Integrating LLMs with QC-OpenDRIVE: Ensuring Normative Correctness in Autonomous Driving Scenarios

Julian Müller(✉), Thies de Graaff, and Eike Möhlmann

German Aerospace Center, Institute of Systems Engineering for Future Mobility, Oldenburg, Germany
{julian.mueller,thies.degraaff,eike.moehlmann}@dlr.de

Abstract. This paper investigates on the integration of Large Language Models (LLMs) with the `QC-OpenDRIVE` framework in order to generate syntactically and semantically correct OpenDRIVE files. OpenDRIVE files play an important role in the scenario-based validation of autonomous driving systems as they define the static part (e.g. road layout) on which the function are validated. While LLMs excel at generating code or similar tasks which mostly needs to be syntactically correct, the validation of semantic, especially normative, correctness remains challenging. To ensure norm-adherent correctness of generated OpenDRIVE files this paper proposes an integration of a feedback-loop with LLMs and `QC-OpenDRIVE`. While LLM allow to easily generate different road layouts, they often show issues like missing or unconnected roads or improper continuity. To address this issue, we have implemented E.5.9.1 to ensure geometric continuity between connected roads, which is a key contribution of this paper.

State-of-the-art models are evaluated on three tasks to create OpenDRIVE road networks and validate the results featuring the feedback-loop. Results show that models leveraging Retrieval Augmented Generation (RAG) or internal reasoning and using the feedback loop can generate syntactically and semantically valid outputs after iterative corrections. However, challenges remain to prompt complex scenarios and tasks, especially following geometric rules without explicit feedback. The results demonstrate the necessity of domain-specific normative validation frameworks to prepare the use of LLMs for safety-critical applications. They can be used to enable scalable generation of edge-case scenarios while ensuring compliance with industry standards. This work bridges the gap between automated scenario generation and rigorous validation of reliable autonomous driving systems.

Keywords: Large Language Models · OpenDRIVE · QC-OpenDRIVE · Scenario Generation · Semantic Validation · Autonomous Driving

B. Steffen (Ed.): AISoLA 2025, LNCS 16220, pp. 103–116, 2026.
https://doi.org/10.1007/978-3-032-07132-3_7

1 Motivation

Beside designing and implementing autonomous driving systems, testing and guaranteeing the safety of such a system is a necessary and non-trivial task. Wachenfeld and Winner [26] state that a pilot for the German Autobahn would need 6,62 bn. driving kilometers to be twice as good as a human driver with 50% certainty. This way of obtaining a safety prove for every new vehicle on the Autobahn is unrealistic and also economically not feasible. A promising approach is virtual testing, where simulations are used rather than real world tests. As the number of potential evaluations are uncountable, scenarios are used as a guiding structure. Such a scenario defines the relevant static and dynamic parts of the vehicle's environment for example using the 6-Layer Model developed in the PEGASUS project [23]. Further, by focusing on dangerous scenarios a driving pilot can be tested more carefully in critical situations without risking property or lives [8,13].

However, creating tailored edge-cases for testing can be time consuming. In the last years, Large Language Models (LLMs) are used for several text-generation tasks. Used in vibe-coding, documentation and summarizing the abilities are used more and more in everyday tasks to save time and generate loads of text in a very short amount of time. By learning stochastic connections in language these models can generate many types of textual output. Using this ability to write test-scenarios could potentially save time and automatically create systematic test-cases tailored to the needs of the developers and safety engineers.

Given the enormous amount of code available on the internet (for example github.com), this data can be used to train LLMs and to create benchmarks to evaluate these models. Current benchmarks mostly evaluate the models through syntactic correctness and functional accuracy. Though there are still challenges in evaluating the models ability to generate efficient, readable and maintainable code [16,17,31].

Even in subjective and creative tasks like story-telling or legal writing the use of LLMs gains popularity. But given its subjective and inherent complexity, its validation do need expert supervision. Compared to code generation evaluating expert models on different domains need interdisciplinary experts in machine learning and for example legal expertise to ensure correctness [18,21].

