Bridging Models and Language: An Encoder-Decoder Approach for Automated Architectural Documentation with LLMs

Lasse M. Reinpold and Felix Gehlhoff
Institute of Automation Technology
Helmut Schmidt University Hamburg, Germany
Email:{lasse.reinpold, felix.gehlhoff}@hsu-hh.de

Hussein Hasso and Hanna Geppert
Institut für Kommunikation, Informationsverarbeitung und Ergonomie
Fraunhofer FKIE, Wachtberg, Germany
Email: hussein.hasso@fkie.fraunhofer.de

***Abstract*—In modern software development, maintaining consistency between architectural documentation and implementation remains a significant challenge. This research explores how large language models (LLMs) can be integrated into an encoder-decoder framework to enable real-time architectural documentation and seamless conversion between semi-formal models (e.g., UML diagrams) and natural language descriptions. Building on previous work in LLM-based requirements verification, we develop a system that automatically converts models into text and vice versa. By allowing LLMs to assess textual descriptions for compliance with requirements, this approach reduces the need for extensive model-side validation through rule-based queries. Additionally, the decoder helps transform text-based process descriptions into structured architectural models, supporting digitalization in organizations. The system's effectiveness is evaluated by comparing reconstructed models with their originals, assessing how well information is preserved and how accurately the transformations are performed.**

***Index Terms*—Large Language Models, Architectural Documentation, Model-to-Text Conversion, Text-to-Model Conversion, Requirements Engineering, UML**

I. Introduction

As industries pursue digital transformation and intelligent automation, the demand for accurate, up-to-date system documentation has increased significantly. In complex engineering domains, model-based systems engineering (MBSE) has emerged as a powerful approach for system development and verification [1]. MBSE centers on creating abstract representations of systems using semi-formal modeling languages like unified modeling language (UML), SysML, or domain-specific languages. These models serve as the primary artifacts throughout the development lifecycle - from requirements specification and system design to implementation and testing. By providing structured, visual representations of system logic, behavior, and architecture, MBSE reduces ambiguity and improves consistency in system design [2]. It enables early error detection through simulation and validation before implementation, potentially preventing costly design flaws. Engineers can verify system properties, test edge cases, and generate code directly from validated models [1]. The visual models can also facilitate better collaboration between engineers, managers, and domain experts by providing a shared reference point for system understanding [1]. MBSE supports iterative development through model refinement, where high-level abstract models are progressively detailed into concrete implementations [2].

Despite its advantages, MBSE faces practical challenges in implementation and maintenance. Maintaining synchronization between models, textual documentation, and actual system changes remains a persistent bottleneck. The manual effort required to keep these three aspects aligned is error-prone and labor-intensive, with updates often lagging behind—undermining the very benefits of traceability, compliance, and cross-disciplinary collaboration that MBSE aims to provide. Additionally, Traditional approaches to documentation and model validation rely heavily on human effort and domain expertise, with semi-formal models such as UML or SysML not being easily accessible to non-technical stakeholders [3].

This exploratory research investigates how LLMs could potentially be leveraged to automate the generation and validation of architectural documentation. Through early experiments, we aim to understand approaches for making system models more accessible, easier to maintain, and less dependent on manual updates.

We propose a two-way transformation system, which we refer to as an encoder-decoder approach: a model encoder converts semi-formal models into structured natural language descriptions, while a decoder reconstructs those models from text. This approach not only enables practical transformations but also provides a framework for quantitative evaluation of LLMs' capabilities in understanding, describing, and generating semi-formal models, which is difficult to achieve when only investigating one part of the transformation. Therefore, we can establish the reliability of LLMs in performing transformations between natural and semi-formal language. This foundational capability paves the way for potential future benefits such as automatic maintenance of documentation and verification processes. Once validated, LLMs could also potentially automate the synchronization of documentation with the underlying architecture and facilitate the validation of textual process descriptions against high-level requirements—transforming validation from technical rules into intuitive natural language checks.

This work is guided by two key research questions:

How accurately can LLMs translate semi-formal system models into human-readable documentation?

Can LLMs reliably reconstruct structured models from informal or semi-structured text?

This research addresses these questions through a systematic experimental approach. We implement our encoder-decoder framework using state-of-the-art LLMs and evaluate it using a publicly available dataset of UML models in XMI format, a standardized XML-based representation that enables model interchange between different tools and platforms. To assess the effectiveness of our approach, we establish metrics that measure both the semantic accuracy of the generated documentation and the structural fidelity of the reconstructed models.

The remainder of this paper is structured as follows: Sec. II surveys related work in automated documentation and model transformation. Sec. III introduces our encoder-decoder architecture and implementation, including the experimental design, dataset selection and evaluation metrics. Sec. IV presents transformation results and model similarity scores. Sec. V discusses the implications and limitations of our findings as well as future research directions.

II. Related Works

We divide the existing literature into two groups based on (i) whether they implement a two-way transformation between semi-formal models and natural language descriptions (yet do not leverage Large Language Models, LLMs), or (ii) whether they do use LLMs but only perform a one-way conversion, instead of a two-way conversion as is proposed in our approach. Below, we briefly present each group, highlight their goals, applied technologies, and main findings, and then explain—on a group level—how they differ from our approach.

