2026 IEEE/SICE International Symposium on System Integration (SII)
January 11-14, 2026. Cancún, México

A LLM-Assisted Compiler for Generating Standard-Compliant Driving Scenarios from Natural Language

Majid Jegarian[1*], Amir K. Esfahani[1], Katharina Bause[1], and Tobias Düser[1]

Abstract— **This paper introduces a Scenario Compiler for converting textual concrete scenarios into executable XML files compliant with simulation standards. It serves as the final module in a structured framework for automated driving scenario generation, transforming natural language descriptions into simulation-ready XML files. The proposed approach combines schema-guided parsing with contextual inference using a fine-tuned large language model (LLM). In the first phase, the parser generates a complete and schema-compliant XML structure and fills in all directly extractable values, and inserting placeholders for missing or ambiguous information. The challenge addressed here is the frequent absence of explicit values for certain scenario parameters in the input text, which makes it difficult to generate fully specified XML solely through rule-based methods. In the second phase, the fine-tuned LLM infers and fills in these missing values by analyzing the broader scenario context, ensuring that the final output is both complete and plausible. An evaluation against non-specialized LLMs shows that the Scenario Compiler produces significantly more correctly instantiated XML elements while avoiding invalid tags or schema violations. By combining rule-based schema compliance with LLM-based reasoning, the approach automates the scenario generation process and reduces manual effort in simulation-based validation workflows.**

I. INTRODUCTION

The safety validation of automated vehicles (AVs) relies on evaluating their performance in diverse, realistic, and safety-critical driving situations. A scenario, in this context, is defined as a structured representation of a driving situation that includes road infrastructure, static and dynamic actors, temporal progression, and events relevant to the ego vehicle. Scenario-based testing has become a central methodology in AV validation, offering a systematic way to assess behavior under controlled yet representative conditions. [1]

Scenarios are typically categorized by abstraction levels. A functional scenario refers to a high-level natural language description of a situation, such as "a vehicle cuts in during highway driving." An abstract scenario specifies the entities involved and their expected behaviors without assigning concrete parameters. A concrete scenario provides full parameterization, including positions, timing, speeds, and conditions, allowing direct execution in simulation environments [2].

While scenario-based testing is critical, the creation of executable and standards-compliant scenarios remains a bottleneck. Manual scenario engineering is labor-intensive, requires domain expertise, and frequently results in structural inconsistencies or omissions. Producing compliant outputs that adhere to formal standards introduces additional complexity, including strict schema adherence, hierarchical structure, and semantic dependencies. These challenges hinder scalability and slow down testing cycles in simulation pipelines. [1]

Recent advancements in large language models (LLMs) have enabled the automatic generation of structured content from natural language prompts [3], a capability particularly suitable for use in scenario generation. However, generating simulation-executable scenarios introduces additional constraints that general-purpose models are not equipped to handle. Formats like ASAM OpenSCENARIO and OpenDRIVE [4] are rarely represented in pretrained corpora, leading to outputs that often violate schema constraints, omit required elements, or exhibit semantic inconsistencies. Even with task-specific training, limitations in data availability and standard complexity continue to affect generation accuracy.

To address these challenges, an automated framework was introduced in our prior work [5] to progressively refine user input into simulation-ready artifacts; additional details are provided in Section III. The present work focuses on the final transformation stage of this framework, which ensures that the resulting scenario code adheres to the structural and semantic constraints of standard simulation formats. This transformation involves schema-guided parsing, targeted LLM interaction, and validation processes that collectively address ambiguity, incompleteness, and conformance requirements in LLM-generated content. The resulting outputs preserve both structural integrity and scenario realism, enabling scalable generation of executable test cases for AV simulation environments.

II. RELATED WORK

Scenario generation lies at the core of scenario-based testing methodologies for autonomous vehicle validation [6]. Various paradigms have been explored to create realistic and safety-critical driving scenarios, each addressing different levels of abstraction and operational complexity. Methods span knowledge-based design [7], data-driven sampling [8], search-based exploration [9], and hybrid techniques such as gamification [10]. While these techniques offer scenario diversity, they often struggle with scalability, computational cost, or lack of alignment with evolving simulation standards.

