2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)

[NKR+25] L. Netz, F. Kampe, J. Reimer, B. Rumpe:
Unintended Changes: How LLMs Corrupt and Correct Textual Models.
In: 2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C),
pp. 636-645, DOI 10.1109/MODELS-C68889.2025.00087, IEEE, Oct. 2025.

Unintended Changes: How LLMs Corrupt and Correct Textual Models

Lukas Netz
Software Engineering
RWTH Aachen University
Aachen, Germany
netz@se-rwth.de

Finn Kampe
RWTH Aachen University
Aachen, Germany
finn.kampe@rwth-aachen.de

Jan Reimer
Software Engineering
RWTH Aachen University
Aachen, Germany
jan.reimer@rwth-aachen.de

Bernhard Rumpe
Software Engineering
RWTH Aachen University
Aachen, Germany
rumpe@se-rwth.de

***Abstract*—This work evaluates to what degree AI-based model-driven software development approaches may unintentionally alter parts of a model, potentially affecting its semantics, particularly when the model is defined in a domain-specific language unfamiliar to the LLM, and discusses to what degree the perplexity metric can be used to indicate susceptible modeling languages. Although much work has focused on evaluating the correctness of AI-based model creation, it remains important to assess whether and to what extent LLMs introduce unintended modifications in parts of the input model that they are merely expected to reproduce. Previous research has primarily focused on the success of programming and modeling tasks using AI, such as achieving syntactic and semantic correctness. In contrast, this work investigates the side effects of AI usage, specifically the introduction of unintended modifications during iterative modeling processes. We evaluate the likelihood of error introduction in an AI-based modeling approach for uncommon DSLs and use the perplexity-metric to identify error prone elements in the model. We use our own modeling language for class diagrams CD4A, an LLM that is not trained on this DSL and prompt it with simple modeling tasks. Next, we measure additional changes to the model that are not related to the given modeling task. We show that AI-driven Modeling approaches need to also focus on side effects of iterative LLM usage with unfamiliar modeling languages, and demonstrate how these risks can be identified.**

***Index Terms*—LLM, MDE, AI4SE, DSL**

I. Introduction

Large Language Models (LLMs) have gained recognition for their capability to synthesize and edit source code [1]–[4]. Additionally, LLMs have been successfully applied to various modeling tasks [5]–[8] as well as low resource programming tasks [9]. Their effectiveness in these tasks primarily comes from extensive training data and the similarity between the targeted programming or modeling languages and the examples included within their training corpus [10], [11]. However, LLMs are also known for their unreliability due to hallucinations [12]. In addition they can exhibit biases and conformity issues in these programming and modeling tasks [13]. When the correct response differs even slightly from the dominant patterns in their training data, LLMs tend to provide incorrect responses by conforming to the prevalent data from their training [14]. This bias creates specific challenges [15], especially when the targeted modeling language is either absent or minimally represented in the training data, yet shares concepts with a more familiar language included in the training set. This property can apply to many domain-specific languages (DSLs) that are not public or are only available with a small data sets in the public domain. In particular, modeling languages that are developed within a company or as part of scientific work (low-resource languages [9]). In this work, we evaluate how LLMs display bias toward frequently encountered training data in model generation tasks. Although it is already well-known that LLMs exhibit various forms of bias, we specifically investigate how these known biases affect modeling tasks. We focus on instances where the LLM incorrectly introduces modifications based on familiar concepts into sections of code from a less common DSL, even when those sections were not directly targeted by the provided prompt. We pose the following as our first research question:

> **RQ 1:** In LLM-based modeling approaches, to what extent do unintended changes occur in parts of the model that are unrelated to the prompt or the modeler's intended customization?

Perplexity can be used as a metric to assess an LLM's ability to comprehend a given input language [16]. Assuming that an LLM is more likely to introduce errors when it encounters unfamiliar input it was not trained on, perplexity may correlate with the frequency of such errors. Thus we define the second research question as follows:

> **RQ 2:** Can the perplexity metric serve as an indicator of a modeling language's susceptibility to unintended adaptations in LLM-based modeling approaches?

979-8-3315-7990-6/25/$31.00 ©2025 IEEE
DOI 10.1109/MODELS-C68889.2025.00087

II. Related Work

Generative AI, particularly in the area of Large Language Models, is a rapidly evolving research field. New findings are published within months or weeks [17]–[20]. While LLMs are extensively utilized for code generation tasks, there is currently limited research exploring their application and evaluation specifically for generating domain-specific language source code or the usage of LLMs in modeling tasks.

In their work [21] Kammakomati et al. focus on assessing the performance of LLMs in generating code for domain-specific languages. It emphasizes the unique challenges posed by DSLs, which often have specific syntax and semantic constraints that differ from general-purpose programming languages. The evaluation framework *ConCodeEval* proposed in the paper aims to measure how well LLMs adhere to these constraints while generating code, which is crucial for ensuring the correctness and functionality of the generated outputs.

In their work [22], Bassamzadeh and Methani investigate the challenges faced by LLMs in generating code for Domain-Specific Languages, particularly when dealing with custom function names. It highlights that LLMs often produce higher rates of hallucinations and syntax errors in DSLs due to the complexity and frequent updates of function names. The study presents optimizations for using Retrieval Augmented Generation in DSL code generation and includes an ablation study comparing RAG with fine-tuning methods. A dataset representing automation tasks across approximately 700 APIs was created for training and testing. The findings indicate that while fine-tuning a Codex model yielded the best results in code similarity metrics, RAG optimizations achieved comparable performance. However, both methods still struggled with syntax errors, with RAG performing slightly better. The paper concludes that an optimized RAG model can match the quality of fine-tuned models while offering advantages for handling new, unseen APIs.

The work by Xiaodong et al. [23] investigates the performance of large language models like ChatGPT in generating code tailored to specific domains, such as web development or game programming. The authors highlight that while LLMs have shown impressive capabilities in general code generation, they often struggle with domain-specific tasks due to limited proficiency in utilizing specialized libraries and APIs. The study explores various strategies to enhance LLM performance in this area, including: External Knowledge Inquirer: Integrating external API knowledge into the generation process. Chain-of-Thought Prompting: Encouraging the model to reason through the code generation step-by-step. Chain-of-Thought Fine-Tuning: Adjusting the model's training to better handle domain-specific tasks. The results indicate that these strategies, collectively referred to as DomCoder, significantly improve the effectiveness of domain-specific code generation under certain conditions, suggesting that targeted approaches can enhance LLM capabilities in specialized contexts.

III. Fundamentals

A. Large Language Models

A Large Language Model (cf. Figure 1) is a deep learning-based artificial neural network trained to generate coherent and contextually relevant text by predicting subsequent tokens given an input sequence.