Similar to generating working code in a programming language, a scenario can be generated in a given specification-language, like OpenDRIVE [4] by the Association for Standardisation of Automation and Measuring Systems (ASAM). Checking for the correct syntax can be easily achieved through the XML-specification, but checking for their semantic correctness is not done yet. To evaluate the content of the output the problem occurs that experts do need to check the correctness and plausibility, like in legal writing. Especially for never learned prompts, it is known that LLMs generalize poorly and can hallucinate or have difficulties with the semantic content and logical reasoning [15,28,31]. Therefore, it is important, that the output of LLMs is evaluated, such that, in our application, can be used to generate test scenarios.

One part of the semantic correctness is *normative correctness*, i.e., correctly adhere to a given set of norms, rules, or regulation. OpenDRIVE files need to follow a given set of rules defined by the standard. For example Schwab et al. [24] checked for gaps between the lanes at road crossings after recognising them in parametric road space models. The connection of geometric shapes (e.g. the roads), but also others geometric, topological or semantic rules are defined by OpenDRIVE rules. Until recently they were not checked at all. To guarantee the normative quality ASAM published the QC-Framework to check OpenDRIVE and OpenSCENARIO `XML`-data. `QC-OpenDRIVE` [6] uses this framework and implements some rules to ensure their correctness.

Generated OpenDRIVE-Scenarios by LLMs can be evaluated with the quality-checker. Key contributions of this paper are:

1. Integrating LLMs into a feedback-loop by evaluating generated outputs using `QC-OpenDRIVE` (Sect. 3),
2. Implementation of the rule E.5.9.1 (road.geometry.contact_point) for QC-Open-DRIVE (Sect. 4),
3. Evaluating correctness and content of generated outputs (Sect. 5).

2 Related Work

Evaluating code via benchmarks, like OpenAI's HumanEval [10] or Google Research's Mostly Basic Python Problems (MBPP) [7], is common practice. Some LLM-agents are also capable of using compilation errors of generated code to refine their output (e.g. GitHub Copilot agent mode [20]). But more subjective tasks, like evaluating the semantic correctness of different tasks, require experts to evaluate the model outputs [16–18,21].

The generation of concrete scenarios using LLMs is subject to prior works [9,22,32]. But none of these methods use a unified output syntax (e.g. OpenSCENARIO) to generate the data. This renders a standardized evaluation for LLM-generated scenarios much more complex or even impossible. Xiao et al. [29] describe scenarios in multiple logical steps, which could be used to evaluate the output by the logical definitions.

ASAM [3] develops standards for the development of autonomous driving systems. Members of the association are international car manufacturer, suppliers and research institutions. Use-cases like the development, testing and evaluation of driving systems are standardized by ASAM e.V. Therefor different data-formats, -models, protocols and interfaces are defined, establishing an easier interchange of data and tools [3,4].

ASAM OpenDRIVE [4] defines syntax and rules to describe road networks using Extended Markup Language (XML). OpenDRIVE is mainly concerned with describing the geometry of roads, lanes and objects like lane markings or signals. Such definitions can be based on real road-data or be synthetically generated.

Eisemann und Maucher [12,13] generate OpenDRIVE-Maps from LiDAR pointclouds. Segments of these points are combined and translated to OpenDRIVE. Becker et al. [8] define a logical description of a road-map, which is

translated to OpenDRIVE. Both approaches do not use natural language processing.

Regarding the validation of OpenDRIVE, Schwab et al. [24] implemented simple rules, which mainly check for gaps between lanes. This uncovered gaps at road crossings in their OpenDRIVE data.

3 Integration of LLMs with QC-OpenDRIVE

The OpenDRIVE file format is based on Extensible Markup Language (XML). ASAM provides an XML schema that defines the structure of a valid OpenDRIVE file. The main OpenDRIVE document provides in-depth explanations about every aspect of the file format. It also defines a set of rules, that an OpenDRIVE file has to conform to. These rules do not cover syntactical correctness but aim to result in logical correctness of the described road network. E.g. rule E.5.9.5 (road.geometry.paramPoly3.length_match) requires that the actual curve length [of a road], as determined by numerical integration over the parameter range, should match [the parameter] @length [4]. Recently, ASAM published quality checkers for different standards, including OpenDRIVE. This framework called `QC-OpenDRIVE` [6] checks the syntax using the OpenDRIVE XML schema as well as a subset of the rules defined in the OpenDRIVE standard.

