Comparing Approaches for the Automatic Generation of System Models in Generative Systems Engineering

Martin Becker[1*], Damun Mollahassani[1], Jens C. Göbel[1]

[1]Rheinland-Pfälzische Technische Universität Kaiserslautern, Lehrstuhl für Virtuelle Produktentwicklung, 67663 Kaiserslautern

*Corresponding author. E-Mail address: martin.becker@mv.rptu.de; Tel.:0631 205 3787

Abstract: The reuse and adaptation of SysML modules and model fragments is central to efficient model-based systems engineering, yet it doesn't remain easy in practice due to heterogeneous documentation, fragmented repositories, and the high effort required to produce syntactically valid and tool-agnostic models. Large language (LLM) models offer automation potential, but unconstrained generation produces hallucinations and syntactically invalid SysML v2 that cannot be parsed or visualized. This paper proposes an approach to automatically selecting generation assistance in LLM-assisted SysML v2 modeling based on prompt complexity and legacy model hints. For this approach, retrieval augmented generation (RAG) that injects task-specific SysML v2 fragments and domain hints into the prompt, grammar constrained decoding (GCD) based on a lightweight SysML v2 grammar subset, and rule-based post processing (PP) with repair loops are used and compared. The result is a robust approach to engineering assistance that leverages the LLM's existing knowledge, aiming to generate reasonable, syntactically valid SysML v2 Notation. Results indicate that RAG is best used with extensive prompts and detailed product knowledge. At the same time, PP improves syntactic correctness for output close to correct and GCD for short prompts and limited knowledge.

Keywords: AI, LLM, SysMLv2

1 Introduction

The transition from traditional, domain-specific products to highly complex, interdisciplinary smart products demands development processes that are both agile and robust [1]. This shift substantially increases the multidisciplinary complexity of product development and cross-domain collaboration, while also necessitating shorter development cycles and increased cost efficiency [1]. In this dynamic environment, innovation is typically not characterized by radical new developments, but rather by iterative improvements based on pre-existing products [2]. Consequently, the ability to effectively manage and reuse existing engineering knowledge becomes a critical competitive advantage. However, as the complexity of these products increases, the manual adaptation of existing solutions becomes progressively more resource-intensive, requiring deep interdisciplinary expertise, which many organizations struggle to maintain.

MBSE addresses these challenges by representing requirements and architectural relationships in formal, human, and machine-readable models that support consistency across the product lifecycle. However, for iterative development, adapting existing model structures for new projects remains manual mainly and is aggravated by communication barriers and heterogeneous nomenclature across organizations and value creation networks. LLMs can reduce the effort required to create model artifacts from natural-language input [3], but their introduction presents technical hurdles. Existing strategies for reusing and reintegrating model fragments into new system models often lack sufficient automation [4], depend on manual search mechanisms [5], or adhere to rigid formalisms that limit the flexibility required

in early design phases [6]. Furthermore, while generative AI has shown potential for generating initial model fragments [7], it currently lacks the semantic grounding required to reliably modify specific legacy data without the risk of syntactic errors. However, multiple approaches exist that can improve the reliability of syntactically correct code generation. These need to be carefully selected, depending on the required task, and provided with additional information

This paper proposes an approach that automatically selects a mode of adapting the LLM output generation, reusing and adapting SysML v2 modules and model fragments. The approach generates SysML v2 textual notation that is directly parseable by engineering tools. Based on the prompt characteristics and available legacy hints, the selector selects one of four output adaptation techniques. The LLM is treated as a proposal generator, while a systems engineer performs final verification and decides whether the fragment can be reintegrated into the larger model.

The Design Research Methodology [8], was used to provide the critical context of the approach, therefore the paper follows the following structure. Chapter 2 details the current state of the art and the specific technical challenges associated with LLM-based modeling. Chapter 3 presents the investigative setup and the different strategies applied to enhance LLM performance, including the technical integration and grammar constraints. Chapter 4 provides a detailed validation of these capabilities through a descriptive study involving the incremental development of an electric mountain bike based on an existing electric city bike model. Finally, Chapter 5 concludes with a summary of the findings and an outlook on future research directions concerning the automation of systems engineering tasks.

2 State of the Art

Smart products integrate various engineering disciplines with software development and service design, challenging established product-development practices by increasing integration and coordination demands across heterogeneous stakeholders and subsystems. Abramovici et al. [9] note that this interconnectivity significantly increases development time and costs, as traditional methods require managing seamless integration across heterogeneous disciplines within various subsystems. In addition to these challenges, Albers et al. [2] emphasize that the majority of product innovations are iterations of established products and, therefore, product generations, rather than radically new developments without any prior product knowledge to build upon. However, the manual adaptation of this legacy knowledge is often hindered because the required interdisciplinary expertise is frequently unavailable or siloed within specific departments [4], or because individuals do not share their knowledge due to personal relations, culture, or identification as professional colleagues [10].