To address these issues, recent works leverage LLMs to translate functional descriptions into structured scenarios. Frameworks like ChatScene [3], LLMScenario [11], and InternDrive [12] demonstrate how LLMs can capture semantic

[1]Majid Jegarian, Amir K. Esfahani, Katharina Bause, and Tobias Düser are with IPEK – Institute of Product Engineering at Karlsruhe Institute of Technology (KIT), Kaiserstr. 10, 76131 Karlsruhe, Germany
*Corresponding author, E-mail address: majid.jegarian@kit.edu

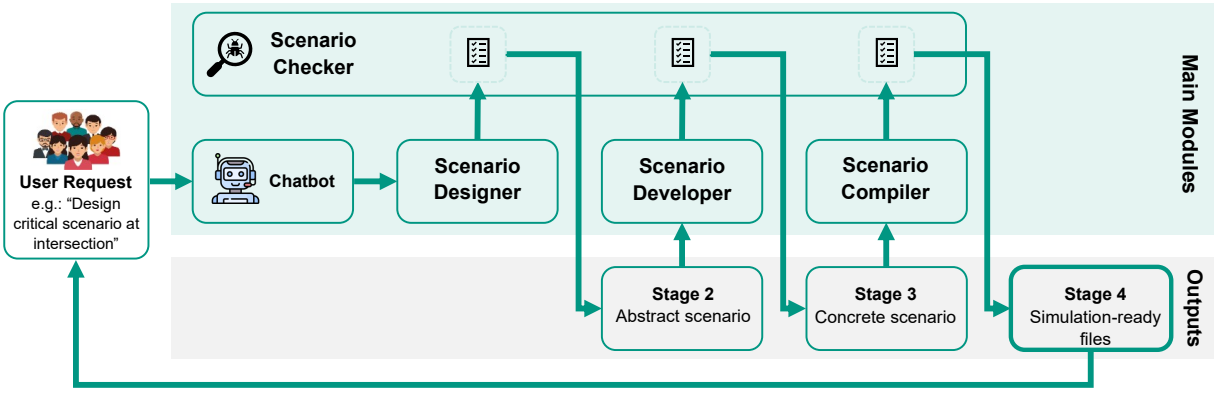

Fig. 1. LLM-based framework for scenario generation [5]

intent and automate scenario creation. However, standalone LLMs frequently produce structurally invalid outputs that violate schema rules or miss required parameters. Fine-tuning improves structural conformity but remains limited by the scarcity of annotated scenario datasets.

A. Code Generation and Schema Adherence

LLM-driven code generation has shown success in programming tasks, supported by benchmarks like CodeXGLUE [13] and CoNaLa [14]. However, generating AV scenarios in XML-based standards poses unique challenges—complex hierarchies, strict field dependencies, and schema (XSD)-level validation. Many existing tools lack deep integration of domain ontologies or XML constraints, sometimes resulting in outputs that do not fully meet structural or semantic simulation requirements [15]. Moreover, simulators require precise timing, triggers, and action semantics—difficult to learn from general code data. These gaps emphasize the need for hybrid approaches that combine LLM reasoning with strict structural enforcement for simulation-ready outputs.

B. Hybrid Toolchain

To compensate for LLM limitations, hybrid architectures integrate rule-based validation with generative models. Techniques such as self-refining generation loops, schema-aware repair modules, and structured output verification have improved the reliability of structured code generation [16]. For example, recent frameworks leverage reinforcement fine-tuning [17] or in-context schema prompting [18] to guide iterative correction and improve semantic alignment with formal standards. These methods enhance XML compliance and logical consistency but are often domain-specific or lack modular integration into scenario generation workflows tailored for AV simulation.

C. Summary and Positioning

Existing approaches have made progress in scenario modeling, structured content generation, and LLM-based reasoning. Nonetheless, many of these methods rely on rigid templates, lack formal structural enforcement, or fail to produce simulation-executable outputs. A critical gap persists in systematically transforming natural language scenario descriptions into validated artifacts that satisfy both schema constraints and contextual plausibility. The proposed work addresses this challenge through a Scenario Compiler that integrates locally fine-tuned LLM with schema-constrained symbolic parsing and iterative validation. This hybrid architecture supports scalable, standards-compliant scenario generation tailored for automated vehicle simulation environments.