First experiments indicate that LLMs like GPT-4 can generate an OpenDRIVE file from simple prompts, when using the ChatGPT interface. But even the generation of very simple road networks composed of one or two roads often results in incorrect output. These errors can mostly be attributed to a violation of the OpenDRIVE XML schema, but also violations against rules of the OpenDRIVE standard do occur. It follows, that `QC-OpenDRIVE` should be able to detect such mistakes of an LLM. In such an event, the error report from `QC-OpenDRIVE` should help to find the faulty part of the generated OpenDRIVE and correct the mistake.

The idea of this feedback-loop is visualized in Fig. 1. Starting from a prompt to an LLM in natural language, an initial OpenDRIVE map will be generated and checked via `QC-OpenDRIVE`. In case of a detected error, the error report is given to the LLM to correct the mistake. This procedure can iteratively be repeated until all violations to the XML schema and the implemented rule set of `QC-OpenDRIVE` are eliminated. If not all mistakes can be corrected, the loop might either be terminated after a fixed number of iterations or when it is detected, that the same errors appear repeatedly. Using this approach, OpenDRIVE files can be generated automatically from natural language, while being syntactically correct and adhering to the implemented rules.

3.1 OpenDRIVE Generation Tasks for the Models

To identify the current abilities of different technologies and architectures in generating OpenDRIVE files, we tested three distinct models. They are given the same prompts of three tasks to generate simple road networks using a simple

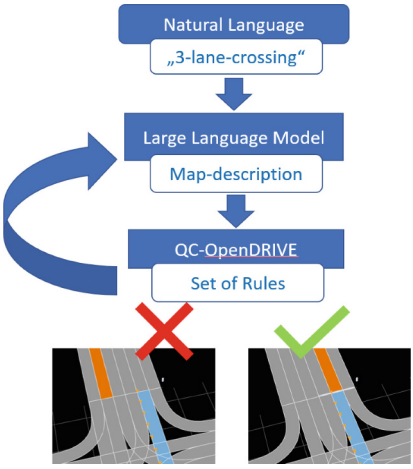

Fig. 1. Idea of a feedback-loop to correct invalid OpenDRIVE files generated by a LLM. The idea proposes to check its correctness by a given set of OpenDRIVE rules. With the error, the LLM hopefully can generate a correct output.

Chain-of-Thought (Cot) [27]. The validity of each output after one prompt is evaluated with the external tool QC-OpenDRIVE (see Fig. 1). The models are presented in Table 1 and represent state-of-the-art models leveraging Retrieval Augmented Generation (RAG) [19] or reasoning [11].

Table 1. List of used models for evaluation with their parameters and properties. Llama uses RAG, Qwen using reasoning and the mistral model uses none.

Model	Parameters	Reasoning [11]	RAG [19]
Mistral Large Instruct [25]	123B	✗	✗
Meta Llama 3.1 [14]	70B	✗	✓
Qwen 3 [30]	235B	✓	✗

The chosen models are not fine-tuned to generate OpenDRIVE examples and are chosen to show the current abilities to correct its output with given feedback from QC-OpenDRIVE. As a baseline, the model Mistral Large [25] is chosen and used without any additional feature like reasoning or RAG. Meta's Llama 3.1 [14] features less parameters, but is using RAG to access the OpenDRIVE 1.7.0 documentation [2] as a PDF-file. Lastly, the most powerful tested model Qwen 3 [30] is trained to reason before giving the answer to the prompts. This model is added, as it may have been trained on some rules and might use the reasoning to validate the steps to make less errors. Each model was integrated into the feedback-loop with `QC-OpenDRIVE` to augment the models with the intended error correction capabilities.

We designed a small benchmark of three tasks, each composed of multiple prompts (see Table 2). To assist the models CoT is used. Wei et al. [27] have shown, that giving multiple step-by-step prompts can assists the models ability to generate more accurate responses. The first prompt in each task tests the capabilities of the models to generate an OpenDRIVE file purely from natural language. The additional prompts per task are intended to test the editing capabilities based on a already generated OpenDRIVE and an editing instruction. An editing instruction is only performed, if the previous prompt did not result in an error. Each task is generated in a new chat, so that the previous task is not in the context window.