This siloing is a direct consequence of the complexity of smart products, which compels stakeholders to collaborate in Value Creation Networks (VCNs) to access complementary capabilities [11]. While this collaboration enables the development of advanced systems, it simultaneously fragments engineering knowledge across different organizations and stakeholders. This can result in the loss of access to engineering knowledge due to communication issues, differences in discipline-specific nomenclatures and understanding, and stakeholder-specific perspectives on challenges [12].

MBSE offers a solution to centralize this knowledge and enable holistic system understanding among various stakeholders, even when different disciplines are involved, but Chami and Bruel [13] highlight that steep learning curves and specialized nomenclature reduce acceptance in industry and frequently hinder its adoption despite MBSE's potential. Akundi et al. [14] add that organizational resistance and technical barriers prevent its widespread acceptance among non-experts. Henderson and Salado [15] therefore show that reuse remains inconsistent across industries because most organizations lack the infrastructure to manage model libraries effectively. Nonetheless, Madni and Sievers [16] indicate that

effective model reuse can be a primary driver for cost reduction in MBSE and, therefore, iterative product development.

Consequently, attempts have been made to structure models for greater reusability, resulting in less effort required to generate new system models. Mahboob and Husung's [17] framework for describing SysML model structures as an example focuses on describing reusable elements but lacks a mechanism to automate the actual reuse process. This leaves the systems engineers with an increased workload and therefore does not sufficiently help with the adoption of the process. Focusing on retrieval, Mendieta et al. [18] developed an approach to identify and find past model fragments relevant to current projects. While this delivers potential solutions, the retrieved artifacts are not adapted or modified for the new use case, limiting the level of automation, in an attempt to automate modeling efforts and deliver significant automation and relief to the systems engineers, Biase et al. [19] utilized Behavior Driven Development (BDD) to automate the generation of state machines. Although precise, the reliance on strict formalism limits the approach's applicability solely to state machines, rendering it unsuitable for holistic system adaptation. The emergence of LLMs as a readily available technology has increased their adoption in almost all aspects of life and introduced new possibilities and challenges for engineers. Ariffud [20] shows broad adoption of LLMs across engineering disciplines. Despite this potential, Kretzschmar et al. [3] warn that hallucinations restrict LLM usage in critical paths. To apply AI to MBSE, Akundi et al. [21] introduced a text-to-model framework generating SysML parts from natural language and therefore making SysML and systems engineering more approachable. However, this approach does not utilize LLMs or the flexibility that modern LLMs offer users. Additionally, SysML v2, as the coming standard, has not been used. Similarly, Schleifer et al. [7] presented an LLM-based pipeline that effectively generates use case diagrams directly from requirements. Yet, this solution is limited to the initial requirements phase and does not support modifications to complex architectural models later in the lifecycle. Furthermore, the use case diagrams are not generated directly in SysMLv2 textual notation for immediate rendering, requiring additional steps to generate the diagrams. Addressing code generation, Kourani [22] proposed a framework using LLMs and RAG to generate executable code for process modeling. While effective for code generation, the strict adherence to the code generation, code execution, and code repair pattern is applied only to process engineering tasks, not to model-based engineering tasks. Furthermore, the approach is not compared to alternative approaches of ensuring code validity.DeHart [23] developed a methodology enabling LLM interaction with SysML v2 via Python scripts, while using human feedback to ensure valid output in an engineering context. However, this approach relies on advanced LLMs and customized LLM assistants, not on automatic code validation. This limits the degree of automation achieved in the modeling support of systems engineers. Addressing the circularity of engineering assets, Mollahassani et al. [24] introduced a framework for AI-driven MBSE to enhance sustainability and model reuse. While this framework establishes the theoretical basis for circularity, it does not yet fully detail the interactive, agentic workflows required to mitigate LLM hallucinations during the specific adaptation process. Finally, Johns et al. [25] developed the AI Systems Modeling Enhancer, a plug-in allowing an LLM to model directly in Catia Magic. This allows a partial automation of modeling tasks using AI. The approach, however, does not use the more modern SysMLv2 language and, as a plug-in for Catia Magic, is vendor-specific. This severely limits the number of applications supported.