LLMs are commonly pretrained on extensive datasets and subsequently refined through methods such as *Fine-Tuning*, where the entire model or select layers are further trained on task-specific data. *Low-Rank Adaptation (LoRa)* [24] represents an efficient fine-tuning alternative, modifying only low-rank matrices in the model to adapt rapidly to new tasks while maintaining computational efficiency.

Additionally, LLMs leverage *In-context Learning*, including *Few-Shot Learning (FSL)* [25], where models perform tasks by generalizing from limited examples provided at inference time. Advanced prompting techniques, such as *Chain-of-Thought (CoT)*, guide models through step-by-step reasoning, enhancing accuracy on complex tasks. *Prompt Tuning* further adjusts input prompts without altering model weights, allowing efficient adaptation. Another important technique is *Retrieval-Augmented Generation (RAG)* [26], which integrates external knowledge retrieved from a *vector database* (cf. Figure 1) containing embeddings of documents. This approach enables LLMs to dynamically incorporate context-specific information, improving response accuracy and relevance.

Specialized systems like GitHub Copilot[1] or Cursor[2] utilize LLMs specifically trained and fine-tuned on code repositories, focusing on software development tasks, thereby enhancing code completion accuracy and contextual relevance compared to generic LLMs. Often especially in code or model generation a post processing step is added to verify syntactic correctness of the approach [27].

B. Domain Specific Languages

MontiCore [28] is a language workbench designed for the efficient development and maintenance of DSLs. It was used to develop several DSLs covering UML and other modeling languages[3]. One of those is **CD4A**[4] (Class Diagrams for Analysis) [29]–[31]: A domain-specific language designed to support the conceptual analysis phase and generative approaches in software engineering. CD4A provides tailored features specifically for creating and managing class diagrams during early software analysis. CD4A employs a Java-like syntax, making it intuitive for users familiar with Java and easing the transition from conceptual analysis to implementation.

CD4A allows software architects and developers to clearly express the structural aspects of a system by defining entities, their attributes, associations, multiplicities, and inheritance hierarchies [31].

[1] https://github.com/features/copilot
[2] https://www.cursor.com
[3] https://monticore.github.io/monticore/docs/DevelopedLanguages/
[4] https://github.com/MontiCore/cd4analysis

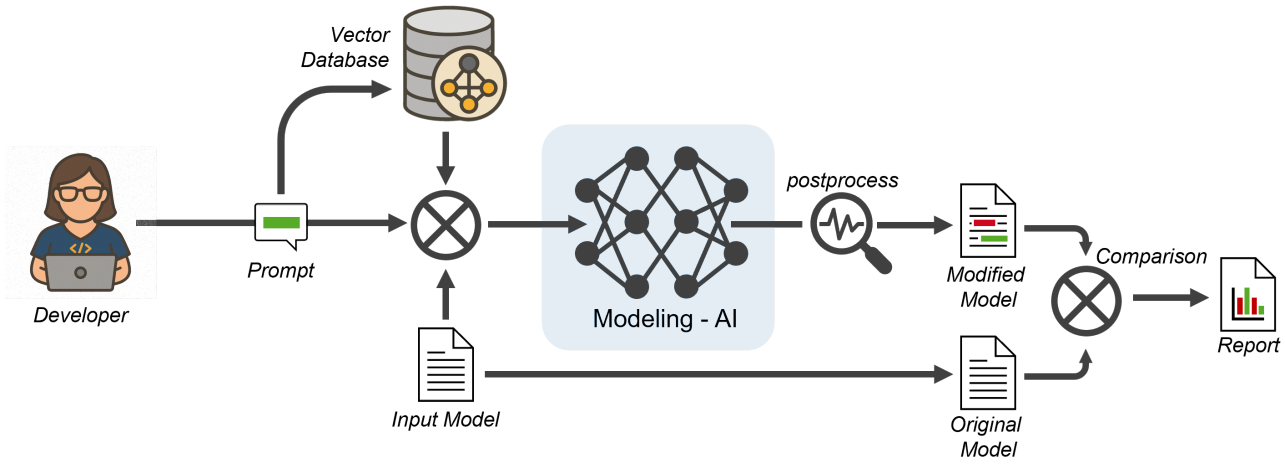

Fig. 1. Usage of an LLM to adapt a textual model. Input Artifact (Input Model) is provided together with a prompt, that define the intended changes. The input is embedded and further prompt optimizations are performed (FSL, RAG, etc.) this input is provided to the LLM. The output is post processed (e.g. parsed) and provided to the user.

```
1  classdiagram MyClassDiagram {
2    class Person  {
3      String name;
4      Date dateOfBirth;
5      /Integer age;        //derived parameter
6    }
7    class Home {
8      String Address;
9    }
10   association Person <->  Home;
11 }
```

Listing 1. Class Diagram in CD4A syntax, containing a derived parameter in line 5.

Listing 1 shows a simple example of a Class Diagram defined in the Java-like CD4A DSL. The diagram defines two classes: `Person` and `Home`, as well as abidirectional association between both. Note in line 5 the deviation to known syntax from Java: a *derived attribute* [32] `/Integer age;`. Derived attributes can only be associated with a type and a name, and they cannot be modified directly, which implies that they do not have corresponding mutable variables in the generated code. As such, when implementing these derived attributes in a programming language like Java, they are typically represented as accessors that provide read-only access to the computed value. The derived attribute in Listing 1 provides the `age` of `Person`. The age itself is not defined as a member variable of the Person, but is rather computed based on other parameters, such as `dateOfBirth`. This specific method of computation is not part of the CD4A model.

PlantUML [33] is a Modeling Language that is well known to common LLMs [34] (such as GPT-3.5, GPT-4o or deepSeek). It allows users to create UML diagrams using a simple and human-readable text description. In the case of class diagrams, PlantUML supports the definition of classes with attributes and methods, as well as relationships such as inheritance, associations, and dependencies. The PlantUML class diagram defined in Listing 2 defines the same use case as Listing 1: Two classes, `Person` and `Home`, each with their respective attributes, and models a bidirectional association between them.

```
1  @startuml
2      class Person {
3        String name
4        Date dateOfBirth
5        Integer age
6      }
7      class Home {
8        String Address
9      }
10     Person -- Home
11 @enduml
```

Listing 2. Class Diagram in PlantUML syntax.

IV. Methodology

In this section, we discuss our approach for measuring the extent to which LLMs either correct or corrupt textual models. As an example, we take a closer look at the derived attribute notation in CD4A and to what extent it is preserved throughout an AI-based modeling process.

