Universität Rostock

Traditio et Innovatio

Enhancing Model-to-Text Transformations with LLMs

A Comparative Study of Existing Tools and Opportunities for AI-driven Improvements

Last name:	Aiken, Hosseinzadehattar
First name:	Bradley, Rowshanak
Stu-Nr.:	224201878, 2241000092
Major:	CSI
E-Mail:	{Bradley.Aiken, rowshanak.hosseinzadehattar} @uni-rostock.de
Professor:	Prof. Dr.-Ing. Kurt Erich Dirk Sandkuhl
Faculty:	Informatik
Module:	Anwendungen der Unternehmensmodellierung (Applications of Enterprise Modeling)
Semester:	SS2025
Abgabedatum:	10.09.2025

Scientific Report

Universität Rostock

Inhaltsverzeichnis

1 Abstract

Conceptual models—such as business process models or enterprise models—are essential tools for capturing and communicating the structure, behavior, or transformation of organizations. However, while these models use standardized notations that are machine-readable, their visual representation often remains inaccessible to stakeholders unfamiliar with modeling languages. To address this gap, Model-to-Text (M2T) transformation offers a bridge to render models into natural language, improving accessibility and collaboration.

Our work explores existing M2T techniques, particularly from the field of model-driven engineering (MDE), and investigates whether and how such approaches can be adapted to the domain of enterprise modeling. We reviewed model transformation software from a state-of-the-art survey (10) with traditional M2T approach implementations and evaluated their functionality. In addition, we explore the role of Large Language Models (LLMs) in enhancing or complementing M2T pipelines. We report the challenges encountered, compare the output quality of the tested methods, and discuss the potential of LLM-supported M2T transformation. Our findings highlight both the strengths and current limitations of LLM-supported M2T systems in producing accurate and readable natural language descriptions of enterprise models.

2 Introduction

Enterprise modeling describes the structure, processes, and information flows of organizations. Whether for documenting existing operations or planning future change, conceptual models play a key role in aligning stakeholders. Yet, as helpful as process models are for experts, they often pose challenges for non-technical audiences(12).

Natural language can bridge this gap in stakeholder comprehension. If a model could be automatically transformed into natural language that explains its logic, sequence, and purpose, it would become more accessible—especially for managers, customers, or employees who are not trained in enterprise modelling.

Transforming models into textual representations is not a new concept; M2T transformation already serves many purposes in MDE, such as generating source code, documentation, or configuration files(17). There are many widely-implemented technical applications of M2T, but in this context, M2T transformations are needed to produce readable, natural text to support communication for stakeholders.

3 Motivation

We started this project with a simple but important question: How can we make models more accessible to non-experts?

While enterprise modeling tools allow us to create precise and semantically rich representations, their usefulness is often limited to those with formal training. Most stakeholders, however, want to know: *"What does this model actually mean for my job?"*—and they want the answer in plain language. This motivated us to look into M2T transformation for natural language.

We also saw a second opportunity: Can modern LLMs help improve M2T transformations, especially when combined with traditional methods?

The idea is to enhance traditional M2T transformation methods. LLMs like GPT-4 are remarkably good at generating readable text—but are they trustworthy? Can they handle complex models without hallucinating content? What if we combined their language fluency with the structural reliability of template-based or visitor-based tools?

4 Background

M2T transformations serve many important functions in MDE(10)(12). These functions include generation of documentation and executable code, serialization to enable the interchange of models between languages or formats, and model validation(17). Furthermore, M2T transformations are foundational to supporting software development. Modeling tools such as Xtend or Modelio are capable of transforming visual models into code in different programming languages. Additionally, many modelling languages such as BPMN and UML, which are typically displayed visually, can be easily transformed into textual formats to be exported or validated. However, while traditional M2T transformation methods have been developed that are capable of producing code or even basic documentation, there has been a gap in research in transforming models into natural language(15).

