Automated Generation of SysML Activity Diagrams from Industrial Requirements Using LLMs

Bastian Franze*, Dominik Fuchß†, Friedrich Wattenberg*, Till Fuchs*, Jan Keim†, Tobias Hey†
*Dr. Ing. h.c. F. Porsche AG, Weissach, Germany
{bastian.franze2, friedrich.wattenberg1, till.fuchs2}@porsche.de
†Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{dominik.fuchss, keim, hey}@kit.edu

***Abstract*—The increasing complexity of cyber-physical systems, such as modern automotive E/E architectures, demands efficient, consistent, and scalable modeling processes. Traditional manual modeling of SysML diagrams from natural-language requirements is time-consuming and error-prone.**

We target to assist E/E architects in the process by utilizing large language models (LLMs). Therefore, we designed a three-step approach for automated generation of SysML activity diagrams. First, the approach identifies diagram element candidates in the requirements and matches them to potential existing elements. Second, missing elements are generated, and finally, the elements are linked.

Evaluated on real-world system-level requirements documents, comprising more than 720 requirements and 72 use cases, our approach generates system-level diagrams comparable to those created by experts and provides valuable architectural recommendations. A survey with nine industry architects shows that at the system-level LLM-generated diagrams were even rated higher than the diagrams modeled by experts. However, quality degrades at the subsystem-level, where human refinement remains essential for correctness and completeness.

***Index Terms*—SysML, activity diagrams, LLM, systems engineering, requirements engineering, E/E architecture, automated modeling, natural language processing**

I. Introduction

In the development of cyber-physical systems (CPSs), systems engineering is often the preferred development process. Due to the complexity of modern CPS, many different stakeholders are involved in the early development stages. This complicates maintaining consistency across processes, requirements, and architectures. In systems engineering, functional requirements of the CPS are often manually translated by system and function architects into stakeholder-specific use cases and subsequently mapped to logical, functional, and physical architectures [1]. One common intermediate representation is the translation of use cases into System Modeling Language (SysML) activity diagrams. In doing so, the system's behavior is specified in further detail.

Through this process, multiple viewpoints on the CPS yield numerous diagrams describing system behavior using an even larger number of system elements. In particular, concurrency and branching are difficult to describe and communicate in natural language alone. As it can be difficult for experts to ensure consistency across these many elements and with respect to existing systems, tool support may improve both the quality and efficiency of this process. To this end, approaches for generating Unified Modeling Language (UML) diagrams from requirements have been researched for years [2]–[7]. Recently, LLMs have enabled new possibilities for automating or assisting the process of diagram generation [8]–[19].

In this paper, we propose and evaluate an approach for automatically generating SysML activity diagrams from use cases and their requirements using LLMs in an industrial setting from the automotive domain. To evaluate our approach, we compare real, manually-created diagrams with those generated by an LLM. Due to the ambiguity of natural language and the degrees of freedom in the solution space, we also conducted a survey of experts to rate the outcomes for three example use cases. We aim to answer the following questions:

RQ1: How consistent are SysML activity diagrams generated by LLMs in a real-world industrial setting?
RQ2: Which factors have the strongest impact on the performance of LLMs in SysML activity diagram generation?
RQ3: What impact does the use of an automated activity diagram generation tool have on architecture quality in an industrial setting?

II. Industrial Background

This work was conducted in collaboration with an industry partner and is based on their data and reflects their processes. This section examines differences from common research bases and introduces industry-specific terms used in this paper.

A. E/E architecture

The requirements and use cases used in this paper focus on E/E architectures of vehicles. E/E architectures describe the structure and connections of electrical, electronic, mechatronic, mechanical, and software systems of a CPS. We examine the high-level functional architecture of vehicles. The functional architecture is modeled with cross-referenced SysML activity diagrams. The CPS is described using high-level features across multiple use cases and their associated requirements. System functions are described in system-level requirements documents (SLRDs) using natural language. In our industrial context, the requirements describing vehicle functions are structured in the SLRD using use cases.

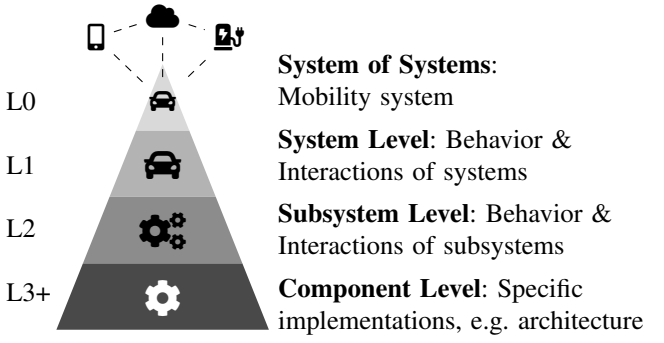

Fig. 1. Hierarchical system structure of a system of systems used by our industry partner. The pyramid illustrates the decomposition from the system-of-systems level to individual constituent systems and further to their internal subsystems.

B. SysML

SysML [20] is a general-purpose modeling language for systems engineering that is derived from UML. While UML primarily targets software-intensive systems, SysML extends UML by introducing additional diagram types and modeling constructs to describe requirements, physical components, parametric relationships, and interdisciplinary system aspects. Although activity diagrams in SysML are largely based on their UML counterparts, their interpretation in practice differs. For example, swimlanes in activity diagrams are often directly mapped to logical systems or subsystems. SysML activity diagrams also use ports more often than their UML counterparts.

C. Development Process

In the development of a new architecture, the stakeholders define high-level requirements for their use cases. Those requirements are then transformed into high-level systems and functions, which are used to derive lower-level requirements through a system-of-systems approach. An overview of the different abstraction levels is shown in Figure 1. In this paper, we focus on system-level (SL) and subsystem-level (SSL) diagrams [21], [22]. The emphasis on activity diagrams is motivated by industrial practice. In model-based systems engineering processes, activity diagrams are commonly used to describe the behavior of systems and their subsystems. Interconnected activity diagrams subsequently serve as a basis for behavioral analysis, particularly in functional safety assessments and the derivation of test cases.

III. Related Work

Over the past decades, research on automated model generation has evolved through several distinct phases. Ahmed et al. [2] provide a broad perspective on this development by examining 70 publications on automated architecture derivation. Their analysis shows that most early work, published between 1994 and 2021, focused on generating UML class diagrams using classical natural language processing (NLP) techniques. Activity diagrams, by contrast, were addressed only twice in the surveyed literature, and exclusively through traditional NLP pipelines.