A. Perplexity

A DSL has a well-defined syntax, often described using a *context-free grammar (CFG)*, which defines the valid structure of expressions in that language. In contrast, the **syntactic fluency** [35] of a LLM refers to its ability to generate text that conforms to grammatical rules, whether in natural language or a DSL. DSLs are highly structured and follow strict formal rules, typically defined using grammars such as BNF (Backus-Naur Form) or PEG (Parsing Expression Grammar). They allow little tolerance for syntactic variation; a small syntax

error (e.g., a missing semicolon) can make an entire statement invalid. Natural languages, in contrast, have a tendency to be more probabilistic and flexible, allowing for more ways to express the same idea. A sentence with spelling errors will still be interpretable by an LLM, however, a parser will not parse code with syntax errors. A Large Language Model trained on a DSL should ideally recognize syntactic violations.

To measure how well an LLM understands the syntax of a DSL, **perplexity (PPL)** is a useful metric. PPL is a common metric to identify if a text is created by an LLM or not [36], [37]. It quantifies how well the model predicts the next token in a sequence and is defined as:

$$PPL = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p(w_i|w_1,...,w_{i-1})\right)$$

where N is the number of tokens in the sequence, and $p(w_i|w_1,...,w_{i-1})$ is the probability assigned by the model to token w_i.

In our implementation, we calculate these probabilities by extracting the raw logits $\mathbf{l}_i \in \mathbb{R}^V$ (where V is vocabulary size) at each position and applying softmax normalization:

$$p(v|t_1,...,t_i) = \frac{\exp(l_{i,v} - \max(\mathbf{l}_i))}{\sum_{j=1}^{V}\exp(l_{i,j} - \max(\mathbf{l}_i))}$$

The subtraction of $\max(\mathbf{l}_i)$ ensures numerical stability. We then accumulate the negative log-likelihood $-\log(p_i)$ for each predicted token to compute the final perplexity. For longer sequences, we also calculate chunk-level perplexity over windows of 512 tokens to identify regions of varying model confidence.

Lower perplexity indicates that the model assigns high probability to syntactically correct sequences, meaning it has learned the patterns well, whereas higher perplexity suggests that the model is uncertain or struggles to predict correct structures. Perplexity is primarily a local feature: it measures token-by-token prediction accuracy but does not directly indicate if an LLM understands long-range dependencies [16].

Similarly, for a DSL, incorrectly structured definitions should contribute to increased perplexity, while syntactically valid definitions should lower perplexity. If an LLM is successfully trained on a DSL, we expect a sharp contrast in perplexity values between well-formed and malformed expressions.

Perplexity can be used to identify tokens in an input artifact that an LLM is likely to miss or misinterpret due to insufficient training on the given syntax. The underlying assumption is that an LLM struggles to reproduce unfamiliar syntax (high perplexity) more than familiar syntax (low perplexity). As a result, it may replace the unfamiliar syntax with a well-known structure from a similar language it has been extensively trained on. Additionally, we compute a local normalized probability for each token—the ratio between the actual token's probability and the maximum alternative probability—to assess the model's relative confidence in its predictions.

While syntactically correct but unusual programs can have high perplexity, this complexity is not relevant here. We use perplexity to identify tokens unfamiliar to the LLM from its training data. High perplexity indicates constructs the LLM is likely to misinterpret or replace with familiar patterns, which is our primary concern when processing unknown DSLs.

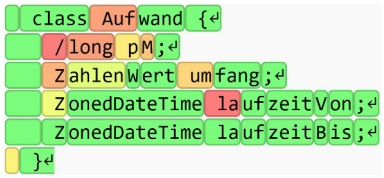

Fig. 2. Perplexity measure on CD4A Syntax with the Phi 4 Model using a GPT2 tokenizer [39] (red = less likely token). The LLM is not only perplexed by 'novel' syntax (`'/'`), but also by irregular naming. E.g. it expects Date-attribute-names to start with 'date'.

Figure 2 shows the perplexity heatmap for an excerpt of a CD4A class diagram (The full model can be found at [38]). The heatmap shows, that the LLM did not predict the class name to start with "`Auf`" and clearly did not expect the first attribute to be defined with "`/`". In addition it also did not expect the `ZonedDateTime` attribute to be named starting with "`la`", in this case the predicted token is "`datum`" as the LLM expects the Date attribute to be named starting with a date-like name. In addition we notice that the '`Z`' tokens is getting more likely with each line of code. This beautifully visualizes the in-context-learning methodology: The LLM adapts to patterns, and starts to predict repeated tokens.

We can use this metric to identify tokens that perplex the LLM and thus are likely to be badly reproduced by it.

B. Input Output Comparison

In order to assess whether, and to what extent, an LLM-based approach introduces unintended changes into the modeling process, e.g. by badly reproducing tokens, we define the following experimental setup (cf. Figure 1): We begin with an artifact expressed in a DSL that is unlikely to be significantly represented within the training data of the employed LLM, yet closely resembles another language the LLM was likely trained on. This applies to modeling language variants that were developed internally, or modeling languages that are strongly oriented towards well-known GPLs. Next, we prompt the LLM to perform a minor modification on this artifact. This modification must be defined in such a way that it can be recognized in a later analysis of the finished artifact, e.g. by adding a specific component or wording. After completion, we compare the resulting artifact with the original version, specifically examining alterations unrelated to the task defined in the input prompt. Following the conformity hypothesis (cf. section I), we expect that deviations from the original artifact will predominantly occur in lines where the DSL syntax slightly differs from the syntax of the dominant language present within the LLM training data. In our approach, we use a foundational model (gpt-3.5-turbo (Version 0125), gpt-4o mini (Version 2024-07-18), o1-mini (Version 2024-09-12), deepseek R1 (Version 2025-01-20)) and extend it with an in-context-learning approach (cf. Figure 1) in order to prime

the LLM with prompts to produce the targeted syntax. In our experimental setup, we did not include methods such as constrained decoding [6] or LoRa, however, we used a vector database to provide fitting examples to the FSL-approach [5].

V. Results

For the evaluation, we use the CD4A class diagram of a model-driven information system called MaCoCo [31], [40]. The model from this real-world application defines the core data structure of the data-centric web application and is used to generate the application. It contains 38 enumerations, 72 classes, 63 associations, and 368 attributes, 13 of which have the 'derived' modifier introduced in section III. The text included in the model is in German. The model was chosen for its complexity and its origin in an industrial use case. The modeling language CD4A was chosen because it represents a language whose concepts (class diagrams) are familiar to the LLM, while its syntax is not. For PlantUML, the MaCoCo model was converted into PlantUML syntax.