A reliable method to generate natural language text from process models would facilitate important parts of an MDE-driven process. M2T transformations for natural language would make models more accessible to stakeholders who do not have high levels of expertise in process modelling(15), leading to improved communication and validation of models. Klievtsova et al. identified two cases where this type of M2T transformation would be useful during a requirements engineering process: during requirements elicitation, where stakeholders analyze an AS-IS model of an existing system, and during requirements specification and analysis and creating the TO-BE model. M2T transformation could create natural language documentation of the current and updated system models to be validated and better understood by stakeholders(12).

A survey by Kahani et al. analyzed the state of available modeling tools and their capabilities of model transformation, including Model-to-Model and Text-to-Model transformations, which are outside the scope of this study(10). While there are many tools that provide M2T functionality, the ability to generate text that resembled natural language process descriptions was either not well-developed, not present, or not accessible without a commercial license in any of the over 60 surveyed applications.

There are three main traditional M2T methods which can be used to generate textual artifacts such as code or documentation(10). A visitor-based approach traverses a tree-based representation of a model and generates text for each visited node in the tree, which could represent either an element in the model or a relationship between elements. A template-based approach uses static text templates to generate documentation or basic descriptions from model elements, usually supported by a templating language. Finally, a hybrid approach combines both of these methods, traversing a tree-based model representation while using templates to generate text. This is the most suitable traditional M2T method for generating natural language(14).

A traditional hybrid approach, proposed by Leopold et al. in 2014, converts BPMN models to Refined Process Structure Trees (RPSTs) to generate textual process descriptions(14). This approach traverses through the RPSTs, collecting information about each task or connector to structure the output text. Additionally, semantic information is collected about who or what

performs each task, what actions are taken during tasks, and what the task acts on. This is stored in a structure called a deep syntactic tree, which maps grammatical roles to each node to generate grammatically correct English text. The evaluation of this approach showed that each element of the BPMN process was able to be adequately described with a grammatically correct sentence, but the final text was not always stylistically favorable or contained many repeated and redundant phrases.

4.1 Previous Investigation into LLM-Driven M2T Transformations

Recent progress in natural language processing and especially in LLMs has made it easy and accessible to generate natural-sounding text and could provide modellers with a powerful tool to extract information from or transform models into other formats. Klievtsova et al. investigated whether LLMs, such as ChatGPT and Llama, could be suitable for performing M2T transformations and compared the results from LLMs to traditional M2T transformation approaches(12).

The study compared the output texts from various ChatGPT and Llama model versions and from a traditional M2T transformation approach based on the proposal by Leopold et al.(14) However, because the source code for that approach is not available, Klievtsova et al. developed a similar hybrid approach, but did not implement the deep syntactic trees. The traditional approach implemented in the study instead relied more heavily on templating.

Key performance indicators that were examined included the amount of words and sentences in each text, the ease of readability, and a similarity score to a human-annotated process description used as an ideal control. The study used models from the PET dataset, an annotated dataset for process extraction from tasks written in natural language(11). The results indicated that LLMs are more than capable of producing textual descriptions that are highly similar to human-written descriptions and just as easily readable. The hybrid approach produced text that was similar to the human-written annotation as well, but produced much more irrelevant text, as the templates describe every part of the model with a fixed phrase.

While the description generated from the hybrid approach contained redundant text, the study found that LLMs were able to extract tasks from the traditionally-generated text more reliably than from the LLM-generated text. Additionally, this study did not specifically examine accuracy in the LLM-generated output text. Furthermore, because LLMs were not able to extract all tasks from the LLM-generated text as often as they were with the traditionally-generated text, this could indicate that the LLM-generated text may occasionally omit or obfuscate parts of a model when describing it. Given the tendency of LLMs to hallucinate, thoroughly examining the accuracy of text generated by LLMs is paramount to ensuring that they can be useful(9), especially when extracting information from artifacts used to capture high degrees of specificity. The ability to automatically detect potential errors in text generated based off a model would help make LLM-driven M2T transformations reliable.