Table 2. Prompts used for the different tasks. The system-prompt is the same for every model and task. Each prompt has a specific identifier to see the task and prompt id. For example the first task contains the three different prompts P1.1, P1.2 and P1.3. They are used in the same chat and used iteratively.

Task	ID	Prompt
System-prompt	-	You are a helpful assistant to create OpenDRIVE xodr-files. Be sure to use the correct schema-format. The output has to strictly follow the defined xml-schema - be sure to include the correct required attributes! But leave unnecessary elements and attributes out. For example leave the predecessor out, when there is no predecessor road. Use this header for the version 1.7.0 of OpenDRIVE: ‘‘‘ <?xml version="1.0" standalone="yes"?> <OpenDRIVE> <header revMajor="1" revMinor="7" name="" version="1.00" date="Wed Aug 14 11:25:56 2024" north="0.0000000000000000e+00" south="0.0000000000000000e+00" east="0.0000000000000000e+00" west="0.0000000000000000e+00"> </header> ‘‘‘ Resume the instructed content with the provided header.
Task 1	P1.1	Create a straight 10m road with a 2.5m wide lane in each direction.
	P1.2	Change the road, so that is 50m long.
	P1.3	Add one lane.
Task 2	P2.1	Create two roads. Each should have one left lane and one right lane.
	P2.2	Rotate one road for 180 degrees.
	P2.3	Add a link to connect both roads.
Task 3	P3.1	Create two roads. Each should have two left lanes and one right lane.
	P3.2	Add a link to connect both roads.

The chosen tasks include simple instructions to generate one or two roads, to modify and connect them. We currently refrain from more complex instructions, since experiments with different models have shown their inabilities in creating maps with three or more roads. We also noticed, that plenty of errors originate

from an incorrect header in the generated OpenDRIVE files, which keeps meta information like the version or the global position of the map. To eliminate this error and focus at the content of the tasks, the header is provided as part of the system prompt.

4 Checking Normative Semantics

The rules from the ASAM standard OpenDRIVE are implemented in github.com/asam-ev/qc-opendrive/tree/main/qc_opendrive/checks. Since only a subset of all rules defined in the standard are currently implemented in `QC-OpenDRIVE`, an allegedly valid OpenDRIVE file might violate an unimplemented rule from the standard. And without an error report, such an invisible error cannot be corrected in the feedback-loop between an LLM and `QC-OpenDRIVE`. In our experiments, we often encountered a violation of rule E.5.9.1 (`road.geometry.contact_point`) by the LLMs. Thus, we implemented this rule to extend the rule set of `QC-OpenDRIVE`.

In `QC-OpenDRIVE`, the logic of a rule is implemented in the method `check_rule` of a new checker module. By registering this checker module in the `main.py`, the implemented method is automatically called when running `QC-OpenDRIVE`. For testing the implementation of a rule itself, it should also be tested using pytest. Test-`xodr`-files are stored in tests/data/ and added in the intended check-file in tests/.

4.1 OpenDRIVE Terminology

Before discussing rule E5.9.1 and its implementation in more detail, we need a rough understanding of the affected OpenDRIVE structures.

The geometry of the course of a road is defined in a `<planView>`. The line it follows is called the road reference line (see blue arrows in Fgure 2), from which the lanes of the road extend to the sides. Two roads can be connected with the `<link>` element, where the connection is defined by either the predecessor, when the other road connects to the start-point, or the successor, when the road connects to the end-point of the road reference line. The attribute *contactPoint* states, which end of the road reference line of the other road is connected to either the predecessor or successor point [1,4].

4.2 Rule E5.9.1: `road.geometry.contact_point`

The rule E.5.9.1 (road.geometry.contact_point) is performing a geometry check on a road definition. It states:

`Rule 5.9.1` If two roads are connected without a junction, the road reference line of a new road shall always begin at the `<contactPoint>` element of its successor or predecessor road. The road reference lines may be directed in opposite directions.

Each road defines its predecessor- and successor-elements in the `<link>` element of each connecting road. The point of contact is obligatory and given by the parameter *contactPoint*. The road reference line of both roads need to geometrically touch the end or start of the other contact point. With this rule, it has to be defined in the element, using the contact point `"end"` or `"start"`. The correct usage is illustrated in Fig. 2.