Therefore, this paper discusses the application of LLMs and especially small open-source LLMs in the role of active modeling assistance in MBSE using SysMLv2, during the iterative product development. To make the assistance independent from specific vendors, SysML v2 textual notation must be generated directly. Since the required textual notation can be generated using various approaches with varying results, this paper presents an automated selection to support engineers with the best option for generating valid SysML v2 textual notation under the given circumstances. This assumes the existence

of established product knowledge, which the LLM can reuse to generate new, adapted system models and thereby relieve systems engineers during product development. This results in the following research questions:

- How can SysML v2 code be automatically generated, and how do the different approaches affect the model quality?
- Under which circumstances should which generation method be used?

3 Approach for Ensuring Syntactical Validity of SysML v2 Models

This paper focuses on code generation for model reuse in MBSE using SysML v2. As identified in the previous chapter, LLMs can generally support modeling efforts by serving as a translation layer between natural-language requirements and formal engineering specifications. This capability significantly lowers the entry barrier for the engineers involved in the product development process, who may not be fluent in formal modeling languages. This can, therefore, increase acceptance of MBSE and SysML. However, SysML v2 presents a unique challenge to LLMs because it is a relatively new standard. Consequently, the amount of public training data available for SysML v2 is significantly lower compared to established programming languages, such as Python. This scarcity often leads to LLMs failing to produce syntactically correct code despite understanding the semantic intent.

Iterative product development typically relies on heterogeneous legacy knowledge describing the current system, including documents, requirements specifications, and prior model versions. In this work, the development of an electric bike represents such a setting in which existing bicycle documentation is reused and adapted to develop a new product. Previously discussed research proposed workflows and tool interactions to integrate this knowledge into MBSE activities and to capture additional expert knowledge in natural language. When LLMs are provided with both documents and expert input, they can support the elicitation and transformation of engineering information into candidate model structures. However, SysML v2 text notation imposes strict syntactic constraints, and unconstrained generation frequently violates them, limiting direct tool import and visualization. As a result, a plain large language model is often insufficient for standalone modeling assistance when automation and reliable tool compatibility are required.

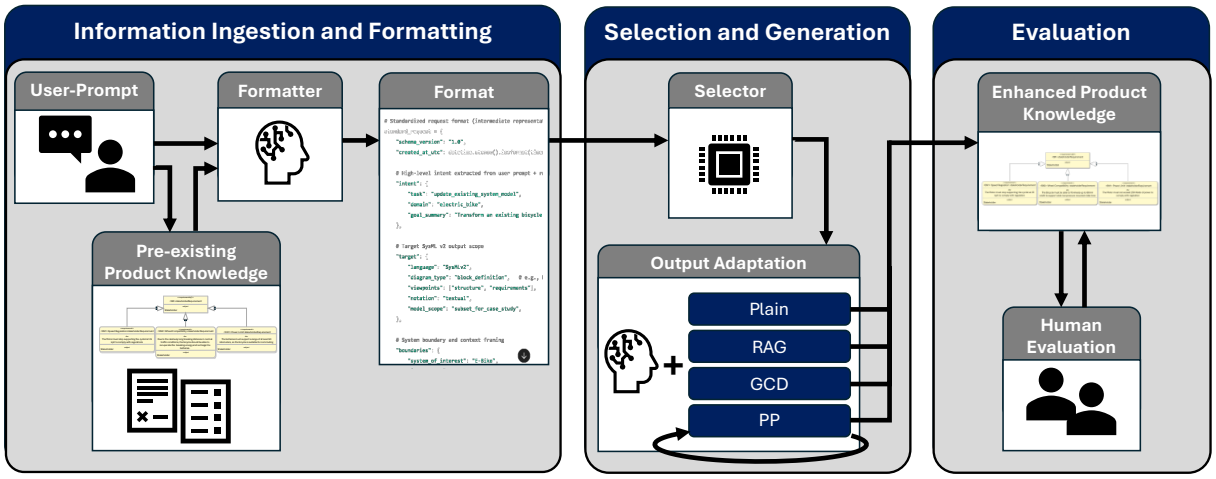

Figure 1: Process for ensuring syntactical validity of SysML v2 models

The approach is implemented as an orchestration pipeline around a local LLM to generate SysML v2 textual notation. Figure 1 summarizes the workflow for iterative product development, where user input and legacy engineering assets are transformed into an updated system model. The primary requirement is that the generated SysML v2 text is syntactically valid and can be parsed and visualized in engineering

tools. To improve reliability, the pipeline first standardizes the combined prompt and legacy knowledge into a structured intermediate representation and then applies an algorithmic selector to choose an output adaptation strategy. The considered adaptations are RAG, GCD, and PP, with a plain LLM serving as a baseline and fallback.