5 Analysis of Traditional M2T Tools

To determine whether LLM-driven M2T transformations could be useful, and when it could be appropriate to use them, current software capabilities needed to be examined to see if there were areas for improvement. Several tools from the survey on model transformation software(10) were

analyzed specifically for their M2T capabilities. Although there is a lot of software that claim M2T functions, in almost all cases this only applies to procedural code generation or exporting models in various fixed notations without utilizing any natural language processing. Each tool was analyzed with the goal to implement an M2T transformation on enterprise models. In practice, this revealed several technical and conceptual challenges.

5.1 ActifSource and Xtend

ActifSource(1) and Xtend(6) are both extensions for the Eclipse IDE that bring different modelling capabilities. ActifSource features native modelling support directly in the IDE with multiple modelling languages, including UML. ActifSource is capable of generating arbitrary code or text from a model through a templating system. While this feature can be used to generate natural language descriptions, according to its documentation, it is most apt for generating code to a target programming language. Xtend has similar model transformation features, but is a broader extension to the Java language and does not include modelling features itself. Xtend is designed to work with models and is capable of transforming or extracting structured information from them, but also falls short of functionality tailored for natural language generation.

5.2 Modelio

Modelio(5) is a GUI-based modelling tool for many commonly used modelling languages, such as UML, BPMN, ArchiMate, and TOGAF. Similarly to previous tools, Modelio is capable of generating text from various model formats to various target programming languages. In addition to a GUI, Modelio offers a scripting console where models can be directly read or manipulated. This can be used to generate other arbitrary texts, though this would require scripts that may be specifically tailored to the model in question to yield results that are more pleasant to human readers.

Various extensions are available for Modelio, however, many require a commercial license. The Document Publisher extension is capable of aiding in the production of high quality documentation with the assistance of templates, and is able to format tables, diagrams, and glossaries of model concepts. While this appeared to be an interesting approach, without the license for the Document Publisher extension, this was not able to be tested. Based on the documentation, this would be a traditional hybrid approach with highly customizable templates.

5.3 Traditional Hybrid Approach by Klievtsova et al.

This approach was developed after the modelling software survey was conducted, and is based off of another traditional hybrid M2T transformation approach(12)(14). A Ruby script converts process models, formatted in BPMN, Graphviz, or Mermaid.js, to RPST trees and then traverses each node in the tree. Prewritten templates are concatenated with text from the nodes in the tree to build stale, but accurate natural language descriptions(3). Nodes consist of all events and gates, and labelled relationships are described as well. While the results are not overwhelmingly impressive, they provide a concrete, testable example of what current, traditional M2T transformation methods allow for in transforming process models to text.

The approach by Leopold et al. incorporated deep syntactic trees, which appear capable of producing more dynamic English language descriptions than this approach, but the implementation is not publically available for testing.

The results from both approaches, however, reveal areas that could be improved with accurate LLM-based natural language processing. Namely, the structure of such generated texts are stale and do not read well, as they describe every element of the model, rather than just the relevant information about the process itself. Transitions and sequences are included in the templates of both approaches, but do not always chain together well. For example, if there are many tasks that get executed in sequence, each task gets its own entire sentence starting with "Then".

5.4 ATOM3

5.4.1 Motivation for Selecting AToM^3

AToM^3 is a tool designed to generate modelling environments through the combined use of meta-modelling and graph grammars (13). Its core idea is that “everything is a model”—from the user interface to the model transformation logic. The tool allows formalism designers to define their own domain-specific modelling languages (DSMLs) using an extended entity-relationship meta-formalism. These formalisms are then interpreted and used to generate visual modelling environments. All models are internally represented as graphs, and manipulations—including simulation, optimization, or code generation—are expressed as graph grammars.