A. Two-Way Transformation Without LLMs

The following works use natural language processing (NLP) technologies to enable round-trip transformations between semi-formal models and textual descriptions. They do not, however, rely on LLMs, nor do they propse a comparable encoder-decoder pipeline. In contrast to our approach, these studies typically employ more traditional NLP methods (e.g., rule-based or template-based) and do not integrate an LLM for real-time documentation or fully automated back-and-forth conversion.

Guerreiro Azevedo et al. [4] automate synchronization between business process model and notation (BPMN) models and textual instructions to maintain consistency in process descriptions. They employ traditional NLP methods, including linguistic analysis and pattern matching, for round-trip updates. Their work demonstrates 74–78% equivalence in knowledge preservation, reducing manual effort by keeping BPMN models and text descriptions aligned.

Ballard et al. [5] convert text-based requirements to SysML requirements (and vice versa) to reduce manual overhead in system engineering. Their approach uses a rule-based NLP pipeline that incorporates tokenization, part-of-speech tagging, and requirement templates to parse requirements and generate SysML artifacts. They establish the feasibility of automated bidirectional transformations, though their evaluations focus on template matching rather than extensive quantitative metrics.

Thomas Freytag et al. [6] provide an API for converting BPMN and petri net markup language (PNML) process models into text and back. Their technology integrates open-source NLP tools, such as Stanford CoreNLP and WordNet, within a web service to handle model-to-text and text-to-model processes. Their findings demonstrate practical two-way conversions for BPMN, though the evaluations are primarily illustrative rather than quantitative.

B. LLM-Based Approaches Without Two-Way Conversion

The following works do leverage LLMs to transform between semi-formal models and natural language or to generate textual artifacts from model data.

de Bari et al. [7] assess how LLMs (e.g., ChatGPT) generate UML class diagrams from textual requirements in an educational setting. They employ GPT-4o to generate PlantUML code and analyze syntactic, semantic, and pragmatic correctness. Their findings show that LLM-generated diagrams show promise yet exhibit more semantic errors than human-created ones, highlighting challenges in capturing domain meaning.

Ferrari et al. [8] investigate how well an LLM produces UML sequence diagrams from textual requirements. Their technology utilizes ChatGPT for text-to-diagram generation. They find that generated diagrams are understandable but lack completeness and correctness, particularly for ambiguous or domain-specific scenarios.

Härer [9] implement a "conceptual model interpreter" that uses LLMs to create and refine visual models in an iterative dialogue. Their approach combines ChatGPT or LLaMA with PlantUML/Graphviz interpreters. They demonstrate feasible conversational model-building, but do not provide quantitative performance metrics.

Köpke and Safan [10] present a BPMN-focused chatbot that turns text or voice input into BPMN processes while minimizing token usage. Their technology employs an LLM-based prompt strategy with an intermediate JSON format for efficient BPMN generation. Their findings show significant cost savings and high correctness compared to existing BPMN generation approaches.

Kourani et al. [11] use LLMs to automate process model generation (BPMN, Petri nets) from textual descriptions, with iterative refinement. They employ prompt engineering, a secure code-generation workflow, and error handling. Their findings indicate that iterative feedback and validation significantly improve model soundness, handling domain ambiguities more effectively.

Naimi et al. [12] generate textual documentation (e.g., use case descriptions) from UML diagrams to save time on software documentation. Their technology extracts UML data via XML, then uses LLM prompts to produce descriptive text. They find that this approach speeds up documentation creation; however, a lack of quantitative measures means the exact accuracy of generated text remains unclear.

Neuberger et al. [13] propose a universal prompting strategy for extracting process model information from natural language using LLMs. Their technology uses modular prompts with GPT-4o to extract process entities (activities, actors, relations) from text and evaluate extraction accuracy across several datasets. They achieve state-of-the-art performance for process information extraction, with up to 8% F1 improvement over traditional baselines. However, their approach focuses solely on unidirectional text-to-model extraction and does not support transformation from semi-formal models back to text.

Taken together, these works demonstrate that while LLMs show promise in model-text transformations, they focus predominantly on one-directional use—often text-to-model generation or model-to-text description—rather than a fully integrated two-way pipeline. Moreover, several of these approaches rely on qualitative or partial evaluations, or they target narrower tasks (e.g., creating a single type of UML diagram). In contrast, our proposed system uses a two-way, encoder-decoder approach that integrates LLMs for both directions—semi-formal-to-text and text-to-semi-formal—and includes plans for quantitative evaluation of transformation accuracy.

C. *Summary of related works*

Overall, prior work demonstrates the feasibility of converting semi-formal models into natural language or vice versa, either through traditional NLP approaches (often achieving round-trip consistency but without LLMs) or by employing LLMs in a one-directional manner (text-to-model or model-to-text), frequently with partial or qualitative evaluation. Our research aims to integrate these strengths—offering two-way conversion within an LLM-driven encoder-decoder framework—while adding quantitative evaluation (e.g., preserving information fidelity and assessing reconstruction accuracy) that enables systematic measurement of LLMs' capabilities in understanding, describing, and generating semi-formal models.