A. Derived Attributes

In this study we focus on derived attributes, as their notation `'\'` is very similar to the notation for a comment `'\\'`. We expect that this similarity will make it particularly likely (c.f.: Figure 5) that the LLM will adjust the derived attribute to a comment or delete the notation. The MaCoCo class diagram [31], [41] contains several derived attributes[5]. Some of the attributes directly precede each other in the model, thus they should be less impacted by perplexity due to the in-context-learning effects known from few-shot learning. This effect can be observed in Figure 2: The `'Z'` Token is initially very unexpected for the LLM, as indicated by the heatmap. With each new line the model gets used to the pattern until it expects the new line to start with `'Z'`.

```
Add the attribute {attribute_name} to the
class {class_name}. Respond with
the full updated document.
```

Listing 3. The LLM was prompted to add an attribute to a given class and to return the entire modified Class Diagram. The corresponding change to the class diagram, is excluded from all preceeding measurements, as we focus on *unintended changes*.

In order to measure the frequency in which an LLM-Based modeling approach would change aspects of the provided model, a very simple modeling task was given to the system (cf.: Listing 3). This task requires the addition of one new attribute in the given class diagram and should in no circumstance remove or modify any of the existing elements. Therefore, any changes to other unrelated attributes can be classified as erroneous or unintended changes.

In the experiment with the derived statement, there is a high potential for failure, independent of the size of the class diagram used as input. The data (cf.: Figure 3) shows that either no statement is changed, an entire block is modified at once, or all occurrences are changed simultaneously. Table I examines the model corruption based on different input artifact sizes and used LLMs. *GPT-3.5-turbo* can only be applied to the smaller model variants due to context size limitations. In the shortest model variant the LLM does only manage to preserve all attributes in 42 of 100 iterations. Of a total of 600 (6×100 runs) derived attributes, 315 have been modified incidentally. The larger and more capable *gpt-4o-mini* model retains more attributes on average. Changing only a third of all Models. Gpt-4o-mini does not have an issue with fitting larger models in its context window. *o1-mini* is a reasoning model, that excels among others in programming tasks. However, this model still is not capable to reproduce the complete class diagram without corrupting some parts of it. In the 657 lines variant, 3 of 100 files were semantically modified in a sum of 21 lines. In the 1135 lines variant 2 files were modified in a total of 16 lines. Using a deepSeek R1 reasoning model, only one occurrence was unintentionally modified, indicating that reasoning models perform better at this kind of modeling tasks.

CD4A-Model gpt-3.5-turbo (Version 0125)					
Lines	'/'-Notation	Runs	CDs w. Diff	$\sum$ Changes	Error Rate
1135	13	100	N/A (Insufficient context window size)		
657	7	100	N/A (Insufficient context window size)		
222	6	100	58	315	58%
CD4A-Model gpt-4o mini (Version 2024-07-18)					
Lines	'/'-Notation	Runs	CDs w. Diff	$\sum$ Changes	Error Rate
1135	13	100	27	157	27%
657	7	100	24	148	24%
222	6	100	25	78	25%
CD4A-Model o1-mini (Version 2024-09-12)					
Lines	'/'-Notation	Runs	CDs w. Diff	$\sum$ Changes	Error Rate
1135	13	100	2	16	2%
657	7	100	3	21	3%
222	6	100	0	0	0%
CD4A-Model deepseek R1 (Version 2025-01-20)					
Lines	'/'-Notation	Runs	CDs w. Diff	$\sum$ Changes	Error Rate
1135	13	100	1	1	1%
657	7	100	0	0	0%
222	6	100	0	0	0%

TABLE I
Analysis on the frequency of unintended modifications to a CD4A model that contains '/'-Notation to indicate derived attributes (Fewer changes is better).

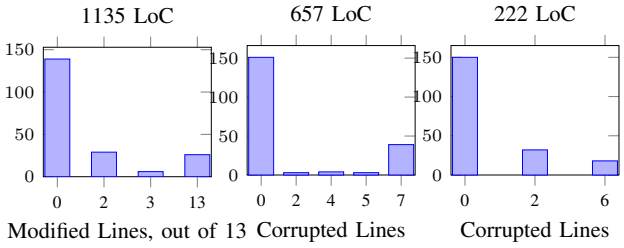

Fig. 3. Changed derived attributes in 100 runs with *gpt-4o-mini* for three datasets. Most derived attributes are preserved without any changes. In case a modification occurs either all derived attributes are corrected, or a low amount is changed. The amount of modified derived attributes is not distributed evenly.

Figure 3 presents the distribution of changes among the

[5]https://zenodo.org/records/6422355

input artifacts provided. A reliable operating algorithm should corrupt 0 lines, or change all occurring derived attribute lines for any artifact size. We notice a tendency for both. Most files remain unchanged; and we also show a tendency to change all occurrences: There are 13, 7 and 6 derived attributes in the respective class diagrams. However there are also files in which only some of the attributes where changed while others remain unchanged, indicating an inconsistency in the behavior of the LLM (gpt-4o-mini).

To minimize the risk that the measurements will be triggered by a certain prompting technique. The experiment was repeated with a different task. In this case, the LLM should only add comments. In this case, the same distribution of changes to the lines could be measured: cf. Figure 4.

In all cases in when a derived attribute was modified, the notation was either removed, making the attribute a regular member variable of the class, or another '/' was added, making the attribute a commented line.

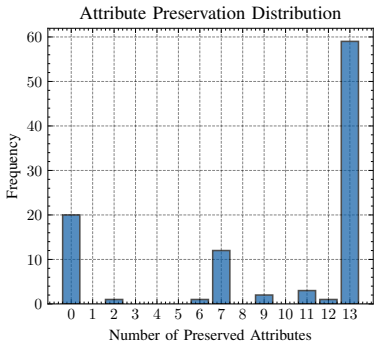

Fig. 4. Distribution of derived attribute preservation (out of 13) across 100 regeneration trials when LLMs were tasked with adding comments. The full CD4A model was used. This measure reflects the results of the findings presented in Figure 3.