With the emergence of modern LLMs in late 2023, the research landscape shifted noticeably. First, studies began to explore whether these models could support or even automate software architecture generation. While the conceptual feasibility was quickly demonstrated, follow-up analyses revealed persistent challenges: several works reported semantic inconsistencies and omissions, particularly when requirements were ambiguous or incomplete [8], [10], [11], [14]. Nevertheless, the potential of LLMs to reduce pipeline complexity sparked increased interest.

A. Diagram Generation using NLP

Before LLMs became widely available, classical NLP approaches formed the foundation of automated diagram generation. Most contributions targeted class diagrams and domain models, while later work extended to behavioral models such as sequence and activity diagrams.

1) Structural Diagram Generation using NLP: Ibrahim and Ahmad [3] were among the early researchers to propose rule-based extraction of class diagrams from natural language requirements. Their method relies on eight rules for identifying classes, two for attributes, and six for relationships, using sentence structure and parts of speech to derive UML elements. Although demonstrated only on a simple example, the work illustrated how linguistic patterns can be systematically mapped to model elements.

Meng and Ban [4] advanced this direction with a pipeline integrating preprocessing, sentence classification, syntactic analysis, and diagram generation. Their method employs nine rules for class identification and six rules for relationship extraction. When evaluated on 600 high-quality diagrams from several domains, the approach achieved classification accuracies above 88 %, demonstrating that classical NLP can yield robust results when applied to well-structured requirement texts.

2) Behavioral Diagram Generation Using NLP: Abdelnabi et al. [23] proposed a structured approach for deriving sequence diagrams from natural language. Their methodology begins with a preparation phase that simplifies requirements using sixteen grammatical rules, followed by fourteen NLP rules to extract sequence diagram elements. Four verification rules help reduce false positives by checking the coherence of identified interactions. Although detailed evaluation data are not fully disclosed, the authors report that their approach generates a broader range of UML diagram types than previous methods and is particularly effective for sequence diagrams.

Compared to class and sequence diagrams, activity diagram generation has been studied less extensively. Yue et al. [5] introduced an approach based on structured use case descriptions that limits natural language variability through templates and predefined keywords. Using grammatical relations and sentence structures, their method derives activity diagrams that, according to their evaluation on five use cases, outperform commercial tools available at the time.

Maatuk et al. [6] extended this line of work by defining 33 structured rules for identifying actors, use cases, activities, and decision nodes. Their methodology uses tokenization, part-of-speech tagging (POS tagging) information, and grammatical dependencies to generate both use case and activity diagrams. Validation was conducted on a single case study, but the work demonstrates that even complex behavioral models can be derived through classical NLP.

B. SysML Diagram Generation Using NLP

Zhong et al. [7] explored the generation of SysML diagrams, focusing on block definition diagrams. Building on established ideas from UML class diagram generation, they employ POS tagging information, parsing, heuristic frequency analysis, and grammatical rules to extract entities and relationships. Their evaluation reveals high accuracy in identifying diagram elements, while also highlighting a recurring issue in rule-based methods: incomplete or low-quality input requirements directly degrade the quality of the generated diagrams. Their work illustrates that UML extraction techniques can be adapted to SysML with relatively minor adjustments.

C. UML Diagram Generation Using LLMs

With the advent of large language models, research attention shifted from manually created linguistic rules to more flexible, prompt-based approaches. Cámara et al. [8] conducted one of the first systematic evaluations of ChatGPT for modeling tasks. They found that while syntactic correctness is generally high, semantic accuracy remains strongly dependent on notation, domain, and prompting strategy. Task decomposition improved results, but output variability remained significant.

Arulmohan et al. [9] compared GPT-3.5 to a classical NLP system [24] for extracting domain models. Their findings show that the rule-based approach achieved an F_1-score of 0.85, clearly outperforming GPT-3.5, which reached 0.6. De Bari et al. [10] similarly observed that GPT-4-generated class diagrams suffer from semantic and textual inaccuracies, even though syntactic correctness is comparable to human-created diagrams. Ferrari et al. [11] reported the same pattern for sequence diagrams.

More promising results were achieved when prompt design was optimized. Li et al. [12] demonstrated that GPT-3.5-turbo, guided by carefully structured chain-of-thought prompts, can outperform students in class diagram generation. Bragilovski et al. [13] compared novice and expert modelers with three automated methods—rule-based NLP, machine learning, and LLMs. They found that no single method consistently matched human performance, but LLMs performed on par with experts in class identification, while machine-learning approaches excelled at detecting associations.

Chen et al. [14] concluded that LLMs alone are insufficient for fully automated class diagram generation due to frequent omissions. Building on this insight, Yang et al. [25] proposed a multi-step approach that combines LLM queries with classical NLP, includes design pattern detection via dedicated prompts, and integrates self-review mechanisms. Their evaluation shows substantial performance improvements over single-step prompting strategies.

D. SysML Generation with LLMs

Recent work consistently shows that while LLMs are capable of translating natural language requirements into formal modeling artifacts, their effectiveness strongly depends on how semantic grounding, consistency, and validation are addressed.

A recurring challenge is the tendency of LLMs to generate syntactically plausible yet semantically incorrect model elements. To mitigate this, Qualis [18] propose a knowledge-grounded generation pipeline that tightly couples LLMs with a tri-layered knowledge graph. By combining reusable SysML modeling patterns, domain knowledge, and system-specific requirement-capability relations, their approach effectively constrains the generation space and reduces hallucinations.

Beyond the correctness of individual diagrams, maintaining consistency across multiple SysML views is another critical concern. Sultan and Apvrille [15] address this issue by integrating LLMs into a rule-based consistency management framework. Rather than replacing formal consistency rules, LLMs are used as a complementary mechanism to detect and resolve semantic mismatches between diagrams, such as use case and block diagrams. Their results indicate that LLMs are particularly valuable for identifying inconsistencies that are difficult to formalize, while rule-based checks remain essential for enforcing structural correctness.

While such hybrid approaches focus on improving robustness, empirical evaluations reveal fundamental differences in LLM performance across diagram types. Wang et al. [16] conduct a large-scale empirical study on the generation of SysML behavior models, showing that LLMs achieve high syntactic accuracy but struggle with semantic completeness, especially for interaction-heavy models such as sequence diagrams. Their analysis highlights that rule-based model checking can effectively correct formatting and grammar errors, but only partially addresses deeper semantic inconsistencies, underscoring the limitations of current feedback mechanisms.

Dehn et al. [17] demonstrate that carefully structured prompting is a key enabler for reliable SysML v2 generation. By explicitly encoding ontological knowledge, modeling conventions, and few-shot examples into the prompt, they significantly improve information extraction, traceability across architectural layers, and output robustness. Their results suggest that constraining the LLM's interpretive freedom is essential for transforming unstructured requirements into semantically meaningful and syntactically valid SysML models.