Several important challenges surfaced across the literature: the difficulty of ensuring semantic correctness, handling domain-specific ambiguities, designing effective prompt engineering strategies, and establishing robust, quantitative evaluation metrics. These insights guide our work in devising model-text transformations that preserve architectural details, minimize errors, and streamline real-time documentation within organizations—ultimately contributing a more comprehensive and empirically validated approach to bridging models and language.

III. Methodology

This section presents the methodological foundation of our approach for transforming UML models into natural language and vice versa. We detail the technical components, evaluation metrics, and experimental setup that enable a systematic assessment of the transformation quality.

It is structured into four parts: the encoder-decoder architecture, the prompting strategy used to guide the LLM behavior, the metrics defined for evaluating transformation quality, and the benchmark dataset used in the experiments.

A. *Encoder-Decoder Approach*

At the heart of our system lies a two-stage encoder-decoder pipeline that enables the transformation between structured UML models and natural language descriptions. The encoder receives a UML model—represented in XMI format—and generates a structured textual description that captures the model's elements and relationships. The decoder then processes this textual description and regenerates the UML model in XMI format.

This bidirectional transformation potentially supports a range of use cases, including automatic creation of model documentation, model verification, and model re-generation from process descriptions.

The transformation pipeline follows these steps:

1. The process starts with a UML model in XMI format.
2. The encoder transforms the model into natural language, including both an element overview and a description of how the elements are connected.
3. The decoder uses the natural language description to regenerate the UML model structure.
4. The original and reconstructed models are compared by suitable metrics to assess transformation fidelity.

B. *Prompting Approach*

To guide the behavior of the LLM during encoding and decoding, we use a structured prompting approach that includes context setting, task breakdown, format constraints, and few-shot examples. The prompting approach is based on the approach proposed by Neuberger et al. [13], who developed and tested a universal prompting approach for the extraction of process model information from text in natural language [13]. Due to this focus on information extraction from natural text, some modifications were implemented to adapt the prompting approach to our encoder-decoder approach.

The structure of the prompt for the encoder is shown in Listing 1.

```
### CONTEXT
You are an expert in business process and UML modeling with
  deep knowledge of BPMN, UML, and XMI structures. Your
 task is to generate a coherent, human-readable description
  from an XMI export of a process or UML model. [...].

### TASK DESCRIPTION
- Read the given XMI file line by line.
- Identify relevant model elements and determine how they
 are connected.
- Summarize the discovered model elements in natural
 language.
- Ensure each step, use case, or section makes clear what
 happens and who is involved.
- Emphasize process flows and simple causal chains (Chain-
 of-Thought).
- Briefly justify why certain steps follow each other (
 where possible). [...].
```

```
- Add a short Reflection summarizing key points.

### RESTRICTIONS & FORMAT INSTRUCTIONS
- The final output should be a continuous, well-readable
  paragraph in German.
- If intermediate steps are included, mark them under a
  separate heading.
- Clearly distinguish repeated elements.
- Use generic language for ambiguous elements.
- If information is missing in the model, leave it out in
  the text description.
- Rely solely on metadata and elements present in the XMI.
  If parts are unclear, mention this in the reflection
  instead of speculating. [...].

## EXAMPLE (Excerpt)
### Input: A simplified XMI export of a BPMN process with
  user tasks and sequence flows. [...]
### Output: A linear textual description of the process
  steps.
### Reflection: Notes on the structure (e.g., linear, no
  branching) and model limitations.
### Chain-of-Thought: Explanation of extraction logic, e.g
  ., identifying userTask elements and reconstructing order
  based on sequenceFlow.

### XMI Input: [...]
```

Listing 1. Encoder Prompt (adjusted from Neuberger et al. [13])

The prompt is structured into four main sections: *Context*, *Task Description*, *Restrictions and Format Instructions*, and an *Example*. The context defines the LLM's role as a domain expert interpreting XMI-based models. The task description outlines the steps for parsing model elements, identifying relationships, and generating coherent textual descriptions. Restrictions guide output format, language, and handling of ambiguity, ensuring consistency and traceability. An annotated example is included as a one-shot prompt to demonstrate expected input-output behavior and reinforce structural clarity. Depending on whether the UML model is a structure diagram or a process diagram, the *task description* is adapted to the specific model type, by either highlighting the importance of model element hierarchy and relationships (structure diagrams) or the process flow and causal chains (process diagrams).

To ensure optimal transformation quality across different model types, we systematically refined the prompting approach through multiple iterations. This refinement process focused on adapting the task descriptions and examples to fully capture the specific characteristics and requirements of each model type, while maintaining the core structure established by Neuberger et al. [13].

The decoder prompt follows the same structure as the encoder prompt. However, while the encoder guides the LLM to generate fluent, human-readable documentation from XMI, the decoder focuses on reconstructing syntactically precise UML models from text. It prioritizes XML correctness and structural fidelity, whereas the encoder emphasizes interpretability and narrative clarity. Templates for both prompts are available on GitHub[1].