The motivation behind AToM^3, as presented by de Lara et al. (13), stems from the challenge of modelling complex systems that span multiple domains and levels of abstraction. In such multi-paradigm systems, no single formalism suffices. Instead of building a custom tool for each formalism, AToM^3 offers a meta-modelling approach, significantly reducing development time and ensuring consistency between formalisms via transformation grammars.

This strong conceptual foundation, along with its full implementation in Python, made AToM^3 an attractive choice for this project. The fact that the tool is programmable, extendable, and free to use matched our criteria for a customizable M2T pipeline. In particular, we discovered an actively maintained repository (`LSMASOMM`)[1] that provided multiple meta-models, grammar examples, and a working setup of AToM^3. This repository offered an ideal base to experiment with generating natural language from structured visual models in a highly configurable environment.

Figure 1 shows the graphical user interface of AToM^3, while the LSMASOMM_Meta.py is loaded. Figure 2 shows the main menu, where different modelling formalisms and transformation grammars can be defined and applied.

5.4.2 Implementation Experience

The initial excitement was quickly tempered by the reality of working with legacy infrastructure. AToM^3 is built on Python 2.7, a language version that is no longer officially supported. Moreover, many dependencies listed in the project had been deprecated or renamed. To overcome this, a custom Docker environment was created to isolate and restore the functionality of the application.

[1] https://github.com/AILab-FOI/LSMASOMM

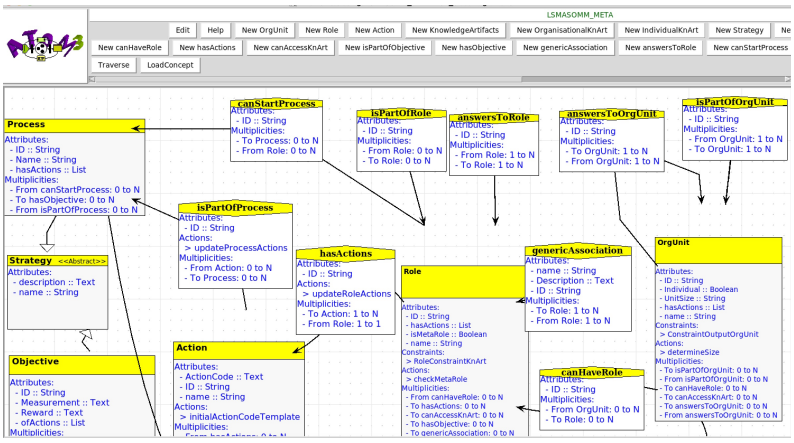

Abbildung 1: User interface of AToM^3 loaded with a meta-model and graph grammar

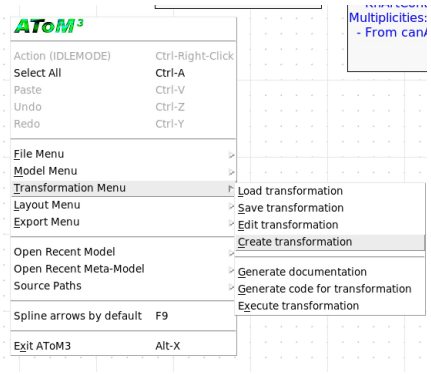

Abbildung 2: Main Menu of AToM^3

This included installing precise versions of Python packages and adapting the environment to run on macOS via X11 forwarding.

5.4.3 Challenges and Decision to Discontinue

Despite the successful technical setup, the tool posed significant usability and maintenance issues. The interface was slow, error-prone, and offered limited feedback when transformations failed. Moreover, the documentation was sparse, and many of the included transformation rules relied on outdated conventions that were difficult to adapt for modern enterprise modelling use cases.

Given these limitations, ultimately the right choice was to discontinue work with AToM^3.

6 Methodology

To determine the viability of LLM-driven M2T transformations, different transformation methods were compared against each other. To test each method, a dataset of example business process models were created. 25 BPMN models were created to be tested against, covering a wide variety of processes and diversity in complexity. The BPMN models can be found in `data/bpmn_models`, the amounts of edges, tasks, and gateways for each BPMN model are stored in `graph_stats.csv` along with the graph density and cyclomatic complexity.