Taken together, these works indicate that SysML generation with LLMs is most effective when embedded in structured pipelines that combine prompt engineering, domain knowledge, and formal validation mechanisms. Rather than fully automating modeling, current approaches position LLMs as powerful assistants that can accelerate early modeling phases, support architectural reasoning, and provide high-quality starting points for expert refinement.

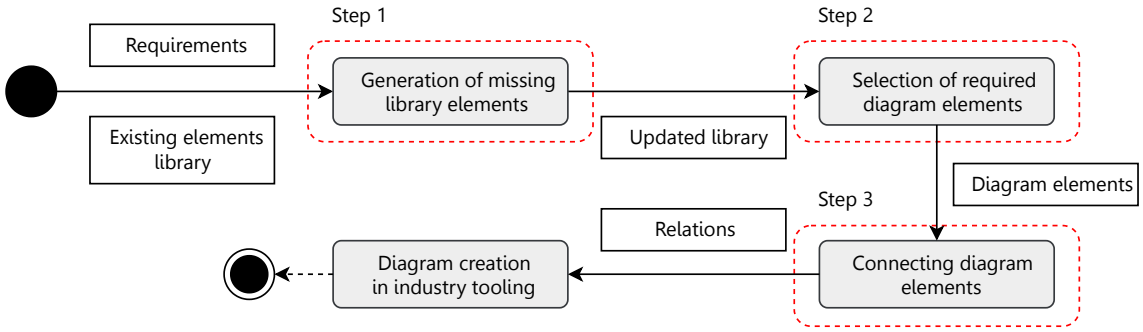

Fig. 2. Overview of the processing steps involved in our approach

E. Summary

Overall, classical NLP methods have demonstrated strong performance in activity diagram generation and have enabled the transfer of concepts from UML to SysML. Research on LLM-based diagram generation remains largely exploratory, focusing on holistic approaches with reduced pipeline complexity. While LLMs show promise, missing elements and semantic inconsistencies remain common challenges. Recent work emphasizes task decomposition, hybrid workflows, and refined prompting strategies as potential solutions. Evaluations are often limited to academic exercises or synthetic datasets.

In contrast, we investigate a multi-step LLM-based activity diagram generation using real-world industrial data, addressing larger diagram sizes and practical applicability.

IV. Approach

The existing approaches for generating and/or improving PlantUML code are not fully suitable for the industrial application considered, as they do not provide a sufficiently simple interface for integration with existing tooling. In industry models, many of the required elements are often already present because they are used in other diagrams. These elements should be unique and reused whenever possible. Therefore, our approach generates SysML activity diagrams from requirements while accounting for existing elements.

In our approach, described in Figure 2, a library of existing elements is first extended by possible missing elements from the requirements. After that, the elements needed for the use case are selected from the updated library. If an element was created in Step 1 but not selected in this step, it is removed from the library again to prevent unnecessary element creation. In Step 3, the relations between the selected elements from Step 2 are generated.

The models provided by the industry partner have previously been exposed to internal documentation during a fine-tuning process, enabling familiarity with the internal language and domain but limiting the reproducibility of the approach.

A. Generation of Missing Diagram Elements

For diagram element generation, we differentiate between the following element types: activities, actors, systems, and signals. We use two prompts to generate potentially missing diagram elements. The first prompt contains the SLRD and the library elements for a given element type, with the goal of identifying potential new candidates for diagram elements of this type. The LLM should avoid duplicates in this step. In the second prompt, the LLM should review its work by checking if newly identified elements can be replaced with other newly identified or preexisting elements. This results in a potentially updated library of existing diagram elements.

As with all the following steps, the prompts differ for SL and SSL. The differences in the prompts lie in the wording used to describe the generation goals and tasks.

B. Selection of Required Diagram Elements

In this step, the LLM identifies all required diagram elements from the updated library. Thus, this step has the greatest impact on model consistency. Similar elements can have different uses or applications, whereas elements that differ should often be grouped into a single element. Systems, actors, and activities at SL and SSL are identified by the LLM using separate prompts for each level. For this purpose, the LLM is provided with a list containing the names of all existing elements of the respective type in the library. The expected output is the names of the diagram elements needed to model the SLRD. These names are later mapped to the names of existing diagram elements based on the editing distance. This reduces errors from misspellings and LLM hallucinations.

The choice of the model can highly influence the outcome. Based on preliminary results, we decided to use OpenAI's GPT-4o for element identification.

C. Connecting Diagram Elements

To connect diagram elements, we use all selected elements from the previous step and consider two types of relations: relations between activities and relations between activities and systems. Note that, since actor-activity relations can be treated similarly to system-activity relations, we consider only system-activity relations, which include actor-activity relations as well. For the system-activity relation, a single prompt is used to assign the identified activities to the identified systems and actors.

The other relationship to examine is between activities. We distinguish two types of activity flows: 1) flows with a signal (item/object flow) and 2) flows without a signal (control flow).

The activity-activity relation is also generated from a single prompt that contains all identified activities and the SLRD as context. In the prompt, the LLM should generate the relations between the activities as single lines of the format (Start—End—Type). In this context, the flow type is distinguished as CF (Control Flow), OF (Object Flow), and IF (Item Flow). In addition, the LLM may insert flow-control elements. In the prompt, the LLM is instructed to insert start or end nodes as the source or target of a relation, denoted by `s` and `e`, respectively. Furthermore, the LLM may insert numbered joins (`j`) and forks (`f`) as start or end nodes of a relation to control execution flow. Note that we do not consider merge and decision nodes in this work. This is an explicit design decision made by the industry partner to avoid ambiguities in function implementation and functional decomposition. In particular, this avoids the question of which of the connected functions (activities) is responsible for implementing the decision. The number of identified signals defines the number of relations between the activities.

For connecting diagram elements, we make use of OpenAI's o1 model, as models with integrated chain-of-thought (CoT) reasoning are expected to achieve better results on logic-based tasks, such as assigning activities to systems.

D. Prompt Design

For each task, we designed a prompt following a predefined template while keeping the instructions as simple and consistent as possible. Depending on the task, the prompt requires the LLM to generate missing diagram elements, select required diagram elements from a library, or establish relations between diagram elements.

Each prompt is composed of five logical blocks:

- **Task Description**: A description of the problem and the task to be solved by the LLM.
- **Element Type Specification**: A definition of the target diagram element type together with a list of corresponding element candidates from the model library.
- **Example Structure**: A specification of the example structure, including an input–output example.
- **Domain Context**: The SLRDs or use case serving as the basis for the task.
- **Constraints**: Additional modeling constraints and assumptions that limit the scope of the generated output.
- **Output Format**: A specification of the required output representation.