C. Evaluation Metrics

To quantitatively assess the fidelity of the model transformations produced by our encoder-decoder system, we adopt a graph-based similarity metric grounded in the Weisfeiler-Lehman (WL) graph kernel [14]. The WL kernel is a well-established method in graph mining and machine learning for capturing topological and label-based similarities between structured data representations [15].

Rationale for Graph-Based Evaluation: Since UML models in XMI format can be naturally interpreted as directed labeled graphs—where nodes represent model elements (e.g., classes, attributes, states) and edges correspond to structural or behavioral relationships (e.g., composition, transitions)—graph-based similarity provides a theoretically grounded approach for evaluating the structural and semantic preservation of the transformation.

Compared to string- or token-based similarity measures (e.g. Levensthein distance, Jaccard Index), graph kernels offer the advantage of capturing both local and global structural patterns, making them particularly suitable for the complex hierarchical organization of UML models.

Graph Construction: For each original and reconstructed XMI model, we parse the XML structure and convert it into a directed graph using NetworkX [16]. The conversion process preserves both the hierarchical organization and the semantically relevant content of the UML model.

Each node in the graph represents an XML element from the XMI source file. Nodes are assigned an enriched label that encodes key semantic information extracted from the element's attributes. Specifically, the label includes the element type (e.g., `Class`, `Attribute`, `State`), derived from the XML tag, as well as selected UML-relevant attributes, such as `name` and `type`, if present. This design ensures that the similarity metric is sensitive not only to the structural topology of the model but also to its domain-specific semantics.

Edges between nodes directly reflect the parent-child relationships inherent in the XML tree structure, thereby preserving the hierarchical organization of the original model in the graph representation. The resulting graphs are directed and labeled, making them well-suited for subsequent similarity computation using graph kernel methods.

This approach enables a fine-grained and interpretable comparison between models by leveraging both structural and semantic cues encoded in the original UML artifacts.

Interpretation of Similarity Scores: Similarity scores derived from the WL graph kernel offer a quantitative measure of structural and semantic fidelity between graphs. While absolute thresholds are task-dependent, we observe the following trends in our experimental context:

- Scores approaching 1.0 generally correspond to near-identical preservation of both structure and node labels.
- Scores between 0.6 and 0.9 indicate partial similarity, often reflecting minor structural variations or label mismatches.
- Scores below 0.6 typically suggest substantial divergence, such as missing substructures or semantic drift.

Activity Diagram Similarity Evaluation: While the Weisfeiler-Lehman (WL) graph kernel provides a robust measure of structural similarity for UML models in general, it

[1]Placeholder for GitHub link

does not directly capture the semantic or behavioral similarity of process-oriented models such as UML activity diagrams. To complement the graph-based evaluation, we implement an additional similarity measure specifically tailored to activity diagrams.

Motivation: Activity diagrams primarily capture workflows and procedural logic rather than hierarchical structure. Their semantics are often encoded in the sequence of actions and the involved entities. Consequently, evaluating the similarity of activity diagrams benefits from analyzing their process descriptions on a textual level.

Similarity Computation: To evaluate the similarity between original and reconstructed activity diagrams, we compute a behavior-oriented similarity score based on their respective XMI representations. This approach is designed to capture both content preservation and sequence similarity of the described process steps.

First, all nouns and verbs are extracted from the XMI files using part-of-speech tagging provided by spaCy. Nouns typically correspond to entities involved in the process (e.g., actors, data objects), while verbs represent actions or events within the activity flow.

Subsequently, the extracted sequences of nouns and verbs are compared separately using two complementary methods:

- The longest common subsequence (LCS) ratio assesses content overlap while preserving word order. It measures how much of the original content is retained in the reconstruction.
- Spearman's rank correlation coefficient evaluates the relative ordering of shared words between both sequences, independent of their absolute positions.

The final similarity score is calculated as the product of the averaged LCS scores and the averaged Spearman correlation scores across nouns and verbs:

$$\text{Similarity} = \left(\frac{LCS_{nouns} + LCS_{verbs}}{2}\right) \times \left(\frac{Spearman_{nouns} + Spearman_{verbs}}{2}\right) \tag{1}$$

While both the LCS ratio and Spearman's rank correlation coefficient are sensitive to ordering, they capture complementary aspects of sequence similarity. The LCS ratio rewards the preservation of long, uninterrupted subsequences, providing robustness to noise and intermediate insertions. In contrast, Spearman's correlation evaluates the global monotonicity of element order, penalizing substantial reordering of process steps. This combined metric rewards reconstructions that preserve both the relevant process content and the logical execution order of the original activity diagram. It complements the graph-based evaluation by providing a semantic and behavior-oriented perspective on transformation fidelity.

D. Benchmark Files

All UML models used in the experiments are sourced from a publicly available dataset curated by Robles et al. [17]. The dataset includes over 90,000 real-world UML diagrams mainly in JPEG, PNG, UML, and XMI formats extracted from open-source software repositories on GitHub.

