrastructure. Additionally, it can be seen that GPT's error rate is lower than Llama3, considering that GPT model is much larger with respect to number of parameters compared to Llama3 variant

used in the experiment. Additionally, it can be noticed that consumption of tokens in case of Llama3 was slightly larger. Moreover, regarding the achieved error rates, it is observable that model instance creation has the highest error rate for both LLMs. This fact can be explained by sensitivity of XML-alike instances to variations, as their structure is strictly defined by metamodel schema, while the number of generated tokens is largest in this case as well, increasing the probability of error.

When it comes to speed up of distinct steps, compared to manual procedures performed by experienced expert, the adoption of LLMs reduces the order of magnitude of time needed from minutes to seconds, resulting in practice with acceleration of up to more than 10 times in case of Llama3.

## VI. Conclusion and Future Works

This paper explores the adoption of novel LLMs for purpose of automated testing in area of automotive. Based on our results, it can be concluded LLMs have huge potential when it comes to automated generation of tests in automotive domain. The benefits of such approach are obvious, as such solutions reduce the time needed for test creation more than 10 times.

Our research demonstrates that commercial GPT-4 still provides more accurate results than Llama3 out of the box. However, further fine-tuning and optimization of Llama3-based LLMs has huge potential to achieve similar performance, close to GPT-4 in terms of accuracy, as shown in our work focused on OCL rule generation [6]. Apart from that, one of the main advantages of Llama3-based solutions is the ability to deploy them locally, which is highly beneficial for automotive industry users, as their data would not be exposed and would remain within the organizational boundaries. Moreover, such kind of deployment would not involve additional costs on per-token basis like in case of OpenAI's ChatGPT. Therefore, fine-tuning Llama3-based LLMs for all the steps, including model instance creation and code generation seems like promising future research direction. Another aspect that is aimed to be covered in our further works is including the asserts as part of generated code, that would give the ability to verify whether certain processing steps satisfy the time budget constraints. Finally, our plan is to integrate the proposed toolchain with physical vehicle testbench building upon [19], which would enable flexible experimentation with real vehicle's digital twin in simulated environment.

## Acknowledgment

This research was funded by the Federal Ministry of Education and Research of Germany (BMBF) as part of the CeCaS project, FKZ: 16ME0800K.

## References

[1] N. Petrovic et al., "Synergy of Large Language Model and Model Driven Engineering for Automated Development of Centralized Vehicular Systems", technical report, Technical University of Munich, pp. 1-15, 2024. https://arxiv.org/pdf/2404.05508

[2] S. Solmaz et al., "Novel Hybrid-Testing Paradigms for Automated Vehicle and ADAS Function Development", Towards Connected and Autonomous Vehicle Highways, EAI/Springer Innovations in Communication and Computing. Springer, 2020, pp. 193-228. https://doi.org/10.1007/978-3-030-66042-0_8

[3] N. Petrović and I. Al-Azzoni, "Model-Driven Smart Contract Generation Leveraging ChatGPT", Proc. of Int. Conf. Syst. Eng. (ICSEng), 2023, LNNS 761, pp. 387-396. https://doi.org/10.1007/978-3-031-40579-2_37

[4] Introducing Meta Llama 3: The most capable openly available LLM to date [online], available on: https://ai.meta.com/blog/meta-llama-3/, last accessed: 06/10/2024.

[5] CARLA: Open-source simulator for autonomus driving research [online], available on: https://carla.org/, last accessed: 27/07/2024.

[6] F. Pan et al, "Generative AI for OCL Constraint Generation: Dataset Collection and LLM Fine-tuning", ISSE 2024, pp. 1-8, 2024.

[7] A. Lyamani, T. Hajji, I. Elhassani, T. Masrour, "Scenarios for ADAS Testing: Modeling and Design", ICDTA 2022, Lecture Notes in Networks and Systems, vol 454. Springer, pp. 753-762, 2022. https://doi.org/10.1007/978-3-031-01942-5_75

[8] N. Petrović, M. Radenković, S. Cvetković, D. Rančić: "Model-driven automated gMock test generation for automotive software industry", XV International SAUM, pp. 1-4, 2021.

[9] B. Yue, S. Shi, S. Wang and N. Lin, "Low-Cost Urban Test Scenario Generation Using Microscopic Traffic Simulation", IEEE Access, vol. 8, pp. 123398-123407, 2020. https://doi.org/10.1109/ACCESS.2020.3006073

[10] D. Schwachhofer et al., "Training Large Language Models for System-Level Test Program Generation Targeting Non-functional Properties", 2024 IEEE European Test Symposium (ETS), 2024, pp. 1-4. https://doi.org/10.1109/ETS61313.2024.10567741

[11] Y. Chen et al., "ChatUniTest: A Framework for LLM-Based Test Generation", FSE 2024, pp. 572-576, 2024. https://doi.org/10.1145/3663529.3663801

[12] T. Rahman, Y. Zhu, "Automated User Story Generation with Test Case Specification Using Large Language Model", preprint, pp. 1-10, 2024. https://arxiv.org/abs/2404.01558

[13] S. Kang, J. Yoon and S. Yoo, "Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction", 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), 2023, pp. 2312-2323. https://doi.org/10.1109/ICSE48619.2023.00194

[14] N. Alshahwan et al., "Automated Unit Test Improvement using Large Language Models at Meta", FSE 2024, pp. 185-196, 2024. https://doi.org/10.1145/3663529.3663839

[15] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang and Q. Wang, "Software Testing With Large Language Models: Survey, Landscape, and Vision" in IEEE Transactions on Software Engineering, vol. 50, no. 4, pp. 911-936, 2024. https://doi.org/10.1109/TSE.2024.3368208

[16] Eclipse Modeling Framework [online], available on: https://eclipse.dev/modeling/emf/, last accessed: 06/10/2024.

[17] UN Regulation No. 157 - Automated Lane Keeping Systems (ALKS) [online], available on: https://unece.org/transport/documents/2021/03/standards/un-regulation-no-157-automated-lane-keeping-systems-alks, last accessed: 12/09/2024.

[18] AVC Consortium, Technical Reports & More from AVCC [online], available on: https://avcc.org/documents/, last accessed: 12/09/2024.

[19] S. Kirchner et al., "AUTOFRAME: Software-Driven Integration Framework for Automotive Systems", ITSC 2024, pp. 1-6, 2024.