3.1 Information Ingestion and Formatting

The first phase of the approach implements a data preprocessing step that unifies heterogeneous inputs into a uniform, machine-readable representation. It combines the natural language user prompt with accessible knowledge from the development of previous product generations, such as documents and model fragments. It converts both sources into a standardized intermediate format. This step is required because the subsequent selector depends on comparable input features across use cases. At the same time, raw user prompts and retrieved text passages can differ strongly in structure, terminology, and level of detail. The transformation is performed by an LLM, shown as the Formatter in Figure 1, which is configured to produce structured output in a fixed schema rather than free text. Therefore, this phase focuses on acquisition, restructuring, and presentation of information in a consistent schema that can be reused across projects and modeling tasks, to reduce surface variability and to expose modeling intent.

The text is checked for system boundaries, named subsystems and components, stated interfaces, and the intended viewpoints and diagram types. In parallel, the attached documents are processed to extract relevant passages that can serve as additional knowledge required in the current task. Retrieved content is condensed into short statements that preserve engineering meaning while reducing token load and redundancy. The formatter reduces the content into a single request object. The intermediate format is a structured representation with defined fields, as “Format” in Figure 1. This structure follows the structured output principle and encodes, at minimum, boundaries describing the explicit system of interest plus in scope and out of scope statements, viewpoints and requested diagram types, extracted requirements in the style of bullet points, candidate components or subsystems named in the prompt, as well as interfaces or flows mentioned in the prompt that can be mapped to SysML v2 elements in later steps. Without this phase, the selector would operate on inconsistent input, and the downstream generation prompt would be less reproducible and more prone to missing constraints or injecting irrelevant context.

3.2 Selection and Generation

After normalizing the user prompt and structuring the request as a dictionary containing bullet points of requirements, information, boundaries, and keywords, the request is algorithmically evaluated to determine which model generation method is most promising. This is done by using a weighted summarization of the amount of information and keywords given. Therefore, system boundaries are lowest in weight, with the required viewpoints being second to last in weight. The more important parameters for this evaluation are the number of keywords extracted from the user prompt, and the number of keywords found in additional code snippets, attached files, and retrieved documents. This estimation then results in a mode selection where the decision between a keyword-based RAG, GCD or PP is made, or whether no additional method is used and the plain LLM should be utilized for code generation.

RAG injects curated SysML v2 notation fragments into the prompt context before inference, shifting generation from unconstrained synthesis toward reuse of known valid patterns. Retrieval is implemented as a deterministic keyword stem match over the normalized user prompt. For each detected keyword, the corresponding curated SysML v2 fragments are appended as contextual hints that illustrate the intended construct. To ensure a minimal baseline of usable notation, a compact SysML v2 reference

package is included when RAG is active, so that generation remains guided even if no keyword matches occur.

GCD restricts token selection during inference to continuations that satisfy a predefined context-free grammar, which prevents syntactic violations within the covered language fragment. In this work, the constraint is implemented using Backus-Naur Form grammar rules, as supported by llama.cpp, in the local inference setup. This improves the likelihood that the generated SysML v2 text is syntactically correct. Still, it can reduce content coverage because the model can only express constructs admitted by the grammar, which does not ensure that names, allocations, and relations match the intended architecture. The applied SysML v2 grammar is therefore scoped to the constructions required by the instantiations.

PP converts near-valid outputs into syntactically correct SysML v2 notation via a validation-and-repair loop. Typical failures include minor formatting violations, missing semicolons, inconsistent keyword choices, or the insertion of extraneous lines that are plausible in other modeling languages but invalid in SysML v2 textual notation. PP targets these high-frequency, low-complexity defects after generation, with the objective of converting near-valid outputs into strictly valid artifacts without requiring repeated full regeneration, with an optional fallback LLM correction if deterministic repairs fail. This method cannot correct significant deviations from the target SysML v2 notation and cannot improve the content covered by the LLM.

3.3 Evaluation

The final phase applies the four-eyes principle by assigning the final decision to the systems engineer responsible. The purpose is to ensure that the generated SysML v2 textual notation is both formally correct and practically usable within engineering tools, and that the content matches the intended system modification rather than merely syntactically correct code. This positions LLMs as supporting tools, with structured output and explicit roles guiding generation, while acceptance and quality assurance remain human responsibilities.