An example prompt template for selecting the SL activities is shown in Figure 3.

Identification of activities on Level 1

Based on the use case, please select all actions required for a SysML activity diagram.
The possible actors or systems for the activity are [*List of possible actors and systems*].

Only provide super activities. If multiple activities can be substituted by a single activity and the activity is an possible activity provide only the super activity.
Remember to add all activities that provide input data or are needed for communication.

As an example given the use case [*Example use case*] with the [*Example actors*] as possible actors and the [*Example systems*] as possible system should return the following activities: [*Example activities*].

Given this Use Case: [*Use case*].
Given the following possible actor activities: [*Actor activities from the library*].
Given the following possible system activities: [*System activities from the library*].

Only provide activities that are directly linked to the use case. Assume all systems are already in the initial state required by the use case.
Do not provide activities that can be described or are fulfilled in a separate Use Case.

Answer only as a comma separated list of the needed activities.

Fig. 3. Prompt template for selecting SL activities (Step 2)

V. Evaluation

We evaluate our approach in two steps. First, we automatically compare the generated diagrams against reference diagrams from our industry partner on a per-step basis (Section V-B). Afterward, we conduct a survey with nine architects from our industry partner. The survey asks them to evaluate and rank sampled diagrams against the reference diagrams (Section V-C) and to reflect on their modeling methodology (Section V-D). Finally, we discuss the survey results with respect to our research questions (Section V-E).

A. Dataset

The dataset used for this evaluation comprises 18 SLRDs, containing a total of 727 individual requirements distributed across 72 use cases. Due to dependencies on organizational structures, quantitative information regarding the SL elements cannot be disclosed. At the SSL, the dataset library includes more than 1,000 activities distributed across more than 300 SSL systems. Furthermore, the reference model contains more than 4,000 interconnections at the SSL.

To evaluate this approach, more than 10% of the reference model elements were used. Due to the reuse of diagram elements, the resulting ground truth at the SSL comprises more than 300 systems and over 400 activities. Each activity is mapped to a corresponding system, and the activities are interconnected by more than 750 relations.

B. Direct Comparison to Existing Diagrams

We evaluate each of the three steps of our approach separately by directly comparing their outputs to the corresponding elements of the reference diagrams from our industry partner. This fine-grained comparison allows us to identify where

TABLE I
EVALUATION RESULTS FOR THE DIFFERENT LLMS IN THE SELECTION OF REQUIRED DIAGRAM ELEMENTS. THE RELATIONSHIPS BETWEEN ACTIVITIES AND SYSTEMS WERE TAKEN INTO ACCOUNT. FOR ACTORS, ONLY SL IS SHOWN, AS NO ACTORS WERE AND HAVE BEEN IDENTIFIED AT SSL.

	GPT-3.5-turbo			GPT-4o-mini			GPT-4o			o3-mini			o1		
Element	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
Actors SL	.39	.69	.50	.41	**.89**	.56	.66	.87	.75	**.81**	.81	**.81**	.71	.75	.73
Systems SL	.60	.68	.63	.62	.81	.71	.88	.84	.86	.83	.77	.80	**.90**	**.90**	**.90**
Systems SSL	.12	.19	.14	.25	.33	.29	**.36**	**.38**	**.37**	.34	.31	.33	.35	**.38**	**.37**
Systems comb.	.18	.27	.21	.31	.41	.36	**.44**	.46	**.45**	.42	.39	.40	**.44**	**.48**	**.45**
Activities SL	.02	.11	.03	.10	.27	.15	.29	.52	.37	.35	.38	.37	**.40**	**.53**	**.46**
Activities SSL	.01	.02	.01	.06	**.06**	.06	.11	**.06**	**.08**	**.14**	.05	**.08**	.06	.04	.05
Activities comb.	.01	.04	.02	.08	.11	.09	.20	**.17**	**.18**	**.24**	.13	.17	.19	.16	.17
Micro Avg. SL	.09	.37	.14	.23	.53	.32	.45	**.67**	.54	.55	.57	.56	**.56**	.66	**.61**
Micro Avg. SSL	.05	.09	.06	.16	.17	.16	.25	**.19**	**.22**	**.27**	.16	.20	.21	.18	.20
Micro Avg. total	.06	.16	.08	.19	.26	.22	**.33**	.31	**.32**	.26	**.37**	.30	.32	.30	.31

TABLE II
RESULTS FOR GENERATING MISSING DIAGRAM ELEMENTS (LOWER LEVENSHTEIN DISTANCE AND HIGHER EMBEDDING SIMILARITY ARE PREFERRABLE)

		Levenshtein Dist.		Embedding Sim.	
Elements	Level	GPT-4o	o1	GPT-4o	o1
Actors	SL	.79	**.76**	**.77**	**.77**
Systems	SL	**1.00**	**1.00**	**.62**	.60
	SSL	**.98**	1.00	**.70**	.64
Activities	SL	**1.00**	**1.00**	**.66**	.62
	SSL	**1.00**	**1.00**	**.66**	.62

errors are introduced and which steps benefit most from prompt engineering or model selection.

a) Generation of Diagram Elements: To evaluate element generation, we remove individual elements from the library and task the LLM with regenerating them based on the requirements. We then measure the normalized Levenshtein distance and the cosine similarity of vector embeddings between the generated and the original elements. For the embedding-based similarity, we use OpenAI's `text-embedding-3-large` model. We remove each element in the dataset exactly once, keeping all other elements intact, so that every element contributes one evaluation instance. Moreover, we compare the results from an instruct model (GPT-4o) with those from a reasoning model (o1).

The results in Table II indicate that the normalized Levenshtein distance for systems and activities is close to one, suggesting that the generated names differ substantially from the original ones. This discrepancy is most commonly caused by overspecification, such as generating *vehicle driver* instead of the more concise *driver*. Additionally, actors tend to benefit from shorter name lengths under this metric. Embedding-based similarity ranges from 0.6 for high-level systems to 0.77 for actors, indicating that generated names are semantically related but not identical to the originals. Notably, SSL elements achieve a slightly higher cosine similarity than SL elements, which we attribute to the same overspecification behavior on SL. In general, GPT-4o performed better than the reasoning model o1 on this task, as it achieved lower or comparable Levenshtein distances for most element types and higher embedding similarity across all elements.

b) Selection of Diagram Elements: Next, we evaluate element selection by comparing the activities identified by the LLM against the reference diagram. An activity is counted as correct only if it is assigned to the correct system. As metrics, we report precision, recall, and F_1-score. For this step, we additionally compare GPT-3.5-turbo, GPT-4o mini, and o3-mini to evaluate if smaller and/or less expensive models perform comparably to the larger and/or more expensive ones.