CD4A-Model gpt-4o mini (Version 2024-07-18)				
Spelling mistake A: 'Buchung' → 'Buhung'				
Lines	Occurrences	Runs	Corrections by LLM	Error rate
222	1 of 17	100	36	36%
657	1 of 13	100	10	10%
1135	1 of 32	100	16	16%
Spelling mistake B: 'Konto' → 'Kondo'				
Lines	Occurrences	Runs	Corrections by LLM	Error rate
222	1 of 33	100	40	40%
657	1 of 13	100	34	34%
1135	1 of 52	100	17	17%
CD4AModel deepseek R1 (Version 2025-01-20)				
Spelling mistake A: 'Buchung' → 'Buhung'				
Lines	Occurrences	Runs	Corrections by LLM	Error rate
222	1 of 17	100	0	0%
657	1 of 13	100	0	0%
1135	1 of 32	100	2	2%
Spelling mistake B: 'Konto' → 'Kondo'				
Lines	Occurrences	Runs	Corrections by LLM	Error rate
222	1 of 33	100	5	5%
657	1 of 13	100	1	1%
1135	1 of 52	100	2	2%
PlantUML-Model gpt-4o mini (Version 2024-07-18)				
Spelling mistake A: 'Buchung' → 'Buhung'				
Lines	Occurrences	Runs	Corrections by LLM	Error rate
222	1 of 17	100	43	43%
657	1 of 13	100	0	0%
1135	1 of 32	100	15	15%
Spelling mistake B: 'Konto' → 'Kondo'				
Lines	Occurrences	Runs	Corrections by LLM	Error rate
222	1 of 33	100	28	28%
657	1 of 13	100	2	2%
1135	1 of 52	100	5	5%

TABLE II
ANALYSIS ON THE FREQUENCY OF UNINTENDED MODIFICATIONS (CHANGING ATTRIBUTE NAMES) TO A CLASS DIAGRAM IN CD4A AND PLANTUML NOTATION THAT CONTAINS SPELLING MISTAKES (FEWER CORRECTIONS IS BETTER).

B. Spelling Errors

Following the assumption, that the modifications to the model occur due to the semantic interpretation of tokens from the input artifact. We can try to provoke further modifications. Allamong et al. [42] successfully leverages the corrective capabilities of LLM to fix spelling mistakes in texts. Although correct spelling in continuous text is a highly desirable function, adapting individual spellings in textual models can be very disadvantageous. We introduce spelling errors into attribute names and class names in the input artifact and evaluate whether the LLM will preserve these errors. We expect the LLM to have a tendency to correct spelling mistakes it encounters, even if it was not explicitly asked to do so.

In the experiment with incorrect spelling, we noticed a high potential for unintended corrections. A larger input class diagram seems to yield fewer unintended changes to the model; in contrast, smaller models seem to be more susceptible to unintended corrections.

A possible cause could be that larger input artifacts give the LLM a larger context due to which the AI locates the text more strongly in a programming or modeling environment and therefore refrains more from renaming or places a greater focus on the accurate reproduction of variable names. Effects known from few-shot learning could also apply here. Table III presents two kinds of changes. In the first modification the `class` keyword was changed to its german translation `klasse` in one occurrence. In the second modification, all but one occurrence were changed. This was performed for the full class diagram, as well as a shortened variant of it. The MaCoCo class diagram can be divided into semantic blocks (as shown in [31]), giving us the option to choose smaller models within the larger one. We observe that in case of the singular change from `class` to `klasse`, in 13 of 100 iterations the LLM had changed this modification back without being asked to do so. In case the majority of `class` keywords were changed to `klasse`, the LLM did not change any keyword back. Note that also the singular remaining `class` keyword was not changed by the LLM to match the majority `klasse` keywords. This effect was also observed for the smaller class diagram variant.

The observed effect could be explained in parts by in-context-learning principles, and with the training data the LLM is based on: In case only one `class` keyword is modified, the

CD4A-Model gpt-4o mini (Version 2024-07-18)				
Lines	Occurrences	Runs	klasse → class	class → klasse
1135	1 of 71	100	13	0
1135	70 of 71	100	0	0
222	1 of 13	100	13	0
222	12 of 13	100	0	0
PlantUML-Model gpt-4o mini (Version 2024-07-18)				
Lines	Occurrences	Runs	klasse → class	class → klasse
1135	1 of 71	100	0	0
1135	70 of 71	100	0	0
222	1 of 13	100	0	0
222	12 of 13	100	2	0

TABLE III
ANALYSIS ON THE FREQUENCY OF MODIFICATIONS TO A CLASS DIAGRAM IN CD4A NOTATION AN PLANTUML NOTATION WITH A DIFFERENT CLASS KEYWORD (FEWER CORRECTIONS IS BETTER).

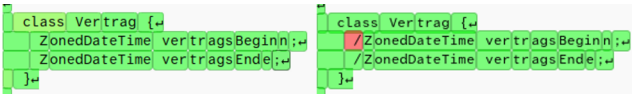

Fig. 5. Perplexity heatmap, for a CD4A class definition. The LLM is 'perplexed' by the added derived attribute notation `'/'` (right image).

LLM is inclined to change it back to the version it was heavily trained on (`class` is very dominant within a programming context in contrast to `klasse`). Thus it is likely to change the keyword back. In the second case, this effect is still present; however, due to the large occurrence of the `klasse` keyword within the input artifact, the LLM is inclined to recognize this anomaly as a pattern and has a stronger tendency to keep it. Similarly the remaining singular `class` keyword in the second case, is still preserved as the LLM is well trained on this keyword and this is not inclined to change it. These effects seem to be also applicable to smaller artifacts.

C. Perplexity

As introduced in subsection IV-A, we use PPL as an indicator to identify if a token is likely to be modified during a modeling task by the LLM (cf. Figure 5). Therefore we compare the complexity of modeling tasks in CD4A with modeling tasks in another DSL. We choose SysML v2[6] [43] for this comparison as it has much higher complexity as a CD4A class diagram. In contrast to CD4A there are more publicly available data describing the SysML syntax. Thus, we can assume, that major LLMs are to a certain extent trained on SysML v2. This can be seen in Table IV: A generic Java file has a low perplexity of 1.84, whereas a comparable the SysML v2 model shows a perplexity of 2.10. Although the CD4A

[6]https://github.com/Systems-Modeling

File Name	Overall Perplexity	Avg. Chunk Perplexity
GenericJavaFile.java	1.84	2.36
Fischertechnik.sysml	2.10	3.05
MaCoCo.cd	2.88	4.70

TABLE IV
PERPLEXITY FOR DIFFERENT ARTIFACT KINDS. JAVA VS. SYSMLV2 VS. CD4A

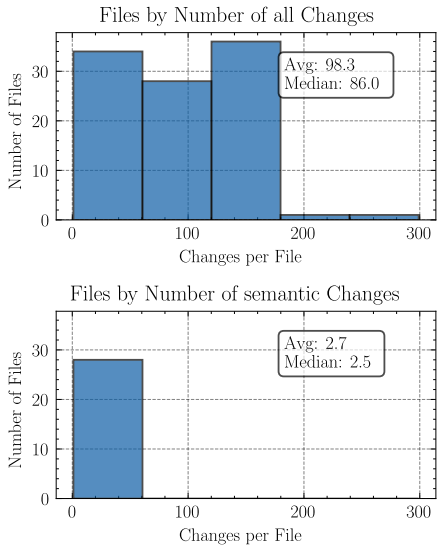

Fig. 6. SysML v2 use case **Top Chart:** The AI-based modeling task introduces several changes into the SysML model. **Bottom Chart:** Filtered for semantic changes, significantly fewer changes remain.