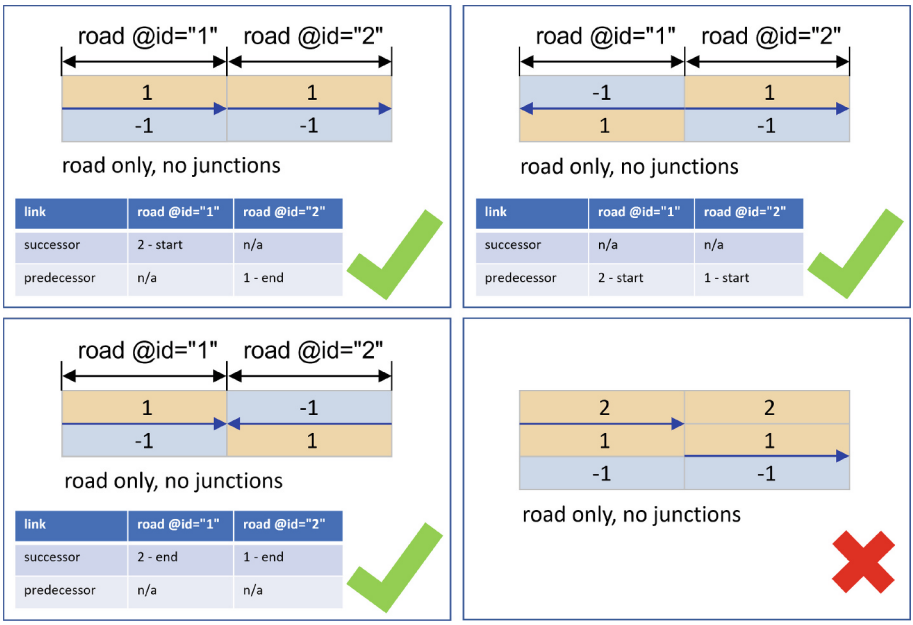

Fig. 2. Valid and invalid connections of contact-points from two roads. From OpenDRIVE [5].

In the upper left image the road with the ids "1" and "2" are connected. The successor link element of the road "1" is the road "2", because the end of the road reference line, points at the start of the reference line of the road "2". Because of this, the contact point of the successor of road "1" is the start of the road "2". On the other side, the contact point of the predecessor from road "2" is the end of road "1". When both road reference lines point to the opposite directions they are each other predecessors and have contact points at the start of each other. Lastly, when both road reference lines point to each other, they are successor elements with contact points at the end of each other. The bottom-right images illustrates the invalid case, when the contact points are not connected.

To check this rule, the geometric coordinates of the contact points and the start/end of the road reference lines have to be compared. To account for floating point errors, equality is assumed if the distance of coordinates is below a threshold value $\epsilon = 10^{-6}$. When the predecessor is checked, the start point of

the current road reference line is used and when the successor is checked, it needs to be the end point. Depending on the parameter *contactPoint* of the `<link>` of the successor or predecessor, the contact point needs to equal the start or end point of the road reference line of the other road.

A pull-request with the implementation and tests is published for the official ASAM QC-OpenDRIVE github-repository. The implementation can be found at github.com/asam-ev/qc-open-drive/pull/126.

5 Evaluation

In the last section the models and prompts were presented. The details and tries are shown in Table 3. The table is ordered by the tasks. Mistrals model failed to generate the first prompt, so this model was not used for the further tasks.

Two of the three tested models, namely Qwen 3 and Llama 3.1, could use `QC-OpenDRIVE` as a feedback-loop. The mistral model could not fix its schema errors and did not return the corrected XML-output, while both other models were able to create correct OpenDRIVE files.

In the first task (P1.1, P1.2 and P1.3) the models were asked to create one road and modify it. Qwen 3 and Llama completed this task. Qwen 3 needed three attempts to fix the returned schema-errors. Llama 3.1 used RAG and finished the complete first task without errors. When investigating the output, it shows some minor differences to the intended task as it added more than one lane.