Table I presents precision, recall, and F_1-score per model for actors, systems, and activities at both SL and SSL, as well as micro-averaged totals. Overall, GPT-4o achieves the highest micro-averaged F_1-score ($F_1 = 0.32$), followed closely by o1 ($F_1 = 0.31$) and o3-mini ($F_1 = 0.30$).

However, F_1-scores of about 30% are not sufficient, especially if the recall is as low as well. This is mainly due to performance differences across levels. For elements on SL, the overall performance is more than double the performance on SSL (54% vs. 27% F_1-score for GPT-4o). The selection of activities performs worse than the selection of systems and actors across levels. This can partly be explained by many activities being assigned to the wrong system. Ignoring the system association, activities combined increase to 35% F_1-score and activities on SL even to 51% F_1-score with GPT-4o.

Model choice has a clear impact: more recent models consistently outperform older ones, confirming outcomes of similar studies [26]. GPT-4o achieves particularly strong results relative to its recency, likely due to its role in prompt tuning

TABLE III
EVALUATION RESULTS FOR THE DIFFERENT LLMS FOR THE TASK OF RELATION GENERATION. WE REPORT RESULTS WITH IGNORING THE TYPE OF THE GENERATED RELATION (W/O TYPE), ONLY MEASURING THE CONNECTED ELEMENTS AND RELATIONS THAT WERE ENTIRELY CORRECT.

	GPT-3.5-turbo			GPT-4o-mini			GPT-4o			o3-mini			o1		
Element	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
Relations w/o type SL	.19	.15	.17	.37	.37	.37	.69	.54	.60	**.70**	**.63**	**.66**	.68	.61	.64
Relations w/o type SSL	.08	.06	.07	.25	.15	.18	.31	.21	.25	.22	.18	.20	**.34**	**.27**	**.30**
Relations w/o type comb.	.10	.08	.09	.29	.20	.24	.40	.29	.34	.34	.29	.31	**.43**	**.34**	**.38**
Relations SL	.13	.11	.12	.33	.33	.33	.60	.47	.53	**.61**	**.55**	**.58**	**.61**	**.55**	**.58**
Relations SSL	.03	.03	.03	.10	.06	.08	.12	.08	.10	.12	.10	.11	**.22**	**.16**	**.19**
Relations comb.	.06	.05	.05	.18	.12	.15	.25	.17	.21	.25	.21	.23	**.32**	**.25**	**.28**

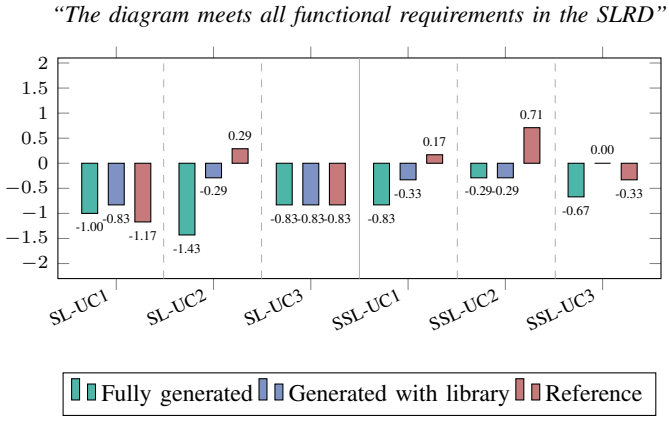

Fig. 4. Results of the survey on whether the diagrams meet the requirements. Higher values indicate that, on average, the architects believe the diagram better fulfills the requirements. The numerical values are based on the Likert scale. A score of 2 indicates full agreement with the statement The diagram meets all functional requirements in the SLRD. A value of -2 indicates complete disagreement.

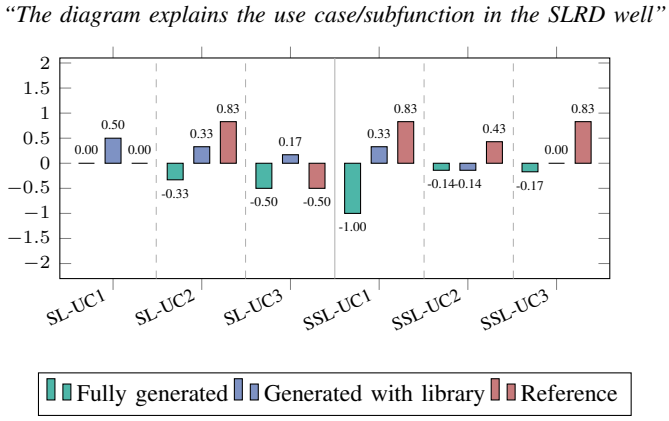

Fig. 5. In this figure, the survey results on whether the diagrams explain the use case or subsystem well are shown. A higher value indicates that, on average, the architects believe the diagram explains the use case or subfunction better. The numerical values are based on a Likert-scale evaluation, where a score of 2 indicates full agreement with the statement The diagram explains the use case or the subfunction in the SLRD well, and a score of -2 indicates complete disagreement.

during development. Notably, o3-mini, being less expensive than o1, achieves results comparable to the larger reasoning model and produces the best overall results for selecting actors. However, overall, GPT-4o performs better and requires far fewer tokens than the reasoning models. Based on these results, we select GPT-4o with zero-shot prompting as the final configuration for the expert evaluation.

c) Generation of Relations: We evaluate relation generation against two criteria: (1) whether the connected elements match the reference, and (2) whether the connection type and assigned signal are correct. A relation is considered fully correct only if both criteria are met. We compare the five different LLMs used in the previous step in a zero-shot setting.

Table III reports precision, recall, and F_1-score for each model across two criteria: relations ignoring the connection type (w/o type) and fully correct relations including the assigned type and signal, each broken down by SL, SSL, and combined. o1 achieves the highest combined F_1-score both without type ($F_1 = 0.38$) and with type ($F_1 = 0.28$), with o3-mini and GPT-4o following closely on the w/o-type metric. Only on SL does o3-mini perform better without type and comparable with type. Again, we see a drop in performance from SL to SSL. For relation generation, this decrease is even greater than for selecting elements, dropping from 58% to 19% for o1. This shows again that generating SL diagrams might be feasible in the near future, while the more detailed SSL seems out of reach at the moment. In general, models with an integrated CoT mechanism perform better than those without. Thus, we select o1 for the expert evaluation.