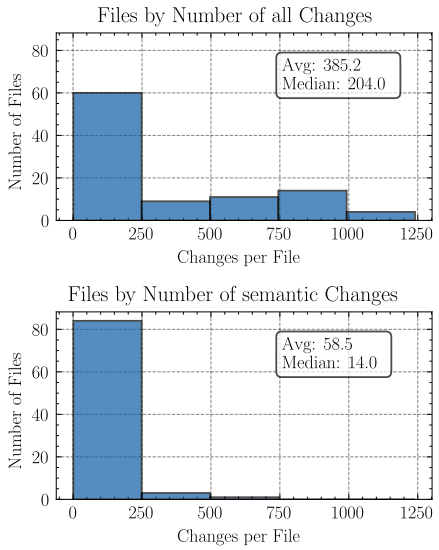

Fig. 7. CD4A use case **Top Chart:** The AI-based modeling task introduces several changes into the CD4A model, some of which are extensive. **Bottom Chart:** Filtered for semantic changes, there is still remains a significant modification to the models.

Class Diagram Syntax is similar to Java is has a higher perplexity at 2.88, indicating that the LLM has more difficulties predicting CD4A code than SysML. Note that programming languages have on average a lower perplexity than natural language, as source code is more machine readable. Next we compare the changes to the models for the different DSLs. We differentiate into generic and semantic changes. Generic changes are changes that modify the model but have no impact on its syntactical interpretation e.g. additional line breaks, changed comments, formatting. Semantic changes [44], [45],

describe changes that result in a different interpretation of the Model, e.g. additional attributes, name changes or component removal. Figure 6 shows overall changes and semantic changes for 100 iterations of a modeling task for a SysML v2 model. We notice that in general the LLM changes several lines in the model. This includes updating comments, white spaces, and formatting. By excluding these changes we notice that median of 2.5 lines where changed unintentionally in 100 SysML files. Given the complexity of the SysML language this is a great result. Looking at the chart for CD4A models: Figure 7 we notice a high level of generic changes, showing a median on 204 changes. If we focus on semantic changes, a median of 14 was measured. Note that these changes include changes to derived attributes, but are not limited to those. We observe more unintended changes in the CD4A model than in the SysML v2 model. More errors do not only occur in absolute numbers, but also in relative ones: In the SysML runs $\frac{2.5}{86.0} = 2.90\%$ of all changes are unintended modifications of the model. In Class Diagram modeling runs $\frac{14.0}{204.0} = 6.86\%$ of all changes where semantic changes that where not intended by the modeler, indicating that the LLM is more capable to retain elements from the SysML model rather than the CD4A model.

VI. Discussion

RQ1: We observe that LLMs tend to inaccurately reproduce semantics of a given model. **RQ2:** This effect increases with the LLM's perplexity of the modeling language. LLMs tend to correct spelling and corrupt unfamiliar syntax of the input model. We have shown that we can provoke this behavior by choosing a language that is unknown to the LLM and contains syntactic elements (derived attributes) that are very close to very well-known modeling languages. It is very likely due to the LLMs not having been trained on certain DSL syntax, thus they frequently alter unfamiliar elements to match more common language patterns from their training data. Although these modifications typically appear plausible and can remain syntactically valid, they unintentionally introduce semantic changes, altering the meaning of the original textual model. This issue cannot be resolved by grammar masking techniques [6], [46], [47], nor can it be reliably detected using a parser. We can reproduce this effect with natural language. By introducing spelling errors, the LLM will have a tendency to interpret the text by its meaning, not by its spelling. If tasked with reproducing, it has a tendency to reproduce the corrected version of the word independently of the original with a spelling error. This can be measured as shown in Table II. The language choice of the model (German) seemed to have less impact than the usage of an unknown DSL, as the LLM correctly represented the German components, and primarily struggled with specifically introduced challenges, such as spelling mistakes and uncommon syntax. Perplexity might serve as an indicator for syntactic elements that are prone to corruption. Especially in DSLs that share patterns with languages the LLM is well trained on, unfamiliar elements will show up as very 'perplexing' tokens (cf. Figure 5). Note that the LLM can familiarize itself with the new token within the context of one large input model, due to in-context-learning effects, so that not all occurrences of the unfamiliar token are seen as unfamiliar by the LLM: E.g. the first occurrence might be unexpected, but with further repetition the LLM can process the new token as a pattern [25]. The reported systematic shortcomings of the LLM highly depend on the training and setup of the used large language model. Thus we expect larger models to reproduce a provided artifact more accurately in comparison to a smaller LLM. This could be shown in Table I. Similarly if the model is trained on the DSL we also expect it to perform better, as shown in Figure 6 and Figure 7. As a consequence techniques such as fine-tuning or LoRa [48], [49] should improve accuracy. More cost-effective and flexible could be in-context-learning based methods [20] and generic prompt engineering [50], such as few-shot learning, chain-of-thought, and function calling. However the effect we measure here will not be extinguished by these improvements as they rely on the inherent hallucinations LLMs have. The key problem that the LLM performs unintended changes might be best mitigated, by introducing an agent, that instead of reproducing the entire model, just provides incremental changes to specific sections of the model. Therefore, a list of dedicated assessable changes is performed in the model instead of an undefined generic update. Such a concept, changing components of a model via a dedicated interface, is already defined for the SysML v2 API [43], [51]. While explicit prompt hints about DSL specific syntax (e.g warning about the '/' notation in CD4A) could reduce error rates, we deliberately avoided such optimizations. Prompt engineering consistently improves LLM performance, as is well documented in literature. However, requiring DSL-specific prompt tuning contradicts the approach of DSL-agnostic modeling. Since time intensive prompt tuning needs to be done for each encountered DSL. While prompt engineering can mitigate some issues, it does not eliminate the fundamental challenge of LLMs misinterpreting unfamiliar patterns.

VII. Threats to Validity

The reproducibility of evaluations for LLM-based experiments poses a special challenge due to their black-box architecture and rapidly evolving nature. We try to tackle this known aspect by running all experiments repeatedly in order to enable us to make statistical statements to specific LLM versions rather than stating claims on the inner workings of an LLM.