III. Overview of Scenario Generation Pipeline

Automated scenario generation for autonomous vehicle testing requires a structured process capable of transforming natural language descriptions into simulation-executable code. The framework addressed in this work follows a modular pipeline that decomposes this transformation into three progressive modules. Each modules corresponds to a level of scenario abstraction, ensuring both semantic clarity and increasing structural precision.

The first module processes functional descriptions to identify relevant entities, environments, goals, and interactions. The result is an abstract scenario that defines structural components such as vehicle types, environmental context, and intended maneuvers without specifying exact values. The second module builds on this abstraction to construct a parameter-rich scenario representation. At this level, attributes such as initial positions, velocities, timing conditions, and trigger logic are introduced to produce a complete but symbolic representation of a scenario. The third module performs the final transformation, converting structured scenario specifications into executable simulation code conforming to standards such as ASAM OpenSCENARIO and OpenDRIVE.

Each module within the framework contributes to a hierarchical refinement process supported by LLMs. The first two modules emphasize semantic interpretation and logical decomposition. The third module, detailed in the subsequent sections, enforces syntactic correctness, schema alignment, and executable fidelity by integrating symbolic parsing with structured LLM prompting and post-processing.

Fig. 1 illustrates the high-level pipeline architecture, including the three modules and their outputs. This diagram,

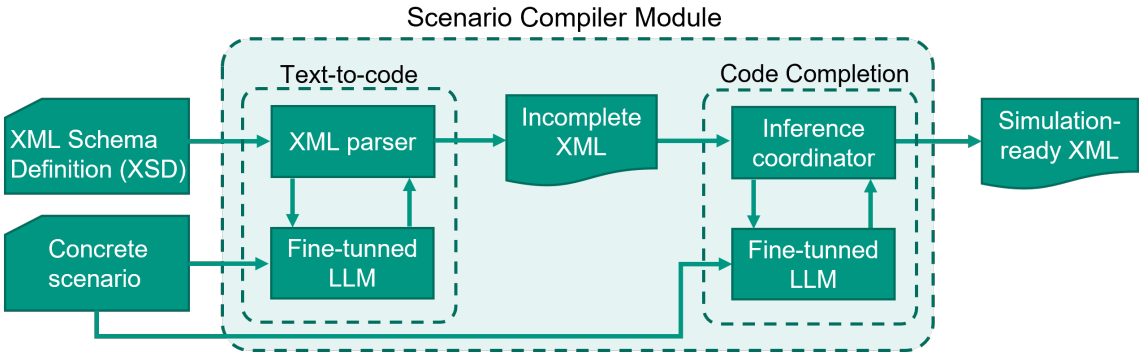

Fig. 2. Overview of the Scenario Compiler architecture. The system integrates schema-driven parsing, value inference via a fine-tuned LLM, and the Inference Coordinator, which manages context construction, prompt communication, and semantic validation to transform incomplete scenario representations into fully executable XML code.

adapted from earlier work on LLM-based scenario generation [5], remains relevant in the current context. It has been included here to provide continuity for readers and to offer a visual overview of the full system. The remaining sections of this paper focus specifically on the third module, the Scenario Compiler.

IV. Scenario Compiler workflow

The Scenario Compiler transforms semi-structured, human-readable scenario descriptions into executable XML files suitable for simulation environments. The compilation process must ensure two core requirements: structural correctness, defined by full compliance with the selected XML schema, and semantic fidelity, which ensures logical coherence, realistic parameterization, and simulator executability.

Meeting both structural correctness and semantic fidelity is challenging for symbolic and learning based approaches alike. Symbolic, rule-based, compilers reliably enforce XML schema compliance but cannot reason about dynamic, context-dependent behavior, especially when dealing with loosely structured input such as partially formatted or LLM-generated scenario descriptions. For example, in a cut-in scenario, a rule-based compiler may produce vehicle trajectories that conform to the schema but miss critical timing or positioning values. This can happen when information is missing, vaguely described, or embedded in parts of the input that the rule-based system does not anticipate, resulting in an unrealistic or unexecutable maneuver. In contrast, learning based approaches are better at interpreting ambiguous or incomplete input but often generate outputs that violate structural rules or include semantically implausible elements due to limited exposure to simulation standards and a lack of deterministic control.