C. Survey: Diagram Evaluation

Nine architects from our industry partner participated in a survey to evaluate the quality of the generated diagrams. Each participant was shown three use cases: a simple scenario (UC1), a complex scenario (UC2), and a scenario (UC3) whose reference diagram was created without any survey participant's involvement. For each use case, three activity diagrams were presented at both SL and SSL without revealing their origin: (1) a fully generated diagram, (2) a library-based

TABLE IV
THIS TABLE SHOWS THE AVERAGE RANK OF THE DIAGRAMS. THE DIAGRAMS ARE RANKED FROM 1 (BEST) TO 3 (WORST).

	Use Case 1		Use Case 2		Use Case 3	
Diagram	SL	SSL	SL	SSL	SL	SSL
Fully generated	2.17	3.00	2.43	2.00	2.00	2.33
Generated with lib	**1.33**	1.67	2.29	2.86	**1.83**	2.17
Reference	2.50	**1.33**	**1.29**	**1.14**	2.17	**1.50**

diagram using the reference elements available in the library, and (3) the reference diagram. Architects rated each diagram on a five-point Likert scale (-2 = "strongly disagree" to $+2$ = "strongly agree") for two statements:

(i) *the diagram fulfills all functional requirements* and (ii) *the diagram explains the use case well*. In addition, architects ranked the three diagrams per use case and abstraction level by overall quality.

a) Requirement Fulfillment: Figure 4 presents the mean ratings per diagram and use case. At SL, only one diagram received a neutral or better score: the UC2 reference ($\mu = 0.29$); all others were rated negatively. For UC1 and UC3 on SL, the reference diagram was rated similar to or even worse than the generated ones. At SSL, the reference diagrams of UC1 and UC2 received positive ratings, while all generated diagrams were rated negatively. For UC3, the library-based generated diagram was rated neutral, while the fully generated and reference diagrams were rated negatively.

b) Use Case Explanation: Figure 5 shows the mean ratings for "The diagram explains the use case well," question. At SL, library-based diagrams for UC1 and UC3 received the highest ratings, whereas the reference diagram received the highest rating for UC2. On SSL, the reference diagram was rated highest for each use case. While for UC2 and UC3 the generated diagrams were rated neutrally, for UC1 the fully generated was rated consistently negative.

c) Overall Ranking: Table IV presents the average ranks per diagram (lower = better). At SL, library-based diagrams rank particularly well. On SSL, reference diagrams are ranked clearly higher than generated diagrams.

Another general observation is the high variance in architects' responses: only four of all eighteen diagrams were not ranked first by at least one architect.

Table V shows the F_1-score of the diagrams evaluated by the architects. The F_1-score was calculated as micro average of all diagram elements and relations against the reference.

D. Survey: Method Evaluation

In addition to evaluating the diagrams, the survey asked architects about their modeling practices using two Likert statements (-2 to $+2$): (i) *"I base the modeling of new features heavily on previously created models or diagrams"*, assessing whether diagram suggestions can be meaningfully integrated into the development process; and (ii) *"I find it complex to take previously created elements into account when modeling"*, assessing how disruptive architects perceive referencing existing library elements. A third open-ended question asked participants to name the most significant current problem they face while modeling the requirements.

TABLE V
MICRO AVG. F_1-SCORE OF THE DIAGRAMS SHOWN TO THE ARCHITECTS.

	Use Case 1		Use Case 2		Use Case 3	
Diagram	SL	SSL	SL	SSL	SL	SSL
Fully generated	0.09	0.00	0.09	0.08	0.00	0.00
Generated with lib	0.71	0.17	0.33	0.09	0.26	0.14

For statement (i), the mean score was 1.22, with no architect rating below 1. For statement (ii), the mean score was 0, indicating neither agreement nor disagreement. Free-text responses to the third question were clustered by keyword. Architects most frequently cite requirements themselves as a core problem in modeling. A second recurring theme was consistent modeling, i.e., maintaining alignment between requirements and diagrams and adhering to a uniform level of abstraction. All responses were independent of the architect's modeling experience.

E. Discussion

We now discuss the survey results with respect to our three research questions.

RQ1: Consistency of SysML activity diagrams generated by LLMs. LLMs produce good results for simple diagrams but do not yet reach the quality of experienced architects for complex structures, consistent with the automated comparison in Section V-B. Notably, diagram quality, as assessed by architects, does not necessarily correlate with automated matching scores; for example, the *Use Case 3* was rated high quality at SL by experts despite showing a low F_1-score in the automated evaluation. The step-wise evaluation further reveals a tendency to overspecify: on average 3.72 elements were generated per requested element, which reduces the apparent precision of the generation step, and fully generated SL diagrams were often perceived as closer to SSL abstraction. From the diagram evaluation (Section V-C), no diagram achieved full agreement on requirement fulfillment, including the reference diagrams themselves, reflecting the inherent difficulty of encoding requirements in a diagram. The widening gap between generated and reference diagrams at SSL suggests that complexity amplifies generation deficiencies. In follow-up discussions, architects noted that different diagrams failed to capture distinct subsets of requirements, making direct comparisons difficult. Differences between generated and reference diagrams are small at SL, while reference diagrams remain clearly superior at SSL. Generated diagrams are competitive in explaining use cases at SL, but fall behind reference diagrams at SSL for more complex scenarios. In follow-up discussions, architects noted that the generated diagrams contain many

accurate and interesting modeling approaches, but also include disconnected activities or a mix of elements from SL and SSL systems. The high variance across architects' ratings and the fact that even reference diagrams were not consistently ranked first underscore the subjective nature of diagram quality assessment.

RQ2: Factors impacting LLM performance in activity diagram generation. LLMs also frequently fail to adhere to predefined output formats such as JSON or CSV; these formatting errors and their corrections can affect the resulting diagrams. The relatively stronger performance of o1 and o3-mini in relation generation, compared to diagram element identification, indicates that CoT models outperform non-CoT models on tasks requiring reasoning.

RQ3: Impact on architecture quality in an industrial setting. From the method evaluation (Section V-D), the strong and unanimous agreement on reliance on existing diagrams confirms that providing architects with reference diagrams is methodologically sound, and that high-quality generated diagrams could serve as useful starting points. The neutral rating on statement (ii) indicates that incorporating previously created elements into modeling is not a pressing concern for the architects. The identified challenges, namely requirements quality and consistent abstraction levels, apply equally to automated approaches, which share the same requirements as input. From the diagram evaluation (Section V-C), generated diagrams are competitive with manual ones for simple scenarios, and follow-up interviews revealed that even lower-quality diagrams may better represent individual requirements. Consequently, automated diagram generation can improve architectural quality by providing novel design perspectives and a foundation for manual refinement.

VI. Threats to validity

This study is subject to several threats to validity, primarily stemming from the stochastic behavior of LLMs, the construction of reference artifacts, and the restricted evaluation context.