A. Low Sample Size

This analysis focused on a limited set of domain-specific languages and models, so we cannot generalize the observed effect to all modeling languages. However, our findings indicate that the effect occurs frequently within the DSLs examined. Since we have consistently observed this effect in our examples, it is reasonable to believe it might also show up in other DSLs with similar features. To better support this idea,

more tests that involve additional DSLs and a wider range of models would help. Future research should therefore include more DSLs and models to better understand how broadly this effect applies.

B. Large Language Models

The field of research in the context of AI is rapidly evolving and changing. New LLMs are released in rapid progression. The effects described above relate to the mentioned versions of the LLMs. Future versions might be less susceptible to introduction of semantic changes to the model at hand. Table I demonstrates that there is a trend of improvement with upcoming models. However, more complex modeling languages could pose a harder challenge for even the newest LLM. Similarly, as shown in the Figure 6 and Figure 7 training data on which the LLM is trained, has a large impact on its performance. Although other LLMs are very likely to have the same effect, they might perform better or worse due to the data they are trained on. The same holds for modified LLMs such as quantized foundational models or fine-tuned models. We only used decoder models for this work because the available encoder-decoder models (as presented in [52], [53]) are nowhere near as powerful in our computing environment as the encoder models that are currently available.

C. Link Between Model Corruption and Perplexity

We examined a link between the perplexity metric and the modification of the tokens identified through this perplexity. We were unable to prove a causal relationship between increased perplexity and increased susceptibility to errors due to many mutually influencing factors, such as in-context learning, but also to the black-box characteristics of the LLM. We can only point to a statistical anomaly and recommend the perplexity metric as a helpful tool to analyze how well an LLM can process a textual model.

VIII. Conclusion

In this paper, we highlight the risks associated with using LLMs for modeling tasks. Beyond the incorrect creation of new elements within existing models, LLMs may also unintentionally alter previously correct model parts, potentially compromising their intended structure or meaning. These changes are not limited to syntax—they can also be semantic in nature and thus escape detection by standard parsers. Analyzing token embeddings using the perplexity metric can provide early indications that a domain-specific language is vulnerable to such effects. Moreover, our findings show that LLMs exhibit a tendency to "correct" perceived spelling errors even in modeling contexts. Such automatic renaming of components can directly impact generated code, particularly in model-driven engineering projects where consistent naming is crucial.

Larger LLMs can process unknown DSLs better and, therefore, perform better in our measurements. However, even with modern reasoning models (o1, deepseek R1), we were still able to demonstrate a susceptibility to errors, although those LLMs performed significantly better.

In addition to the model size, better prompting also has an effect on the susceptibility to errors. However, both model size and in-context learning or constrained decoding do not completely prevent model corruption. We recommend not adapting the model directly to the textual artifact, but using an agent system that translates the modeler's requirements into concrete, traceable and verifiable changes on an abstract level, rather than guiding the AI to produce syntax for the entire model directly, similar to the API-Based modeling approach proposed by OMG for the SysMLv2 [54].

This work primarily measured changes to class diagrams that are defined in few DSLs: CD4A and PlantUML. With a small set of models and LLMs. Experimentation with further DSLs and also open Source LLMs would support the claim of the paper and provide further insights to what extent this problem impacts generic modeling tasks and further determine what kind of domain-specific languages are especially prone to this kind of correction and corruption. Further research should also focus on an approach to efficiently mitigate unintended changes to unrelated elements of the input model, while still allowing dedicated modifications to the model as a whole.

References

[1] X. Gu, M. Chen, Y. Lin, Y. Hu, H. Zhang, C. Wan, Z. Wei, Y. Xu, and J. Wang, "On the effectiveness of large language models in domain-specific code generation," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 3.

[2] T. Huang, Z. Sun, Z. Jin, G. Li, and C. Lyu, "Knowledge-aware code generation with large language models," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*.

[3] M. Macedo, Y. Tian, F. Cogo, and B. Adams, "Exploring the impact of the output format on the evaluation of large language models for code translation," in *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*, FORGE '24, (New York, NY, USA), p. 57–68, Association for Computing Machinery.

[4] L. Netz, J. Michael, and B. Rumpe, "From Natural Language to Web Applications: Using Large Language Models for Model-Driven Software Engineering," in *Modellierung 2024* (M. Weske and J. Michael, eds.), LNI, GI, 2024.

[5] N. Baumann, J. S. Diaz, J. Michael, L. Netz, H. Nqiri, J. Reimer, and B. Rumpe, "Combining Retrieval-Augmented Generation and Few-Shot Learning for Model Synthesis of Uncommon DSLs," in *Modellierung 2024 - Workshopband* (H. Giese and K. Rosenthal, eds.), GI, 2024.

[6] L. Netz, J. Reimer, and B. Rumpe, "Using Grammar Masking to Ensure Syntactic Validity in LLM-based Modeling Tasks," in *Workshop on Artificial Intelligence and Model-driven Engineering, MODELS Companion '24: International Conference on Model Driven Engineering Languages and Systems (MDE Intelligence)*, Association for Computing Machinery (ACM), 2024.

[7] C. D. Costa, J. A. H. López, and J. S. Cuadrado, "Modelmate: A recommender for textual modeling languages based on pre-trained language models," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*.

[8] 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.

[9] F. Mora, J. Wong, H. Lepe, S. Bhatia, K. Elmaaroufi, G. Varghese, J. E. Gonzalez, E. Polgreen, and S. Seshia, "Synthetic programming elicitation for text-to-code in very low-resource programming and formal languages," *Advances in Neural Information Processing Systems*, vol. 37.

[10] P. Wu, N. Guo, X. Xiao, W. Li, X. Ye, and D. Fan, "Itertl: An iterative framework for fine-tuning llms for rtl code generation."

[11] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation."

[12] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. OOPSLA1.

[13] O. Macmillan-Scott and M. Musolesi, "(ir) rationality and cognitive biases in large language models," *Royal Society Open Science*, vol. 11, no. 6.

[14] X. Zhu, C. Zhang, T. Stafford, N. Collier, and A. Vlachos, "Conformity in large language models."

[15] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems."

[16] Y. Hu, Q. Huang, M. Tao, C. Zhang, and Y. Feng, "Can perplexity reflect large language model's ability in long text understanding?."

[17] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, vol. 1, no. 2.

[18] C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He, *et al.*, "A comprehensive survey on pretrained foundation models: A history from bert to chatgpt," *International Journal of Machine Learning and Cybernetics*.

[19] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM computing surveys*, vol. 55, no. 9.

[20] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, T. Liu, *et al.*, "A survey on in-context learning," *arXiv preprint arXiv:2301.00234*.