To address these challenges, the Scenario Compiler adopts a two-phase architecture that separates structure generation from semantic enrichment. A schema-guided parser ensures compliance with the structure defined in the selected simulation standard, while a fine-tuned LLM supports interpretation of the scenario, resolution of ambiguity, and inference of required parameter values such as speeds, positions, or trigger conditions based on the context.As illustrated in Fig. 2, the compiler operates on two inputs:

- An XML Schema Definition (XSD) file defining the structure of the selected standard
- A concrete scenario in natural language, generated in an earlier module of the pipeline

The output is an XML file conforming to the target specification. Implementation in OpenSCENARIO V1.1.0 is used to demonstrate structural compliance and simulation compatibility within real-world test environments. The compilation process consists of two stages. The first stage, Text-to-Code, generates a structurally complete XML file by instantiating all schema-required elements and inserting placeholders where values are missing or ambiguous. The second stage, Code Completion, replaces these placeholders using responses from a fine-tuned LLM and validates the output to ensure compliance with the schema.

A. Text-to-Code

The first phase of the compilation pipeline, Text-to-Code, focuses on producing a structurally valid XML representation of the scenario based on the provided schema. The objective is to generate the full hierarchical structure defined by the target simulation standard, including all mandatory elements and attributes, while deliberately leaving semantically complex or ambiguous value fields unresolved for refinement in the subsequent phase.

The process begins by parsing the XSD file associated with the selected simulation format. A schema-aware XML parser uses these definitions to construct the skeleton of the scenario, including required tags, nested hierarchies, and attribute placements. Decisions regarding the inclusion of optional elements or the population of specific attribute values, however, depend on contextual information provided in the concrete scenario, which is generated by a LLM in the previous module and expressed in a partially structured textual format. Processing such text presents challenges that deterministic methods alone cannot resolve. To address this challenge, a fine-tuned LLM is incorporated to assist the parser with contextual interpretation and schema-aligned decision-making.