First, LLMs inherently exhibit non-deterministic response behavior, which affects reproducibility. To mitigate this threat, each F_1-score calculation was executed 20 times, and the results were averaged. While this procedure reduces random variance, it does not eliminate systematic biases introduced by model sampling behavior. Reproducibility is further limited by the use of proprietary industry-partner models that were fine-tuned on an undisclosed amount of internal documentation. As these models are not publicly accessible and their training distributions are unknown, independent replication of the evaluation is constrained.

Second, LLMs are known to produce hallucinated or semantically imprecise outputs. To address this issue, we implemented a structured mapping procedure that aligns LLM outputs with diagram elements. However, this mapping itself introduces a potential source of error, as mismatches between generated content and diagram entities may distort the computed metrics. We mitigated this risk by empirically determining a sufficiently high similarity threshold to balance false positives and false negatives. Nevertheless, the mapping mechanism remains a methodological sensitivity that may influence quantitative results.

Third, the validity of the evaluation depends on the quality of the reference diagrams and SLRDs. Since these artifacts were manually created, they may contain inaccuracies, inconsistencies, or omissions. Consequently, a mismatch between a generated diagram and its reference does not necessarily indicate an incorrect result, as the reference itself may reflect human error or ambiguous interpretations of requirements. To reduce reliance on any single artifact, the evaluation combines automated F_1-score-based comparisons with rankings provided by experienced architects. Furthermore, 18 independently created and reviewed SLRDs were included to distribute the risk of individual modeling errors. Despite these precautions, residual bias originating from human-created reference artifacts cannot be fully excluded.

We also emphasize the lack of reliable evaluation criteria. The F_1-score is not necessarily correlated with its classification by human architects. Empirical human evaluations further revealed a high variance in the assessment results.

The study was conducted with a single industry partner in the domain of vehicle systems development. Domain-specific modeling conventions, standards, and architectural practices may have influenced both the generated diagrams and their evaluation. Consequently, the generalization of the results to other domains or modeling environments is limited.

Overall, the most substantial risks to validity arise from limited reproducibility due to proprietary, fine-tuned models and stochastic model behavior, and from dependence on manually created reference artifacts that may themselves contain imperfections. While mitigation measures were implemented to reduce these risks, they remain primary factors that could affect robustness and transferability of the reported results.

VII. Industry Application

In response to this study, we decided to implement additional assistance mechanisms for architects, including LLM-based support for selecting diagram elements and providing naming recommendations. In addition, similarity checks between diagram elements were implemented using vector embeddings to improve structural consistency and the detection of reuse.

The approach demonstrates potential to reduce time spent on both architectural diagram creation and training architects to adhere to strict modeling conventions. Automated support for diagram element identification, structural validation, and guideline compliance may substantially shorten modeling cycles, decrease manual revision effort, and reduce inconsistencies. Moreover, the provision of automated feedback on guideline violations and structural deficiencies suggests strong applicability in educational and on-boarding contexts, potentially accelerating the learning curve of junior architects while improving overall model quality and consistency.

However, a fully automated generation process without human correction does not appear feasible based on the

current results. Further limitations arise from the inherent constraints of contemporary LLM architectures, particularly token usage limits and restricted context length. Large and complex architectural diagrams often exceed practical context windows. Consequently, human validation remains necessary to ensure correctness, consistency, and compliance with modeling standards.

VIII. Conclusion

This paper presents a multi-step approach for automatically generating SysML activity diagrams using LLMs. By answering **RQ1**, this paper was able to demonstrate how well LLMs can create SysML activity diagrams in a real-world environment. Compared to the industry standard, the diagrams are positively evaluated by architects at SL. For more complex diagrams, however, the architects preferred the reference diagrams.

The automatic evaluation of the steps, with F_1-scores of 32% for diagram element selection and 28% for relation generation, indicates that the methodology is usable but does not achieve high agreement with the reference diagrams. The evaluation of the architects shows that the impact of low F_1-scores may not be critical for the quality or usability of the diagrams. Furthermore, performance on simpler diagrams was higher, but the approach requires further optimization for complex ones. The requirement fulfillment of the generated diagrams is at a similar level to that of the reference diagrams. As part of **RQ2**, this study identified the factors that influence performance in architecture generation. Prompting technique and the choice of LLM affect diagram quality metrics.

A survey with industry partner architects validated, with respect to **RQ3**, that a tool for the automated generation of activity diagrams can improve architectural design, as many architects rely on existing diagrams. In combination with the analysis results, the proposed method can support architects by generating diagram suggestions.

Based on the results, the use of this approach in a real-world environment is only partially reasonable. However, the findings show that the approach can serve as a source of inspiration for architects. For example, using individual steps, such as selecting diagram elements, can support architects in their work. The results also appear promising, particularly for less complex diagrams.

The paper indicates that further work is needed, particularly on activity selection in complex diagrams. Research in two areas is particularly promising: by integrating classical NLP methods into the identification process, additional validation loops could be introduced into diagram element selection. In addition to classical NLP methods, explicit steps, similar to those used by Yang et al. [25], could be incorporated to ensure that diagram element selection considers specific design patterns or architectural rules, such as the separation of controller and manager systems.

This paper did not focus on the quality requirements of the use cases and their specifications. An investigation into the relationship between the quality of requirements and the quality of generated diagrams, similar to work on traceability link recovery [27], could provide deeper insights into the challenges of automated UML and SysML diagram generation. This is supported by interviews with architects, in which four of the respondents described requirement definition and traceability as the biggest challenges in modeling. Recent work on LLM-based traceability link recovery [28], [29] may offer complementary solutions for this challenge.

Likewise, this paper did not evaluate the approach's economic efficiency. Questions regarding potential time savings or cost reductions in the architecture creation process remain unanswered. Similarly, no cost analysis of the method was performed, which is particularly relevant when considering industrial deployment.

Lastly, agentic workflows and teammates are on the rise and promising candidates for various tasks in the software development lifecycle [30]. Accordingly, tackling this problem in an agentic way with cycles, self-criticizing and feedback, tool usage as well as using them for knowledge management [31] may result in major improvements.

Acknowledgments

This work was funded by Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF). It was supported by the German Research Foundation (DFG) – SFB 1608 – 501798263, by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the National Research Data Infrastructure – NFDI 52/1 – 501930651, and by KASTEL Security Research Labs, Karlsruhe.

References

[1] T. Weilkiens and R. M. Soley, *Systems Engineering mit SysML/UML: Anforderungen, Analyse, Architektur*, 3rd ed. dpunkt.verl.

[2] S. Ahmed, A. Ahmed, and N. U. Eisty, "Automatic transformation of natural to unified modeling language: A systematic review," in *2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA)*, pp. 112–119.