[21] M. Kammakomati, S. Pimparkhede, S. Tamilselvam, P. Kumar, and P. Bhattacharyya, "Concodeeval: Evaluating large language models for code constraints in domain-specific languages."

[22] N. Bassamzadeh and C. Methani, "A comparative study of dsl code generation: Fine-tuning vs. optimized retrieval augmentation."

[23] X. Gu, M. Chen, Y. Lin, Y. Hu, H. Zhang, C. Wan, Z. Wei, Y. Xu, and J. Wang, "On the effectiveness of large language models in domain-specific code generation," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 3.

[24] A. X. Yang, M. Robeyns, X. Wang, and L. Aitchison, "Bayesian low-rank adaptation for large language models."

[25] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33.

[26] Y. Huang and J. Huang, "A survey on retrieval-augmented text generation for large language models."

[27] M. Chen, H. Tian, Z. Liu, X. Ren, and J. Sun, "JumpCoder: Go beyond autoregressive coder via online modification," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (L.-W. Ku, A. Martins, and V. Srikumar, eds.), (Bangkok, Thailand), pp. 11500–11520, Association for Computational Linguistics.

[28] K. Hölldobler, O. Kautz, and B. Rumpe, *MontiCore Language Workbench and Library Handbook: Edition 2021*. Aachener Informatik-Berichte, Software Engineering, Band 48, Shaker Verlag, 2021.

[29] K. Hölldobler, A. Roth, B. Rumpe, and A. Wortmann, "Advances in Modeling Language Engineering," in *International Conference on Model and Data Engineering*, LNCS 10563, Springer, 2017.

[30] M. Heithoff, N. Jansen, J. C. Kirchhof, J. Michael, F. Rademacher, and B. Rumpe, "Deriving Integrated Multi-Viewpoint Modeling Languages from Heterogeneous Modeling Languages: An Experience Report," in *Proceedings of the 16th ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2023, (Cascais, Portugal), pp. 194–207, Association for Computing Machinery.

[31] A. Gerasimov, P. Letmathe, J. Michael, L. Netz, and B. Rumpe, "Modeling Financial, Project and Staff Management: A Case Report from the MaCoCo Project," *Enterprise Modelling and Information Systems Architectures - International Journal of Conceptual Modeling*, vol. 19, 2024.

[32] B. Combemale, R. France, J.-M. Jézéquel, B. Rumpe, J. Steel, and D. Vojtisek, *Engineering modeling languages: Turning domain knowledge into tools*. CRC Press.

[33] A. Roques, "Plantuml: Open-source tool that allows users to create uml diagrams from plain text language." Accessed: 2025-06-25.

[34] B. Wang, C. Wang, P. Liang, B. Li, and C. Zeng, "How llms aid in uml modeling: an exploratory study with novice analysts," in *2024 IEEE International Conference on Software Services Engineering (SSE)*, IEEE.

[35] J. Xu, H. Zhang, Y. Yang, Z. Cheng, J. Lyu, B. Liu, X. Zhou, L. Yang, A. Bacchelli, Y. K. Chiam, and T. K. Chiew, "Investigating efficacy of perplexity in detecting llm-generated code."

[36] G. Alon and M. Kamfonas, "Detecting language model attacks with perplexity."

[37] Z. Xu and V. S. Sheng, "Detecting ai-generated code assignments using perplexity of large language models," in *Proceedings of the aaai conference on artificial intelligence*, vol. 38.

[38] A. Gerasimov, P. Heuser, P. Letmathe, J. Michael, L. Netz, B. Rumpe, S. Varga, and G. Volkova, "Domain Modelling of Financial, Project and Staff Management." Zenodo, https://doi.org/10.5281/zenodo.6422355.

[39] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann, *et al.*, "Phi-4 technical report," *arXiv preprint arXiv:2412.08905*.

[40] A. Gerasimov, P. Heuser, H. Ketteniß, P. Letmathe, J. Michael, L. Netz, B. Rumpe, and S. Varga, "Generated Enterprise Information Systems: MDSE for Maintainable Co-Development of Frontend and Backend," in *Companion Proceedings of Modellierung 2020 Short, Workshop and Tools & Demo Papers* (J. Michael and D. Bork, eds.), CEUR Workshop Proceedings, 2020.

[41] A. Gerasimov, P. Heuser, P. Letmathe, J. Michael, L. Netz, B. Rumpe, S. Varga, and G. Volkova, "Domain modelling of financial, project and staff management."

[42] M. B. Allamong, J. Jeong, and P. M. Kellstedt, "Spelling correction with large language models to reduce measurement error in open-ended survey responses," *Research & Politics*, vol. 12, no. 1.

[43] N. Jansen, J. Pfeiffer, B. Rumpe, D. Schmalzing, and A. Wortmann, "The language of sysml v2 under the magnifying glass.," *J. Object Technol.*, vol. 21, no. 3.

[44] D. Harel and B. Rumpe, "Meaningful Modeling: What's the Semantics of "Semantics"?," *IEEE Computer Journal*, vol. 37, no. 10, 2004.

[45] B. Rumpe, M. Stachon, S. Stüber, and V. Voufo, "Semantic Difference Analysis with Invariant Tracing for Class Diagrams Extended by OCL," in *Workshop on Model Driven Engineering, Verification and Validation, MODELS Companion '24: International Conference on Model Driven Engineering Languages and Systems (MoDeVVa)*, Association for Computing Machinery (ACM), 2024.

[46] B. Wang, Z. Wang, X. Wang, Y. Cao, R. A. Saurous, and Y. Kim, "Grammar prompting for domain-specific language generation with large language models."

[47] S. Ugare, T. Suresh, H. Kang, S. Misailovic, and G. Singh, "Syncode: Llm generation with grammar augmentation," *arXiv preprint arXiv:2403.01632*.

[48] D. Biderman, J. Portes, J. J. G. Ortiz, M. Paul, P. Greengard, C. Jennings, D. King, S. Havens, V. Chiley, J. Frankle, *et al.*, "Lora learns less and forgets less," *Transactions on Machine Learning Research*.

[49] B. Chen, F. Yi, and D. Varró, "Prompting or fine-tuning? a comparative study of large language models for taxonomy construction," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, IEEE.

[50] R. Clarisó and J. Cabot, "Model-driven prompt engineering," in *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, IEEE.

[51] M. Manoury and C. Muggeo, "Praktische anwendung der sysml v2 api am beispiel von mcad und simulation," in *Tag des Systems Engineering 2023*.

[52] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30.

[53] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics.

[54] A. Ahlbrecht, B. Lukić, W. Zaeske, and U. Durak, "Exploring sysml v2 for model-based engineering of safety-critical avionics systems," in *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*.