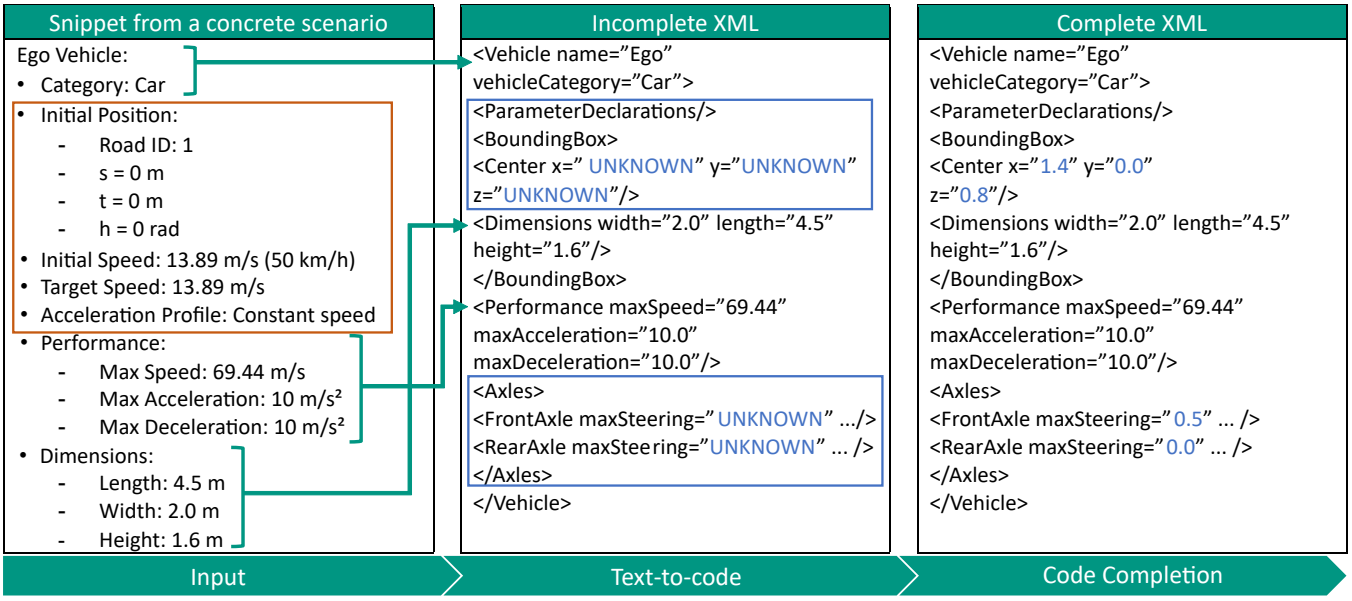

Fig. 3. Illustration of the Scenario Compiler process. Part **(a)** shows a snippet from a concrete scenario. Tags successfully extracted into XML are indicated by green arrows. Tags highlighted in orange blocks represent hallucinated elements introduced by the LLM that are correctly excluded from the output. Part **(b)** displays the incomplete XML generated by the Text-to-Code phase, where all mandatory elements are included and structurally valid. Components like *<Center>* and *<Axles>*, which are not specified in the input, are inserted with placeholder values labeled as `UNKNOWN`, shown in blue. Part **(c)** illustrates the final XML after Code Completion, in which the `UNKNOWN` fields have been replaced with validated values inferred by the model.

When the parser encounters a decision point, such as whether to include an optional tag or what to assign an attribute value, it issues a structured prompt to the LLM. This prompt references the relevant schema element and includes contextual fragments of the scenario. The model, trained specifically on the target standard and scenario semantics, responds with a recommended action or value. This query-response mechanism ensures that only schema-compliant elements are generated. Any extraneous or hallucinated constructs within the concrete scenario are inherently excluded, as the parser restricts LLM interaction to schema-defined components only. As a result, unsupported structures or misaligned information are filtered out during the generation process. In cases where no relevant information is present in the scenario for a required field, the LLM is instructed to return the placeholder `UNKNOWN`. This ensures that the output XML remains complete in structure, even if some values are missing, and avoids omitting any required tags. The output of this phase is a fully structured XML file that conforms to the schema of the selected simulation format. Although complete in structure, the file may contain unresolved values in semantically sensitive fields. This intermediate result is referred to as the *Incomplete XML* in Fig. 2. These fields are marked for revision during the next compilation phase, where contextual reasoning and validation will be applied to generate executable values. Fig. 3, part (a) presents a sample concrete scenario input, and part (b) shows the corresponding incomplete XML generated by the Text-to-Code phase, showcasing the possible cases.

B. Code Completion

The second phase, Code completion, addresses the semantic enrichment of the partially structured XML generated during the first stage. While the overall structure conforms to the selected schema, many value fields remain either blank or marked with the placeholder `UNKNOWN` due to missing or unclear information in the input scenario. In this phase, each unresolved field is processed through a structured interaction with a fine-tuned LLM. For each placeholder, a prompt is constructed that includes two contextual components:

- The location and type of the XML element within the hierarchy.
- The corresponding scenario description in natural language.

The LLM, trained specifically on the selected simulation standard, processes the prompt and returns a contextually appropriate value. This specialization improves precision by ensuring awareness of domain-specific constraints, semantic definitions, and allowable parameters. Unlike general-purpose models, the fine-tuned model can resolve ambiguous references, interpret indirect cues, and infer plausible defaults when necessary.

Each returned value is then subjected to a validation process before being integrated into the XML file. This validation ensures syntactic and semantic correctness through:

- Type checks, verifying that the value format matches the schema definition (e.g., integer, float, string)
- Range constraints, ensuring that values fall within physically and logically valid bounds
- Semantic checks, confirming consistency across related fields, such as matching identifiers or valid speed pa-