[3] M. Ibrahim and R. Ahmad, "Class Diagram Extraction from Textual Requirements Using Natural Language Processing (NLP) Techniques," in *2010 Second International Conference on Computer Research and Development*, May 2010, pp. 200–204.

[4] Y. Meng and A. Ban, "Automated UML class diagram generation from textual requirements using NLP techniques," *JOIV : International Journal on Informatics Visualization*, vol. 8, no. 3, pp. 1905–1915.

[5] T. Yue, L. C. Briand, and Y. Labiche, "An automated approach to transform use cases into activity diagrams," in *Modelling Foundations and Applications*, T. Kühne, B. Selic, M.-P. Gervais, and F. Terrier, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 337–353.

[6] A. M. Maatuk and E. A. Abdelnabi, "Generating uml use case and activity diagrams using nlp techniques and heuristics rules," in *International Conference on Data Science, E-Learning and Information Systems 2021*, ser. DATA'21. New York, NY, USA: Association for Computing Machinery, 2021, p. 271–277.

[7] S. Zhong, A. Scarinci, and A. Cicirello, "Natural language processing for systems engineering: Automatic generation of systems modelling language diagrams," *Knowledge-Based Systems*, vol. 259, p. 110071, Jan 2023.

[8] J. Cámara, J. Troya, L. Burgueño, and A. Vallecillo, "On the assessment of generative ai in modeling tasks: an experience report with chatgpt and uml," *Software and Systems Modeling*, vol. 22, no. 3, p. 781–793, 2023.

[9] S. Arulmohan, M.-J. Meurs, and S. Mosser, "Extracting domain models from textual requirements in the era of large language models," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, Oct 2023, p. 580–587.

[10] D. De Bari, G. Garaccione, R. Coppola, M. Torchiano, and L. Ardito, "Evaluating large language models in exercises of uml class diagram modeling," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 393–399.

[11] A. Ferrari, S. Abualhaija, and C. Arora, "Model generation with llms: From requirements to uml sequence diagrams," in *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*, 2024, pp. 291–300.

[12] Y. Li, J. Keung, X. Ma, C. Y. Chong, J. Zhang, and Y. Liao, "Llm-based class diagram derivation from user stories with chain-of-thought promptings," in *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2024, pp. 45–50.

[13] M. Bragilovski, A. T. van Can, F. Dalpiaz, and A. Sturm, "Leveraging machines to derive domain models from user stories," *Requirements Engineering*, vol. 30, no. 2, p. 241–262, Apr 2025.

[14] K. Chen, Y. Yang, B. Chen, J. A. Hernández López, G. Mussbacher, and D. Varró, "Automated domain modeling with large language models: A comparative study," in *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2023, pp. 162–172.

[15] B. Sultan and L. Apvrille, "Ai-driven consistency of sysml diagrams," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 149–159.

[16] Y. Wang, N. Ge, J. Liu, Z. Cao, Z. Chen, and C. Hu, "Generating sysml behavior models via large language models: an empirical study," in *Proceedings of the 16th International Conference on Internetware*, ser. Internetware '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 366–377.

[17] S. Dehn, S. Schnürer, G. Jacobs, and G. Höpfner, "Generating sysml v2 models from natural language requirements using large language models," in *2025 IEEE International Symposium on Systems Engineering (ISSE)*, 2025, pp. 1–7.

[18] R. A. Qualis, "Mitigating hallucinations in sysml v2 generation using llms and a tri-layered knowledge graph reasoning framework," in *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2025, pp. 357–366.

[19] L. Schmid, T. Hey, M. Armbruster, S. Corallo, D. Fuchß, J. Keim, H. Liu, and A. Koziolek, "Software architecture meets llms: A systematic literature review," 2025.

[20] "Omg systems modeling language 1.6." [Online]. Available: https://www.omg.org/spec/SysML/1.6

[21] T. Fuchs, *Graphenbasierte Methodik zur virtuellen Absicherung von mechatronischen Fahrzeugfunktionen mittels digitaler Zwillinge*, 1st ed. München: Verlag Dr. Hut, 2024.

[22] T. Fuchs, M. Zinser, K. Renatus, and B. Bäker, "Automotive digital twins: A traversal algorithm for virtual testing of software over-the-air updates," in *2023 IEEE International Conference on Mechatronics (ICM)*, 2023, pp. 1–6.

[23] E. A. Abdelnabi, A. M. Maatuk, and T. M. Abdelaziz, "An algorithmic approach for generating behavioral uml models using natural language processing," in *The 7th International Conference on Engineering & MIS 2021*, ser. ICEMIS'21. New York, NY, USA: Association for Computing Machinery, 2021.

[24] M. Robeer, G. Lucassen, J. M. E. M. van der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated extraction of conceptual models from user stories via nlp," in *2016 IEEE 24th International Requirements Engineering Conference (RE)*, 2016, pp. 196–205.

[25] Y. Yang, B. Chen, K. Chen, G. Mussbacher, and D. Varró, "Multi-step iterative automated domain modeling with large language models," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS Companion '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 587–595.

[26] M. Soliman, E. Ashraf, K. M. K. Abdelsalam, J. Keim, and A. P. S. Venkatesh, "Llms for software architecture knowledge: A comparative analysis among seven llms," in *Software Architecture*. Cham: Springer Nature Switzerland, 2026, pp. 99–115.

[27] T. Hey and J. Frattini, "How Requirements Quality Makes (or Breaks) Traceability Link Recovery," in *2026 IEEE 34th International Requirements Engineering Conference (RE)*, Aug. 2026.

[28] D. Fuchß, T. Hey, J. Keim, H. Liu, N. Ewald, T. Thirolf, and A. Koziolek, "Lissa: Toward generic traceability link recovery through retrieval-augmented generation," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 2025, pp. 1396–1408.

[29] D. Fuchß, H. Liu, S. Corallo, T. Hey, J. Keim, J. von Geisau, and A. Koziolek, "Who's who? llm-assisted software traceability with architecture entity recognition," *ACM Trans. Auton. Adapt. Syst.*, Apr. 2026, just Accepted.

[30] D. Amalfitano, A. Metzger, M. Autili, T. Fulcini, T. Hey, J. Keim, P. Pelliccione, V. Scotti, A. Koziolek, R. Mirandola, and A. Vogelsang, "A research roadmap for augmenting software engineering processes and software products with generative ai," *ACM Trans. Softw. Eng. Methodol.*, Jan. 2026.

[31] J. Keim and A. Kaplan, "From scattered to structured: A vision for automating architectural knowledge management," in *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE)*. ACM, 2026.