The second task (P2.1, P2.2 and P2.3) was to create two roads, rotate one and to connect both roads. Llama was not able to complete the task. In the first prompt it had one schema error, which could be fixed with the error message in the first try. After the second prompt it returned only the changes to the road and needed to be asked to return the full OpenDRIVE file. It could not fix the schema errors for the third prompt. Qwen 3 completed this task, while only having one error, missing the implemented rule E.5.9.1 (road.geometry.contact_point). Models might miss this rule and the resulted XML would be incorrect.

The last and third task was similar to the second one. Two roads have to be generated in one prompt. Both models needed three tries to generate a correct schema. As in the second task, Llama first printed only the changes, but after asking it to print the whole XML, the output was correct. Qwen 3 needed three tries for the second prompt. First the schema was invalid and in the second try the contact point of the link was invalid as in the second task. It could fix this error.

Using the feedback-loop both models could complete the tasks in a few minutes. Writing small OpenDRIVE files by hand would, depending on the knowledge, take similar or more time. But the strength of this models can be shown when different scenarios can be generated automatically. With the help of the feedback-loop, distinct scenarios can be described and then generated by the models.

Table 3. Validation results after each iteration (step) using different models for the generation of OpenDRIVE files in the predefined tasks. In each step a validation with `QC-OpenDRIVE` as well as a human inspection was performed and summarized here.

ID	Model	Step	`QC-OpenDRIVE` Validation	Human Inspection
P1.1	Mistral	1-6	Schema Error: invalid lane id	Alternates the same mistake for positive and negative lane-ids. Could not fix the error.
	Llama	1	No QC-OpenDRIVE Error.	Output as expected for the prompt.
	Qwen	1-2	Invalid schema	-
		3	No QC-OpenDRIVE Error.	Output as expected for the prompt.
P1.2	Llama	1	No QC-OpenDRIVE Error.	Two lanes with a width of 0cm on one side of the road reference line.
	Qwen	1	No QC-OpenDRIVE Error.	Output as expected for the prompt.
P1.3	Llama	1	No QC-OpenDRIVE Error.	Added three lanes. Two on the opposite driving direction.
	Qwen	1	No QC-OpenDRIVE Error.	Output as expected for the prompt.
P2.1	Llama	1	Schema Error.	-
		1	No QC-OpenDRIVE Error.	Output as expected for the prompt.
	Qwen	1	No QC-OpenDRIVE Error.	Output as expected for the prompt.
P2.2	Llama	1	-	Printed only the changed lines. Asked to print the whole XML again.
		2	No QC-OpenDRIVE Error.	Output as expected for the prompt.
	Qwen	1	No QC-OpenDRIVE Error.	Output as expected for the prompt.
P2.3	Llama	1-4	Schema Error	Could not fix the error.
	Qwen	1	Invalid rule road geometry contact point.	-
		2	No QC-OpenDRIVE Error.	Output as expected.
P3.1	Llama	1-2	Schema Error	-
		3	No QC-OpenDRIVE Error.	Output as expected for the prompt.
	Qwen	1-2	Schema Error	-
		3	No QC-OpenDRIVE Error.	Output as expected for the prompt.
P3.2	Llama	1	-	Printed only new roads. Manually asked again to print whole XML.
		2	No QC-OpenDRIVE Error.	Output as expected by the prompt.
	Qwen	1	Invalid schema.	-
		2	Invalid rule road geometry contact point.	-
		3	No QC-OpenDRIVE Error.	Output as expected for the prompt.

6 Conclusion

Natural language processing with LLMs might dramatically change many tasks typically done by humans. This includes describing legal writing, text summary generation, or even source code generation. Depending on the task and data availability, specialized experts need to evaluate the model, especially when data is sparse or the topic is of subjective nature. In automated driving, scenarios are a common tool for describing allowed and forbidden interaction of the vehicle and its environment (including vulnerable road users). Obtaining relevant scenarios covering the operational design domain is a huge challenge especially if they need to be handcrafted. In this paper, we investigate the automatic generation of road networks for such driving scenarios analogously to the generation of source code. We generate road networks as OpenDRIVE with different LLMs. By using OpenDRIVE, which is a standardized language, models can be compared and automatically checked for issues since a set of rules exists. We propose to integrate the LLM-based road network generation with automatic rule-checking in a feedback-loop in order to fix errors introduced by the LLM.