The evaluation starts with a formal verification of tool compatibility. The engineer imports the generated SysML v2 text into the tool of his choice, such as the SysML v2 pilot implementation, and checks that it can be visualized without errors. This step validates syntactical correctness and guards against defects that may not be apparent in plain text, such as missing delimiters, invalid keywords, or incomplete constructs. If the import fails, the engineer determines whether the issue is a minor formatting defect that can be corrected locally or a structural defect that requires rerunning the pipeline with a different output adaptation. After syntactical correctness is established, the engineer evaluates the semantic and structural value of the output. The focus here is on whether the model contains the expected elements and relationships implied by the prompt and the retrieved legacy evidence, and whether the chosen viewpoint and diagram type were realized in the intended scope. Typical checks include whether system boundaries were respected and whether key components and interfaces are present with consistent naming. The engineer also checks for overgeneration, where the model introduces elements that conflict with the existing system model and associated artifacts.

If the result is acceptable, the engineer integrates the generated systems model into the ongoing development work and continues modeling in the authoring tool. If the result is incomplete but structurally sound, the engineer adds the missing information manually and treats the output as an initial model fragment. If the result is invalid or systematically misaligned with the intent, the engineer updates the input knowledge and restarts the pipeline so that the ingestion, selection, and generation steps can incorporate the new information.

4 Evaluation of Output Adaptation Techniques

This chapter validates the approach from Chapter 3 by instantiating it in an electric bike development scenario using mistral-small 3.1 24b and by assessing whether it reliably produces SysML v2 textual notation that is syntactically valid and therefore usable in engineering tools. The validation target, therefore, includes both the decision logic that maps prompt and knowledge availability to a generation method and the resulting generated SysML v2 code, which is evaluated for formal correctness and usability. Nevertheless, each method needs to be validated individually to ensure the required quality. The development setting is iterative product generation, where an existing system model and associated artifacts from a previous bike development serve as prior knowledge, while the user prompt expresses the intent to develop an e-bike. The approach assumes that information availability varies widely between development situations, ranging from short, underspecified user prompts to detailed prompts with concrete specifications. To compare results across the different approaches and validate the importance of method selection, the validation uses two prompt granularities that represent extremes of information availability.

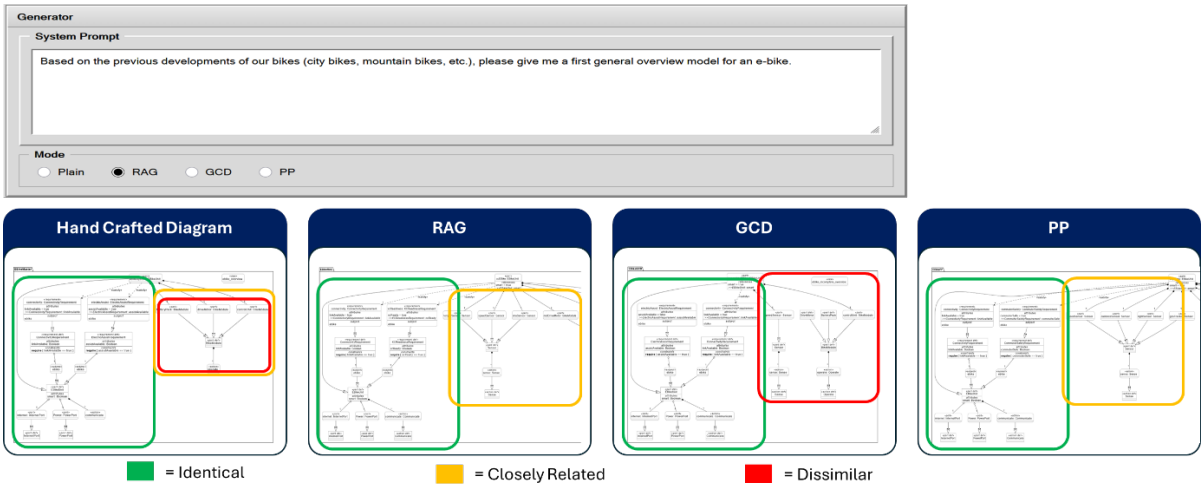

Figure 2: Comparison of outputs to the manually generated master model

These granularities are an abstract prompt and a detailed prompt, while keeping inference settings and the reference system intent constant. Because LLMs output is stochastic, each configuration is prompted repeatedly with slight variations. Model quality is evaluated along the dimensions content coverage against a manually defined reference architecture and syntactical correctness, which is assessed by attempting to render the generated SysML v2 artifact in the SysML v2 pilot implementation. Both the content and syntactic correctness are scored based on the presence of expected elements. Figure 2 compares the SysML v2 diagrams produced with RAG, GCD, and PP to a hand-crafted master model for the same prompt, illustrating how each method preserves or alters structural patterns. Highlighted regions denote identical, closely related, and dissimilar substructures in the rendered views.