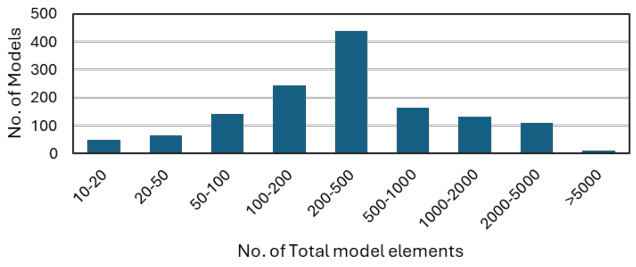


Figure 1. Model size distribution in the benchmark dataset.

From the dataset, 2,827 models in XMI format were retrieved. An initial selection filtered out models that are not in English or German, as well as those unrelated to the documentation of business processes or software systems, resulting in 1,800 models. Some models could not be processed due to corrupted files or because they exceeded the size of the context window of the LLM. The final dataset for the evaluation of the encoder-decoder system consists of 1,165 structure diagrams and 161 process diagrams, with model sizes ranging from 12 to 6,371 elements, with most models containing between 100 and 1000 elements, as shown in Fig. 1.

For all experiments, GPT-4o was chosen exemplarily due to its proven performance in various tasks, but the system is designed to be agnostic to the LLM used.

IV. Results

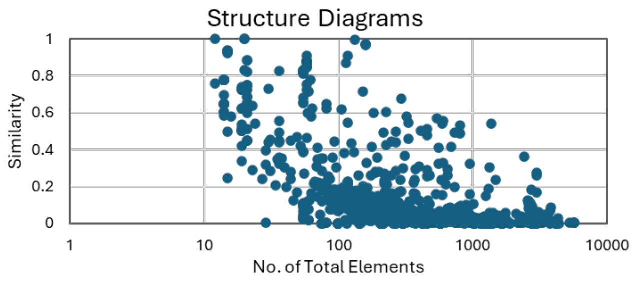
This section presents the evaluation results of our encoder-decoder framework, as described in Sec. III. We analyze the system's performance in transforming UML models from the benchmark dataset into natural language descriptions and subsequently regenerating those models from text. Our analysis focuses on two levels: a macroscopic analysis examining overall accuracy trends across different model sizes, and a microscopic analysis investigating specific transformation patterns.

A. Macroscopic Analysis: Accuracy vs. Model Size

To evaluate transformation fidelity at scale, we applied our encoder-decoder system to the benchmark dataset described in Sec. III.

The evaluation metrics presented in Sec. III-C are used to assess the conformity of the regenerated models to the original models. The overall results are shown in Fig. 2. For every transformed model, the similarity score is plotted against the original model size. Note the logarithmic scale on the x-axis. Process diagrams are shown in the top panel, and structure diagrams are shown in the bottom panel.

Fig. 2 shows that adequate transformation results (similarity scores above 0.6) are only observed in smaller models (less than 100 elements) with few exceptions. While some near perfect transformations are observed for small models in the range of 10-20 elements, the accuracy drops significantly for larger models. The decline in similarity scores with increasing model size follows a power-law trend, where accuracy



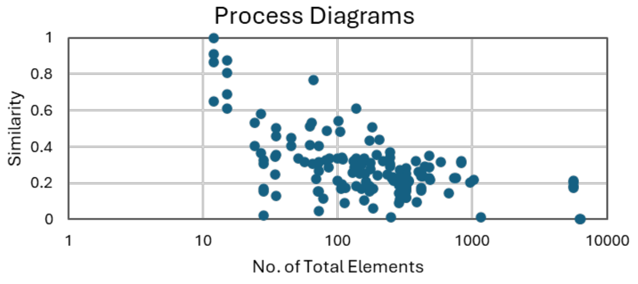


Figure 2. Transformation results for structure diagrams (top) and process diagrams (bottom).

decreases rapidly for smaller models and approaches 0 for larger ones. For models with more than 100 elements, low similarity scores are observed for the vast majority of the models. The transformation results exhibit comparable patterns for both structure and process diagrams, suggesting that model size is a more significant determinant of transformation quality than the specific diagram type. The decrease in similarity scores for larger models can in part be attributed a certain length-bias of the LLM, as the chosen LLM GPT-4o typically does not generate aswers that are longer than 4096 tokens. Since low similarity scores are also observed for many smaller models, a microscopic analysis of individual transformations is conducted to identify specific weaknesses of the encoder-decoder system.

B. *Microscopic Analysis of Individual Transformations*

Having established that the quality of the model transformation decreases rapidly with increasing model size, this section investigates individual transformation results to identify specific weaknesses of the encoder-decoder system. As an exemplary model, we chose a structure diagram of a clock tower. The chosen model has 189 total elements and the similarity score of the transformation is 0.06. Representations of the original and reconstructed model trees are shown in Fig. 3. The figure shows the structure of the model and the relevant named model elements, such as classes, attributes, and associations. Green entries indicate elements that are present in the reconstructed model but not in the original model, while red entries indicate elements that are present in the original model but are lacking in the reconstructed model. The added elements include new actions and a composite state that were not present in the original model, while metamodel annotations, association ends, and hierarchical structure elements were omitted. These changes reflect a tendency of the decoder to hallucinate behavioral details while overlooking structural metadata and semantic context.