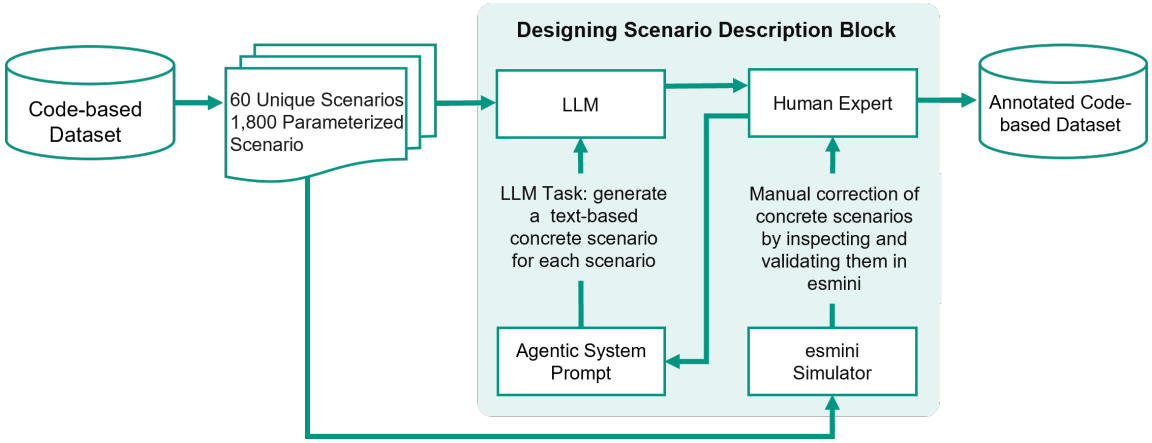

Fig. 4. Dataset construction. Annotated scenario descriptions are derived from existing code using templated prompting, followed by manual correction and simulation-based validation.

rameters

If the LLM cannot extract a valid value from the scenario context, it is instructed to infer a reasonable estimate using previously resolved fields and structural dependencies. This fallback mechanism improves robustness in cases where the input description is under-specified.

The output of this phase is a semantically enriched XML file that satisfies both structural and execution-level requirements. By isolating the structural logic from the semantic reasoning process, the Scenario Compiler maintains schema compliance while leveraging the interpretive capabilities of the LLM. These two phases collectively ensure that the generated scenario is both formally valid and practically executable. In Fig. 3, part (c) illustrates the final XML output after the Code Completion phase, in which all previously unresolved fields have been populated through LLM-based inference.

C. Fine-tuning Strategy

To enable accurate generation of OpenSCENARIO and OpenDRIVE files from text-based concrete scenario descriptions, a dedicated LLM was fine-tuned. The base model selected for this task is Meta's LLaMA 3.1 with 7 billion parameters [19]. This model provides a strong foundation for structured text generation and efficient parameter adaptation. However, out-of-the-box performance on simulation-specific XML generation is limited due to the absence of aligned multimodal scenario data in the original training set.

To adapt the model to this task, a Parameter-Efficient Fine-Tuning (PEFT) [20] method was employed. Specifically, Low-Rank Adaptation (LoRA) [21] was used to reduce computational and memory overhead during fine-tuning. LoRA inserts small, trainable low-rank matrices into each transformer layer and updates only these additional parameters, leaving the core model weights untouched. This method allows for efficient adaptation to domain-specific tasks while minimizing resource consumption and preserving generalization performance.

The fine-tuning process was conducted on a single-node system equipped with four A100-SXM4-40GB GPUs. Hyperparameters were selected based on existing best practices for LoRA-based adaptation, with minor adjustments to batch size and learning rate to fit the task-specific dataset. The fine-tuned model serves as the core reasoning engine within the Scenario Compiler's code completion phase, responsible for value inference, contextual resolution, and simulation-aware tag population.

Due to the lack of any existing dataset that aligns abstract or concrete scenarios with their corresponding .xosc and .xodr representations, a domain-specific dataset was constructed to support fine-tuning. The details of this process are outlined in the following subsection.

Dataset Construction. A publicly available benchmark dataset that aligns natural language scenario descriptions with both OpenSCENARIO and OpenDRIVE files does not currently exist. To address this gap, a structured dataset was created using simulation-ready scenarios from the Safety Pool database [22]. While the Safety Pool provides hundreds of downloadable .xosc and .xodr files, many of these differ only in numerical parameter values and therefore contribute limited diversity for LLMs training. A curated subset of 60 structurally unique scenarios was selected, supplemented by 1,800 parameterized variations. For each selected scenario, the .xosc and .xodr files were fed into a base LLM, accompanied by prompt instructions and a template specifying the required elements of a valid concrete scenario. This template was derived from the internal specifications of the Scenario Developer module and ensured that all mandatory components were covered. The generated outputs frequently suffered from inconsistencies. These included hallucinated entities not present in the original code, alterations to road type, and formatting mismatches between the scenario description and the code representation. To address these issues, every generated description was manually verified using the Esmini simulator [23]. Any deviations between the textual description and simulation behavior were identified and corrected, ensuring that the dataset remained structurally

aligned and semantically coherent. The full pipeline for scenario description generation and correction is illustrated in Fig. 4.