The idea for the selector is that the failure mode changes with information availability. When prompts are extensive, and legacy knowledge is high, the main risk is the loss of domain-specific structure and SysML notation patterns, not the absence of concepts. In such cases, the approach prioritizes keyword-based RAG, which is deterministic and defined by the user prompt as it is scanned for predefined keywords. The validation results shown in Figure 3 support this method, especially for long and extensive user prompts that contain numerous keywords. The reported percentages are obtained by comparing each generated model against a manually created reference architecture for the electric bike scenario. This reference is used as a checklist of expected SysML v2 elements and relations derived from the hand-crafted reference model, and each generated artifact is inspected for the presence of these

expected items. The content score corresponds to the proportion of expected items that are present in the generated SysML v2 text, while the parsing score corresponds to the proportion of expected items that remain usable after parsing and rendering the artifact in the SysML v2 pilot implementation. With a detailed prompt, RAG achieves the highest mean content score at 97 percent and the highest mean representability score at 94 percent, effectively replicating the reference architecture for the electric bike case.

	Run		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	Average
Content Score	Abstract	Plain	55	80	25	35	35	50	40	0	10	5	35	0	20	10	0	0	75	27,9
		RAG	90	0	0	0	0	0	0	0	0	0	0	0	100	95	80	80	0	26,2
		GCD	100	80	55	40	80	40	50	0	10	0	60	15	70	65	40	55	80	49,4
		PP	60	90	10	30	40	30	30	5	5	15	45	0	10	5	0	0	50	25,0
	Extended	Plain	70	90	15	65	100	100	80	70	100	90	100	90	65	50	10	20	100	71,5
		RAG	70	100	100	100	100	100	85	100	100	100	100	100	100	100	100	100	100	97,4
		GCD	90	90	70	80	85	90	80	35	45	100	90	100	100	100	70	60	100	81,5
		PP	45	100	15	85	100	70	60	95	85	100	90	85	55	90	0	0	95	68,8
Parsing Score	Abstract	Plain	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	1,8
		RAG	80	0	0	10	0	0	5	0	5	0	0	0	90	70	0	10	0	15,9
		GCD	80	85	20	30	65	35	5	15	15	0	60	20	50	80	40	65	60	42,6
		PP	65	70	15	20	10	5	0	0	5	0	5	0	0	10	0	0	5	12,4
	Extended	Plain	0	35	0	15	30	10	5	15	5	10	10	10	0	45	0	0	20	12,4
		RAG	60	100	95	90	100	100	100	100	100	100	100	100	100	100	75	70	100	93,5
		GCD	60	100	60	90	80	100	75	20	35	100	90	90	100	95	35	25	85	72,9
		PP	50	85	20	0	70	40	10	50	75	75	65	65	25	10	0	0	60	41,2

Figure 3: Relative content score of 17 prompt runs, with percentage values

Under the abstract prompt, RAG drops to a mean content coverage of 26 percent and exhibits high variance, consistent with a keyword-triggered retrieval mechanism that requires explicit cues for stable activation and effective adaptation.

When prompt completeness and accessible product knowledge are low, the failure mode shifts from semantic incompleteness to reduced formal correctness, because unconstrained decoding frequently emits textual notation that cannot be visualized as SysML v2. In such cases, the approach prioritizes GCD, restricting token selection during decoding and thereby preventing syntax errors within the covered subset. The electric bike instantiation supports this choice. Under the abstract prompt, GCD increases the mean syntactic correctness and the visualization score to 43 percent compared to the plain LLM baseline, while under the detailed prompt it reaches a 73 percent mean parsing score, second only to RAG. The same evidence shows that grammar constraints do not inject missing engineering content, and in some runs, the restriction reduces content coverage by limiting the model's expressive freedom to what the grammar admits.

For prompts of intermediate length and information content, and for cases where neither retrieval effectiveness nor grammar coverage can be confidently assumed, the approach prioritizes rule-based PP as a repair mechanism. The electric bike instantiation shows that PP yields intermediate improvements that are strongest when the base output is already close to valid. Under the detailed prompt, PP raises the mean parsing score to 41 percent while achieving 69 percent content coverage, whereas under the abstract prompt, it remains limited to 12 percent representability and 25 percent content coverage, indicating that PP cannot compensate for significant deviations from SysML v2 notation and cannot add missing architecture information. The significant advantage of PP is that it does not modify the output generation at inference time. Therefore, PP is theoretically compatible with all other approaches investigated and should be usable in tandem. The plain LLM condition is retained as a baseline and as a fallback should fine-tuned LLMs for SysML v2 become available, but the validation demonstrates that it is not suitable as a default strategy for SysML v2 model generation in the considered setting. With a

detailed prompt, the plain LLM achieves approximately 72 percent content coverage but only 12 percent representability, and with an abstract prompt its representability drops to 2 percent. Overall, the electric bike instantiation supports the selector assumptions introduced in Chapter 3. The implications for method selection are consolidated in Chapter 5.