The comparison between the original and reconstructed model trees reveals both strengths and limitations of our encoder-decoder architecture. While the system successfully preserves core structural and behavioral elements, several key differences emerge that warrant discussion.

The reconstructed model demonstrates strong performance in maintaining fundamental architectural components. The high-level structure, including key packages (Environment), classes (ClockTower, Clock, Hand), and their attributes (currentHour, currentMinute, hourHand, minuteHand, clock) are accurately reproduced. The system also successfully preserves the primary operations (updateHands, addOneHour, addOneMinute, and the Clock constructor) and the core elements of the ClockStateMachine, including pseudostates, states (Work, AddHour), and their transitions. Furthermore, the associations between Clock and Hand (hourHand, minuteHand), as well as between ClockTower and Clock, are correctly reconstructed, indicating the decoder's ability to maintain entity relationships from textual input.

Despite these successes, several significant differences highlight areas for improvement:

- **Metamodeling Context Loss:** The original model includes architectural metadata such as the package 'ClockTower', the profile 'Mascaret', and the stereotype 'Entity', which are absent in the reconstruction. This suggests limitations in conveying or interpreting higher-level metamodeling structures. The absence of this metadata leads to significant differences in the model hierarchy, as these elements often serve as organizational containers that structure the model tree. Without them, the reconstructed model lacks the proper hierarchical organization of the original model.
- **Association-Attribute Ambiguity:** The original model contains attributes like clock association, hourHand association, and minuteHand association, which likely represent navigable association ends in the XMI serialization. This dual encoding of associations—both as UML Association elements and as class-level attributes—introduces semantic ambiguity. This ambiguity is not reconstructed by the decoder, thereby partially reducing the measured quality of the transformation.
- **Behavioral Modeling Inconsistencies:** The reconstructed model introduces structural elements not present in the original, such as the composite state 'top' and action elements within the state machine. This contrasts with the original model's use of Region for structural decomposition in the state machine, indicating potential over-specification in the reconstruction process.

Implications for Model Transformation: These findings have important implications for the encoder-decoder approach to model-text-model transformation. While the system demonstrates strong capabilities in handling explicit, well-defined model components, it faces challenges with:

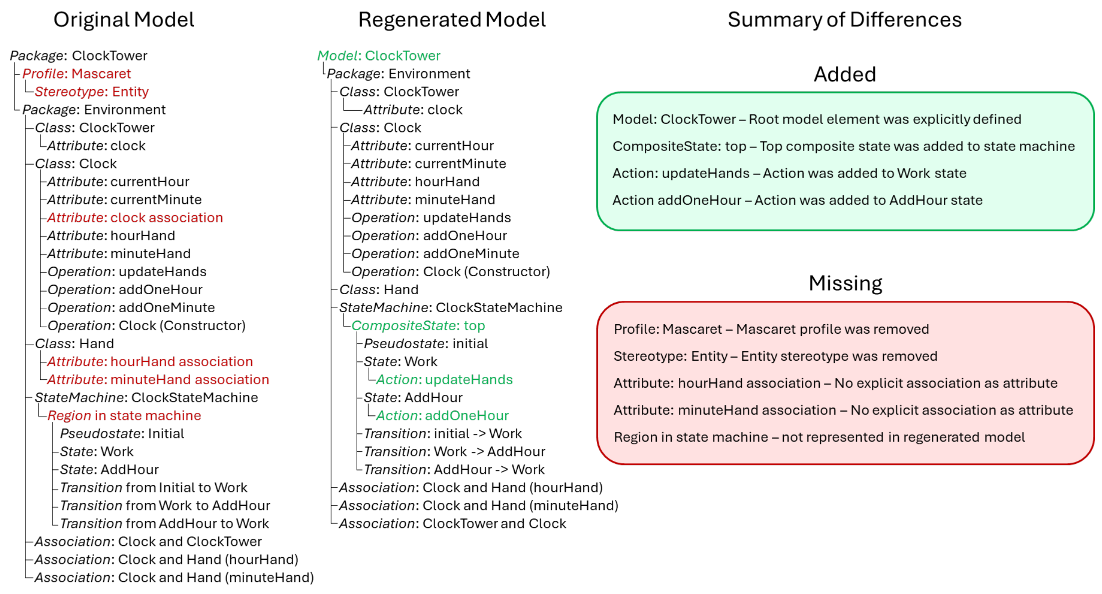

Original Model
Package: ClockTower
Profile: Mascaret
Stereotype: Entity
Package: Environment
Class: ClockTower
Attribute: clock
Class: Clock
Attribute: currentHour
Attribute: currentMinute
Attribute: clock association
Attribute: hourHand
Attribute: minuteHand
Operation: updateHands
Operation: addOneHour
Operation: addOneMinute
Operation: Clock (Constructor)
Class: Hand
Attribute: hourHand association
Attribute: minuteHand association
StateMachine: ClockStateMachine
Region in state machine
Pseudostate: Initial
State: Work
State: AddHour
Transition from Initial to Work
Transition from Work to AddHour
Transition from AddHour to Work
Association: Clock and ClockTower
Association: Clock and Hand (hourHand)
Association: Clock and Hand (minuteHand)
Regenerated Model
Model: ClockTower
Package: Environment
Class: ClockTower
Attribute: clock
Class: Clock
Attribute: currentHour
Attribute: currentMinute
Attribute: hourHand
Attribute: minuteHand
Operation: updateHands
Operation: addOneHour
Operation: addOneMinute
Operation: Clock (Constructor)
Class: Hand
StateMachine: ClockStateMachine
CompositeState: top
Pseudostate: initial
State: Work
Action: updateHands
State: AddHour
Action: addOneHour
Transition: initial -> Work
Transition: Work -> AddHour
Transition: AddHour -> Work
Association: Clock and Hand (hourHand)
Association: Clock and Hand (minuteHand)
Association: ClockTower and Clock
Summary of Differences
Added
Model: ClockTower – Root model element was explicitly defined
CompositeState: top – Top composite state was added to state machine
Action: updateHands – Action was added to Work state
Action addOneHour – Action was added to AddHour state
Missing
Profile: Mascaret – Mascaret profile was removed
Stereotype: Entity – Entity stereotype was removed
Attribute: hourHand association – No explicit association as attribute
Attribute: minuteHand association – No explicit association as attribute
Region in state machine – not represented in regenerated model

Figure 3. Structural comparison between original and reconstructed model trees.

- Processing tool-specific implementation details such as serialization of association ends
- Consistently reproducing structural hierarchies
- Preserving semantic roles of elements
- Handling metamodel constructs

Since the similarity score is strongly impacted by whether the model hierarchy is preserved in the reconstruction, the observed problems in maintaining the model hierarchy are critical for the measured transformation quality, leading to low similarity scores especially for larger models, where the model hierarchy is more complex and less explicitly represented in the textual description.

V. Discussion and Conclusion

This paper investigated the potential of Large Language Models (LLMs) for bidirectional transformation between UML models and natural language descriptions. We proposed an encoder-decoder framework that converts UML models in XMI format into structured text descriptions and back, evaluated its performance on a large benchmark dataset, and analyzed the quality of transformations using graph-based similarity metrics. Our results show promising accuracy for small models but reveal significant challenges with larger models and structural variations. In this discussion, we first examine our core findings and their limitations, then discuss technical challenges and potential solutions, and finally outline promising directions for future research.

A. Core Findings and Limitations

Our evaluation reveals several critical insights about the capabilities and limitations of LLM-based model transformation. The most striking finding is the strong correlation between model size and transformation accuracy. As shown in Sec. IV, models with fewer than 100 elements can often be transformed with satisfactory accuracy, while larger models consistently yield poor results. This limitation is particularly relevant given that most real-world UML models in our benchmark dataset contain between 100 and 1000 elements.

A more nuanced limitation emerges when examining the nature of transformation errors. Our microscopic analysis of individual transformations (Sec. IV) reveals that the current evaluation metrics may not fully capture the semantic quality of model reconstructions. For instance, the clock tower example demonstrates how models can maintain most of their elements and relationships while receiving low similarity scores due to differences in structural hierarchy. This suggests that our current evaluation approach, while providing valuable quantitative insights, may not align perfectly with human judgment of transformation quality.

These findings have important implications for the practical application of LLM-based model transformation. While the approach shows promise for smaller models, its current limitations make it unsuitable for direct application to large-scale industrial models.

B. Technical Challenges and Solutions

The effectiveness of our approach heavily depends on the quality and specificity of the prompts used for model transformation. Our experience revealed several important considerations regarding prompt engineering and its limitations:

- **Domain-Specific Adaptation**: Prompts need to be carefully tailored to different UML diagram types (e.g., activity vs. class diagrams). This requires pre-analysis of the model structure and iterative refinement of prompts to capture the specific characteristics of each diagram type.
- **Version-Specific Requirements**: The prompts must be specifically aligned with the UML/XMI version being used.
- **Context Length Limitations**: The effectiveness of the transformation is constrained by length-bias of the LLM.

- **Iterative Refinement**: The prompt engineering process requires multiple iterations to account for the nuances of different diagram types and to achieve satisfactory transformation quality.

To address these challenges, several potential solutions emerge. First, working directly with raw XMI files introduces substantial challenges due to their verbosity, complexity, and tool-specific syntax variations. One potential solution is to introduce a structured Intermediate Representation (IR) for UML models. This IR would serve as a normalized, simplified graph-based format that abstracts from XMI-specific details while preserving essential model semantics. The IR could be represented in JSON or a domain-specific graph format, with a unified schema for elements (type, name, attributes, relationships).

Benefits of using an IR include:

- Decoupling parsing logic from evaluation and transformation processes.
- Enabling modular extension of the pipeline (e.g., for code generation or visualization).