V. Results

The dataset described in Section IV-C was used for evaluation. Several non-specialized LLMs were prompted to generate XML scenario files from concrete scenario descriptions, with respect to the provided XSD schema. The results produced by these models and the proposed approach were then compared against the ground truth. The evaluation focused on three key metrics, calculated on a per-scenario basis:

- The average number of generated XML tags
- The average number of assigned value fields
- The average number of out-of-context tags (i.e., irrelevant or structurally invalid tags not defined in the target XSD schema)

TABLE I

Quantitative Evaluation of XML Generation of XML files generated by the proposed compiler and non-specialized LLMs, measured by tag count, value assignment, and percentage of structurally invalid tags relative to the XSD schema.

XML Generation Method	Tags	Values	Out-of-context (%)
Ground Truth	298	240	0
Proposed Approach	811	461	0
Llama3.1-70B	90	74	42
DeepSeek-R1-70B	93	20	63
Llama3.1-8B	67	41	73
Ministral-8B-Instruct	54	27	74

The results, summarized in Table I, compare the outputs of the proposed method with those of various LLMs. All non-specialized models generated substantially fewer XML tags than the ground truth. This difference may suggest simplification by omitting optional details, but it more likely reflects the omission of required components, leading to structurally incomplete XML representations. In addition, the out-of-context tag rate remains high across all non-specialized models, indicating a frequent violation of schema structure due to hallucinated or misplaced elements. In contrast, the proposed method generates a larger number of tags than the ground truth. This may result from richer elaboration or possible over-generation due to misinterpretation of the scenario input. Crucially, the approach maintains a zero percent out-of-context tag rate, highlighting the effectiveness of schema-guided enforcement in the Text-to-Code phase.

Table II presents detailed statistics on value retrieval and assignment across the two phases of the proposed approach. The incomplete XML output from the Text-to-Code phase contains numerous `UNKNOWN` placeholders, reflecting missing or ambiguous information in the input concrete scenario file. However, the vast majority of these placeholders are successfully resolved by the subsequent Code-Completion phase, indicating a considerable progression toward generating a simulation-ready XML file.

TABLE II

Value Assignment Progression Across Processing Phases

Metric	Average per Scenario
Total XML tags	811
Required value fields	461
Values filled after Text-to-Code (%)	43
Values filled after Completion (%)	96

Table II shows that the Code Completion phase significantly improves value assignment, increasing the proportion of filled fields from 43% to 96%. This highlights its key role in resolving missing data. However, not all generated XML files could be simulated or validated as the intended driving scenario. This is mainly due to three reasons:

- Poor quality of the concrete scenario, especially when critical aspects of the XML are missing in the input.
- Over-generation of XSD element instances by the LLM in the Text-to-Code phase, which leads to a large number of `UNKNOWN`s and complicates value assignment during Code-Completion.
- Inability of the LLM to assign precise values to `UNKNOWN`s in Code-Completion, resulting in invalid final XMLs.

VI. Conclusion

This work presented a Scenario Compiler that transforms natural language scenario descriptions into executable XML code conforming to structured simulation standards. The proposed architecture integrates a schema-guided parser with a fine-tuned LLM to ensure structural compliance and context-aware value assignment. Through decoupling structural generation from semantic enrichment, the compiler supports accurate and scalable scenario synthesis for automated vehicle validation. Quantitative evaluation demonstrated that the two-stage design effectively increases the number of correctly generated tags and eliminates out-of-context XML tags when compared to unguided LLM baselines. The Text-to-Code stage generates a fully structured XML file using a schema-driven parser, while the Code Completion stage resolves missing values through LLM-guided contextual inference. This architecture significantly reduces manual overhead and advances the automation of simulation-ready scenario generation.

Future work will focus on enhancing two core aspects of the Scenario Compiler: the reliability of the input concrete scenario and the robustness of model fine-tuning. Improving input quality is essential, as incomplete or ambiguous inputs often result in XML files with excessive placeholders or invalid structures. Planned improvements include the introduction of stricter input templates and structured authoring tools to reduce ambiguity. Ongoing investigations are also defining evaluation criteria such as required XSD keyword coverage and XSD-based hallucination rates. Expanding and diversifying the fine-tuning dataset also remains a priority. Evaluation criteria for future versions should include schema compliance, tag completeness, and simulation pass rates. Fine-tuning progress can be monitored using standard loss

metrics on training and validation sets. Further directions include the exploration of advanced prompt engineering techniques and feedback-driven refinement strategies that leverage simulation outcomes to guide iterative model correction. Adaptability to evolving simulation schemas and domain extensions will also be essential for long-term applicability.