5 Conclusion and Outlook

This work proposes a selector-based approach to automated SysML v2 generation, using the techniques of RAG, GCD, and PP. The results show that prompt completeness strongly influences both architectural fidelity and syntactical correctness, with detailed prompts resulting in significantly higher performance across all approaches. GCD improved the syntax under both prompt versions, indicating its usability as a safeguard that increases the proportion of outputs directly usable in engineering tools, but limits LLM token generation and can result in content coverage failures. RAG achieved the highest content and syntax scores when using detailed prompts, but shows sensitivity to abstract prompts. PP delivered moderate gains by converting almost correct outputs into syntactically correct SysML v2 artifacts, but did not substantially increase architectural completeness when the model output strongly deviated from the reference.

These findings highlight the need to dynamically adapt LLM output in engineering workflows to address both semantic correctness and content improvements. RAG support enhances the reuse of domain knowledge when the user prompt is sufficiently detailed. PP functions as a fallback that allows repair stages to increase syntactic correctness without modifying the content of the output. Plain LLMs alone fail to meet the dual requirements of representability and fidelity in most underspecified scenarios, but should not be generally dismissed, as future LLMs will most likely show improved performance, once a large enough dataset of SysML v2 syntax is publicly available for the LLMs to be trained on

The results and the fact that PP can in theory be used in tandem with other approaches indicate that a hybrid approach of RAG and PP most likely delivers the best performance when sufficient knowledge from previous developments is available, and needs further research. Especially considering the iterative nature of smart product development, where previous knowledge is typically available, this should be the most used method for engineering assistance. Further research should also be done using graph-based RAG approaches as well as more complex PP tools that include multiple LLMs working on different specialized aspects of a system model, such as requirements, use-case diagrams, or completeness and mapping between various layers of a SysML v2 model.

References

[1] T. Tomiyama, E. Lutters, R. Stark and M. Abramovici, “Development capabilities for smart products,” CIRP Annals, vol. 68, no. 2, pp. 727-750, 2019. DOI: https://doi.org/10.1016/j.cirp.2019.05.010.

[2] A. Albers, N. Bursac and E. Wintergerst, “Produktentgenerationsentwicklung - Bedeutung und Herausforderungen aus einer entwicklungsmethodischen Perspektive,” in *Stuttgarter Symposium für Produktentwicklung 2015*.

[3] M. Kretzschmar, M. P. Dammann, S. Schwoch, E. Berger, B. Saske and K. Paetzold-Byhain, “Key Concepts, Potentials and Obstacles for the Implementation of Large Language Models in Product Development,” in *Entwerfen Entwickeln Erleben 2024 : Menschen, Technik und Methoden in Produktentwicklung und Design*: Technische Universität Dresden, 2024, pp. 381–393.

[4] D. Stenholm and D. Bergsjö, “Barriers to reuse of codified engineering knowledge in product development - a literature review,” International Journal of Product Lifecycle Management, vol. 12, no. 4, p. 307, 2020. DOI: https://doi.org/10.1504/IJPLM.2020.112773.

[5] L. Jokinen and S.-P. Leino, “Hidden product knowledge: problems and potential solutions,” Procedia Manufacturing, vol. 38, pp. 735-744, 2019. DOI: https://doi.org/10.1016/j.promfg.2020.01.099.

[6] W. J. Verhagen, P. Bermell-Garcia, R. E. van Dijk and R. Curran, “A critical review of Knowledge-Based Engineering: An identification of research challenges,” Advanced Engineering Informatics, vol. 26, no. 1, pp. 5-15, 2012. DOI: https://doi.org/10.1016/j.aei.2011.06.004.

[7] S. Schleifer, A. Lungu, B. Kruse, S. van Putten, S. Goetz and S. Wartzack, “Minimal Data, Maximal Impact: Language Model-based Pipelines for the Automatic Generation of Use Case Diagrams from Requirements,” in *DS 133: Proceedings*, pp. 240–249.

[8] L. T. Blessing and A. Chakrabarti, DRM, a Design Research MethodologyLondon: Springer London, 2009.

[9] M. Abramovici, J. C. Göbel and H. B. Dang, “Semantic data management for the development and continuous reconfiguration of smart products and systems,” CIRP Annals, vol. 65, no. 1, pp. 185-188, 2016. DOI: https://doi.org/10.1016/j.cirp.2016.04.051.