C. Future Directions

Based on our findings, several promising directions for future research emerge:

- **Improved Evaluation Metrics**: Future work should explore hybrid evaluation strategies that combine multiple metrics to better capture both structural and semantic similarity. This could include developing metrics that are more robust to structural variations while maintaining semantic equivalence.
- **Enhanced LLM Capabilities**: Experimenting with different LLMs, especially LLMs with larger context windows and better capabilities in generating longer texts, could potentially prevent the loss of information in transforming larger models. The encoder and the decoder can even be represented by different LLMs, to leverage the strengths of each model.
- **Transformation Pipeline Optimization**: The robustness of model comparison frameworks will depend not only on the choice of similarity metric but on the design of the entire transformation pipeline. Future work should focus on optimizing each component of this pipeline.
- **Integration with Development Workflows**: Embedding the encoder-decoder framework into model-to-code or model-to-pseudocode pipelines opens up new possibilities for evaluating semantic fidelity. By generating code artifacts—such as class structures, method signatures, or API interfaces—from a UML model and then reconstructing the model from this code, deeper semantic mismatches can be revealed. These include, for example, incorrect class responsibilities, missing abstractions, or misinterpreted relationships that may not be evident from model-level similarity metrics alone. Such a round-trip transformation (model $\rightarrow$ code $\rightarrow$ model) can serve as a stricter benchmark, helping to assess whether the intended meaning of the original model is preserved across abstraction levels.

These future directions aim to address the current limitations while building on the promising aspects of LLM-based model transformation. By focusing on both technical improvements and practical applications, we can work towards making model transformation more reliable and useful in real-world software engineering contexts.

REFERENCES

[1] K. Kshirsagar, P. Shah, and R. Sekhar, "Model Based Design in Industrial Automation," in *2022 6th International Conference On Computing, Communication, Control And Automation (ICCUBEA*. IEEE, 2022, pp. 1–6.

[2] S. Friedenthal, A. Moore, and R. Steiner, "Model-Based Systems Engineering," in *A Practical Guide to SysML*. Elsevier, 2012, pp. 15–27.

[3] S. Bashir, "Towards AI-centric Requirements Engineering for Industrial Systems," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, A. Roychoudhury, A. Paiva, R. Abreu, and M. Storey, Eds. New York, NY, USA: ACM, 2024, pp. 242–246.

[4] L. Guerreiro Azevedo, R. de Almeida Rodrigues, and K. Revoredo, "BPMN Model and Text Instructions Automatic Synchronization," in *Proceedings of the 20th International Conference on Enterprise Information Systems*. SCITEPRESS - Science and Technology Publications, 2018, pp. 484–491.

[5] M. Ballard, R. Peak, S. Cimtalay, and D. Mavris, "Bidirectional Text-to-Model Element Requirement Transformation," in *2020 IEEE Aerospace Conference*. IEEE, 2020, pp. 1–14.

[6] Thomas Freytag, Benjamin Kanzler, Nils Leger, and Daniel Semling, "NLP as a Service: An API to Convert between Process Models and Natural Language Text," *Proceedings of the Demonstration & Resources Track, Best BPM Dissertation Award, and Doctoral Consortium at BPM*, 2021. [Online]. Available: https://ceur-ws.org/Vol-2973/paper_279.pdf

[7] D. de Bari, G. Garaccione, R. Coppola, M. Torchiano, and L. Ardito, "Evaluating Large Language Models in Exercises of UML Class Diagram Modeling," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, X. Franch, M. Daneva, S. Martínez-Fernández, and L. Quaranta, Eds. New York, NY, USA: ACM, 2024, pp. 393–399.

[8] A. Ferrari, S. Abualhaija, and C. Arora, "Model Generation with LLMs: From Requirements to UML Sequence Diagrams," in *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*. IEEE, 2024, pp. 291–300.

[9] F. Härer, "Conceptual Model Interpreter for Large Language Models."

[10] J. Köpke and A. Safan, "Introducing the BPMN-Chatbot for Efficient LLM-Based Process Modeling," *Proceedings ofthe Best BPM Dissertation Award, Doctoral Consortium, and Demonstrations & Resources Forum*, 2024.

[11] 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*, ser. Lecture Notes in Business Information Processing, H. van der Aa, D. Bork, R. Schmidt, and A. Sturm, Eds. Cham: Springer Nature Switzerland, 2024, vol. 511, pp. 229–244.

[12] L. Naimi, E. M. Bouziane, A. Jakimi, R. Saadane, and A. Chehri, "Automating Software Documentation: Employing LLMs for Precise Use Case Description," *Procedia Computer Science*, vol. 246, pp. 1346–1354, 2024.

[13] J. Neuberger, L. Ackermann, H. van der Aa, and S. Jablonski, "A Universal Prompting Strategy for Extracting Process Model Information from Natural Language Text Using Large Language Models," in *Conceptual Modeling*, ser. Lecture Notes in Computer Science, W. Maass, H. Han, H. Yasar, and N. Multari, Eds. Cham: Springer Nature Switzerland, 2025, vol. 15238, pp. 38–55.

[14] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.

[15] A. Salim, S. Shiju, and S. Sumitra, "Graph kernels based on optimal node assignment," *Knowledge-Based Systems*, vol. 244, p. 108519, 2022.

[16] A. Hagberg and D. Conway, "Networkx: Network analysis with python," *URL: https://networkx. github. io*, pp. 1–48, 2020.

[17] G. Robles, T. Ho-Quang, R. Hebig, M. R. Chaudron, and M. A. Fernandez, "An Extensive Dataset of UML Models in GitHub," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 519–522.