ACKNOWLEDGMENT

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC).

REFERENCES

[1] S. Riedmaier et al. "Survey on Scenario-Based Safety Assessment of Automated Vehicles". In: *IEEE Trans. Intell. Transp. Syst.* 8 (2020), pp. 87456–87477.

[2] H. Winner et al. *PEGASUS—First Steps for the Safe Introduction of Automated Driving*. 2019.

[3] J. Zhang et al. "ChatScene: Knowledge-Enabled Safety-Critical Scenario Generation for Autonomous Vehicles". In: *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*. 2024, pp. 15459–15469.

[4] ASAM e.V. *ASAM OpenSCENARIO 1.0.0 and ASAM OpenDRIVE 1.6.0 Standards*. Accessed July 2025. 2020.

[5] M. Jegarian et al. "Integrating RAG and Reasoning for the Realization of a Concrete Driving Scenario within an LLM-Based Framework". In: *Proc. IEEE Int. Conf. Autom. Veh. Valid. (IAVVC)*. Accepted. 2025.

[6] S. Ulbrich et al. "Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving". In: *Proc. IEEE Int. Conf. Intell. Transp. Syst. (ITSC)*. 2015.

[7] G. Bagschik et al. "Ontology Based Scene Creation for the Development of Automated Vehicles". In: *Proc. IEEE Intell. Veh. Symp. (IV)*. 2018, pp. 1813–1820.

[8] S. Kuutti et al. "A Survey of Deep Learning Applications to Autonomous Vehicle Control". In: *IEEE Trans. Intell. Transp. Syst.* 22.2 (2021), pp. 712–733.

[9] M. Koren et al. "Adaptive Stress Testing for Autonomous Vehicles". In: *Proc. IEEE Intell. Veh. Symp. (IV)*. 2018, pp. 1–7.

[10] M. Jegarian et al. "Conflict Detection in Automated Vehicle Testing Through Gamification". In: *Proc. IEEE Int. Conf. Autom. Veh. Valid. (IAVVC)*. 2024, pp. 1–7.

[11] C. Chang et al. "LLMScenario: Large Language Model Driven Scenario Generation". In: *IEEE Trans. Syst. Man Cybern. Syst.* 54.11 (2024), pp. 6581–6594.

[12] Y. Zhang and Y. Nie. "InternDrive: A Multimodal Large Language Model for Autonomous Driving Scenario Understanding". In: *Proc. Int. Conf. Artif. Intell. Autom. High Perform. Comput. (AIAHPC)*. 2024, pp. 294–305.

[13] S. Lu et al. *CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation*. 2021.

[14] CoNaLa Team. *CoNaLa: The Code/Natural Language Challenge*. 2025.

[15] J. Li et al. "Generative AI for Self-Adaptive Systems: State of the Art and Research Roadmap". In: *ACM Trans. Auton. Adapt. Syst.* 19.3 (2024).

[16] X. Zhu et al. *Large Language Model Enhanced Text-to-SQL Generation: A Survey*. 2024.

[17] X. Pan et al. *Trinity-RFT: A General-Purpose and Unified Framework for Reinforcement Fine-Tuning of Large Language Models*. 2025.

[18] N. Gupta et al. "Schema and Natural Language Aware In-Context Learning for Improved GraphQL Query Generation". In: *Proc. Conf. NAACL-HLT, Industry Track*. 2025, pp. 1009–1015.

[19] Meta AI. *LLaMA 3: Open Foundation and Instruction-Tuned Models*. Accessed July 2025. 2024.

[20] S. Liu et al. "Parameter-Efficient Fine-Tuning Design Spaces". In: *arXiv preprint arXiv:2202.08906* (2022).

[21] E. J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *arXiv preprint arXiv:2106.09685* (2021).

[22] Safety Pool. *Safety Pool™ Scenario Database*. Accessed July 2025. 2025.

[23] Esmini Developers. *Esmini – OpenSCENARIO Viewer and Simulator*. Accessed July 2025. 2025.