To demonstrate the feasibility of our feedback-loop, three LLMs are integrated with the open-source ASAM Quality Checker `QC-OpenDRIVE` for OpenDRIVE files, to iterative validate and correct the OpenDRIVE files generated by the LLMs. While most models have difficulties generating valid OpenDRIVE, by leveraging reasoning and RAG the tested models could generate syntactically valid road networks, especially when combined with `QC-OpenDRIVE` feedback. We show that syntactic schema errors as well as normative errors could be fixed using the feedback-loop. Unfortunately, not all rules of OpenDRIVE are yet implemented in `QC-OpenDRIVE`. Therefore, we extended `QC-OpenDRIVE` by implementing the rule E.5.9.1 `road.geometry.contact_point` which checks for road continuity at their contact points with other roads. This rule solved a major issue, we identified during our experiments. While Llama 3.1 failed to produce correct contact point in the second task and Qwen 3 in the third, they were able correctly connect roads given the feedback. Because LLMs have problems with implicit domain knowledge (e.g. the contact point), this does not only show practical utility in identifying invalid road connections as a critical requirement and the necessity of having validation tools but also shows the automatic feedback and correction is possible, leaving the creative part to the LLM and the corrective part to a rule checker.

The results show that human inspection is still needed, as there may be differences between the intended results and the generated output. In the future, the set of OpenDRIVE rules should be extended even further, as this will allow to correctly generate more complex OpenDRIVE files and allow automated tests with no human inspection. Moreover, this enables the training of specialized models and to ultimately create scenarios for testing autonomous driving systems in edge-cases. In the field of test-case design, such a automated LLM-based pipeline with a feedback-loop could enable hybrid approaches like using simulation and reinforcement learning to test edge-cases with minimal manual effort in validating the correctness and to meet industry standards.

By integrating LLMs with validation frameworks like `QC-OpenDRIVE`, this study paves the way for scalable, standardized, and semantically sound scenario generation, essential for the safe deployment of autonomous driving systems.

Acknowledgements. The research leading to these results is funded by the German Federal Ministry for Economic Affairs and Energy within the project "NXT GEN AI METHODS âĂŞ Generative Methoden für Perzeption, Prädiktion und Planung". The authors would like to thank the consortium for the successful cooperation.

References

1. ASAM: OpenDRIVE V1.8.1 (2024). https://www.asam.net/standards/detail/opendrive, https://publications.pages.asam.net/standards/ASAM_OpenDRIVE/ASAM_OpenDRIVE_Specification/latest/specification/03_terms_and_definitions/03_terms_and_definitions.html. Accessed 28 Apr 2025
2. ASAM e.V.: ASAM OpenDRIVE Version 1.7.0. https://www.asam.net/index.php?eID=dumpFile&t=f&f=4422&token=e590561f3c39aa2260e5442e29e93f6693d1cccd (2021). https://www.asam.net/index.php?eID=dumpFile&t=f&f=4422&token=e590561f3c39aa2260e5442e29e93f6693d1cccd. Accessed 27 May 2025
3. ASAM e.V.: ASAM Standards (2024). https://www.asam.net/. Accessed 28 Apr 2025
4. ASAM e.V.: OpenDRIVE V1.8.1 (2024). https://www.asam.net/standards/detail/opendrive. Accessed 28 Apr 2025
5. ASAM e.V.: OpenDRIVE V1.8.1/10 Roads/10.3 Road linkage/Figure 39 (2024). https://publications.pages.asam.net/standards/ASAM_OpenDRIVE/ASAM_OpenDRIVE_Specification/latest/specification/10_roads/10_03_road_linkage.html. Accessed 15 May 2025
6. ASAM e.V.: QC OpenDRIVE. https://github.com/asam-ev/qc-opendrive (2024), gitHub repository
7. Austin, J., et al.: Program synthesis with large language models. arXiv preprint arXiv:2108.07732 (2021)
8. Becker, D., Ru , F., Geller, C., Eckstein, L.: Generation of complex road networks using a simplified logical description for the validation of automated vehicles. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), pp. 1–7 (2020). https://doi.org/10.1109/ITSC45102.2020.9294664
9. Chang, C., Wang, S., Zhang, J., Ge, J., Li, L.: LLMScenario: large language model driven scenario generation. IEEE Trans. Syst. Man Cybern. Syst. (2024). https://doi.org/10.1109/TSMC.2024.3392930
10. Chen, M., et al.: Evaluating large language models trained on code. CoRR abs/2107.03374 (2021)
11. Guo, D., et al.: DeepSeek-R1: incentivizing reasoning capability in LLMs via reinforcement learning (2025). DeepSeek-AI. https://arxiv.org/abs/2501.12948
12. Eisemann, L., Maucher, J.: Automatic odometry-less opendrive generation from sparse point clouds. In: 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC), pp. 681–688. IEEE (2023). https://doi.org/10.1109/itsc57777.2023.10421842