[10] C. M. Gardiner, “Knowledge sharing success and resistance in an engineering department: A case study,” Journal of Management & Organization, vol. 22, no. 2, pp. 254-271, 2016. DOI: https://doi.org/10.1017/jmo.2015.30.

[11] P. Krenz, S. Basmer, S. Buxbaum-Conradi, T. Redlich and J.-P. Wulfsberg, “Knowledge Management in Value Creation Networks: Establishing a New Business Model through the Role of a Knowledge-Intermediary,” Procedia CIRP, vol. 16, pp. 38-43, 2014. DOI: https://doi.org/10.1016/j.procir.2014.01.006.

[12] D. Birch, H. Liang, P. H. J. Kelly, G. Mullineux, T. Field, J. Ko and A. Simondetti, “Multidisciplinary Engineering Models: Methodology and Case Study in Spreadsheet Analytics,” 2014. DOI: https://doi.org/10.48550/arXiv.1401.4582.

[13] M. Chami and J.-M. Bruel, eds., A Survey on MBSE Adoption Challenges (2018).

[14] A. Akundi, W. Ankobiah, O. Mondragon and S. Luna, “Perceptions and the extent of Model-Based Systems Engineering (MBSE) use – An industry survey,” in *2022 IEEE International Systems Conference (SysCon)*: IEEE, 2022, pp. 1–7.

[15] K. Henderson and A. Salado, “The Effects of Organizational Structure on MBSE Adoption in Industry: Insights from Practitioners,” Engineering Management Journal, vol. 36, no. 1, pp. 117-143, 2024. DOI: https://doi.org/10.1080/10429247.2023.2210494.

[16] A. M. Madni and M. Sievers, “Model-Based Systems Engineering: Motivation, Current Status, and Needed Advances,” in *Disciplinary Convergence in Systems Engineering Research*, A. M. Madni, B. Boehm, R. G. Ghanem, D. Erwin and M. J. Wheaton, eds.: Cham: Springer International Publishing, 2018, pp. 311–325.

[17] A. Mahboob and S. Husung, “A Modelling Method for Describing and Facilitating the Reuse of SysML Models during Design Process,” Proceedings of the Design Society, vol. 2, pp. 1925-1934, 2022. DOI: https://doi.org/10.1017/pds.2022.195.

[18] R. Mendieta, J. L. de La Vara, J. Llorens and J. M. Álvarez-Rodríguez, “Towards Effective SysML Model Reuse,” in *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development*: SCITEPRESS - Science and Technology Publications, 2017, pp. 536–541.

[19] M. S. d. Biase, S. Marrone and A. Palladino, Towards Automatic Model Completion: from Requirements to SysML State Machines, 2022.

[20] M. Ariffud, “LLM statistics 2025: Comprehensive insights into market trends and integration,” [Online]. Available: https://www.hostinger.com/tutorials/llm-statistics. [Accessed 6 November 2025].

[21] A. Akundi, J. Ontiveros and S. Luna, “Text-to-Model Transformation: Natural Language-Based Model Generation Framework,” Systems, vol. 12, no. 9, p. 369, 2024. DOI: https://doi.org/10.3390/systems12090369.

[22] H. Kourani, A. Berti, D. Schuster and W. M. P. van der Aalst, “Process Modeling with Large Language Models,” in *Enterprise, Business-Process and Information Systems Modeling*, Vol. 511, H. van der Aa, D. Bork, R. Schmidt and A. Sturm, eds.: Cham: Springer Nature Switzerland, 2024, pp. 229–244.

[23] J. K. DeHart, “Leveraging Large Language Models for Direct Interaction with SysML v2,” INCOSE International Symposium, vol. 34, no. 1, pp. 2168-2185, 2024. DOI: https://doi.org/10.1002/iis2.13262.

[24] D. Mollahassani, M. Becker, T. Eickhoff, J. Pickel, S. Goetz, S. Wartzack and J. C. Göbel, “Framework for circular AI-driven model-based systems engineering,” Proceedings of the Design Society, vol. 5, pp. 1883-1892, 2025. DOI: https://doi.org/10.1017/pds.2025.10202.

[25] B. Johns, K. Carroll, C. Medina, R. Lewark and J. Walliser, “AI Systems Modeling Enhancer (AI-SME): Initial Investigations into a ChatGPT-enabled MBSE Modeling Assistant,” INCOSE International Symposium, vol. 34, no. 1, pp. 1149-1168, 2024. DOI: https://doi.org/10.1002/iis2.13201.