13. Eisemann, L., Maucher, J.: Divide and Conquer: a systematic approach for industrial scale high-definition opendrive generation from sparse point clouds. In: 2024 IEEE Intelligent Vehicles Symposium (IV), pp. 2443–2450. IEEE (2024). https://doi.org/10.1109/IV55156.2024.10588602
14. Grattafiori, A., et al.: The llama 3 herd of models (2024). https://arxiv.org/abs/2407.21783
15. Huang, L., et al.: A survey on hallucination in large language models: principles, taxonomy, challenges, and open questions. ACM Trans. Inf. Syst. **43**(2), 1–55 (2025). https://doi.org/10.1145/3703155
16. Huynh, N., Lin, B.: Large language models for code generation: a comprehensive survey of challenges, techniques, evaluation, and applications (2025). https://arxiv.org/abs/2503.01245
17. Jiang, J., Wang, F., Shen, J., Kim, S., Kim, S.: A survey on large language models for code generation (2024). https://arxiv.org/abs/2406.00515
18. Lai, J., Gan, W., Wu, J., Qi, Z., Yu, P.S.: Large language models in law: a survey (2023). https://arxiv.org/abs/2312.03718
19. Lewis, P., et al.: Retrieval-augmented generation for knowledge-intensive NLP tasks (2021). https://arxiv.org/abs/2005.11401
20. Microsoft Corporation: Introducing Copilot Agent Mode (2025). https://code.visualstudio.com/blogs/2025/02/24/introducing-copilot-agent-mode
21. Pires, R., Junior, R.M., Nogueira, R.: Automatic legal writing evaluation of LLMs (2025). https://arxiv.org/abs/2504.21202
22. Ruan, B.K., Tsui, H.T., Li, Y.H., Shuai, H.H.: Traffic scene generation from natural language description for autonomous vehicles with large language model (2025). https://arxiv.org/abs/2409.09575
23. Scholtes, M., et al.: 6-layer model for a structured description and categorization of urban traffic and environment (2021). https://arxiv.org/abs/2012.06319
24. Schwab, B., Kolbe, T.: Validation of parametric opendrive road space models. Ann. Photogrammetry, Remote Sens. Spatial Inf. Sci. X-4-W2-2022 (2022). https://doi.org/10.5194/isprs-annals-X-4-W2-2022-257-2022
25. Team, M.A.: Mistral large. https://mistral.ai/news/mistral-large-2407 (2024). https://mistral.ai/news/mistral-large-2407. Accessed 26 May 2025
26. Wachenfeld, W., Winner, H.: The Release of Autonomous Vehicles, pp. 425–449. Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-48847-8_21
27. Wei, J., et al.: Chain-of-thought prompting elicits reasoning in large language models (2023). https://arxiv.org/abs/2201.11903
28. Williams, S., Huckle, J.: Easy problems that LLMs get wrong (2024). https://arxiv.org/abs/2405.19616
29. Xiao, Y., Sun, Y., Lin, Y.: ML-SceGen: a multi-level scenario generation framework (2025). https://arxiv.org/abs/2501.10782
30. Yang, A., et al.: Qwen3 technical report (2025). https://arxiv.org/abs/2505.09388
31. Yang, J., et al.: Harnessing the power of LLMs in practice: a survey on chatgpt and beyond (2023). https://arxiv.org/abs/2304.13712
32. Zhang, J., Xu, C., Li, B.: ChatScene: knowledge-enabled safety-critical scenario generation for autonomous vehicles (2024). https://arxiv.org/abs/2405.14062