For each BPMN model, additional artifacts were created to run the experiment. A natural

language description was manually written without LLM assistance. These are denoted as the Golden Descriptions, found in `data/golden_descriptions`, as they are ideal descriptions of the process model and act as a control against the M2T transformation method outputs. Additionally, a textual representation of each BPMN model was extracted into the Mermaid.js graphing language, found in `data/mermaid_models_fixed`. This is done to save on the amount of tokens required when interacting with LLMs, as BPMN models contain metadata that is irrelevant to the structure of the process model and could be too long to feed into a commercial LLM such as GPT.

Both a traditional and LLM-driven M2T transformation methods were run on each process model. The traditional hybrid M2T transformation method from the experiment by Klievtsova et al. was adapted. This involved breaking down the process model into a tree, traversing through the tree, and using prewritten templates to generate a natural language descriptions of the tasks and their structure. A Ruby script was written to perform the traditional M2T transformations, with the results stored in `data/traditional_m2t_output`. For the LLM-driven approach, two LLMs were chosen to perform the M2T transformations individually, GPT-4o-mini by OpenAI and Llama 3.1 with 8 billion parameters(4). Using more than one LLM provides some diversity in assessing how suitable LLMs are for performing M2T transformations.

Running the process models through the LLMs required prompts. Here, the Mermaid.js representations reduced the amount of tokens in the prompt enough so that even large process models could be tested against. Three prompts were used against the LLMs for each process model. This was done to provide the LLMs with different levels of specificity in the instructions and to control for any issues that could arise if an LLM poorly interpreted one of the prompts. Prompt 1 is the simplest and most open-ended, only asking the LLM to convert the process model into a textural process description. Prompt 2 aims to reduce the amount of unnecessary technical descriptors in the output, such as mentions of gateways or start and end events. Prompt 3 is the same as Prompt 2, but it also includes the output from the traditional M2T transformation method to see if the LLM could be used to improve the traditional method. Each process model was used with each prompt and both LLMs. The results are stored in `data/llm_m2t_output` where the filename includes the name of the process model, the LLM, and the prompt.

In an effort to improve or ensure the correctness of the LLM outputs, a round of self-correction was run on each of the LLM M2T outputs. For each LLM-generated textural process description, the respective LLM was prompted to extract all the tasks from its own description. The extracted tasks were stored in a `task_extraction.txt` file for each prompt. Then, the LLM was given the original Mermaid.js process model and the extracted tasks and was prompted to determine if there were any conflicts, which would indicate if all tasks were included and no new tasks were hallucinated by the LLM. If the LLM detected conflicts, then the LLM was prompted to fix the textual process description it had previously generated; both the original Mermaid.js process model and the textual process description it had previously generated were included as context in the prompt. The result of this self-correction is stored in a `conflict_fix.txt` file. This self-correction is done twice so that if the LLM still cannot extract the same amount of tasks from its own process description as in the process model, then it is prompted to further correct its original description. Because this process could potentially loop forever, the self-correction is only performed twice at maximum, as the LLM might never be able to produce a process description

that it does not find conflicts in with the original Mermaid.js process model.

After generating the outputs from each M2T transformation method with each LLM and prompt on each process model, they were finally compared. Some metrics could be calculated automatically and others were rated manually by reading the generated process descriptions and assessing how well the LLM followed the instructions in the prompt, how accurate the process description is to the original Mermaid.js process model, and how easy to read and well-formatted the description is. All metrics for each LLM-generated description are stored in `LLM-Generated M2T Results Analysis.xlsx`.

6.1 Implementation

The process models used in this study were both gathered from previous studies and manually created in BPMN. Eight process models were gathered from other studies, mostly from the PET dataset used by Klievtsova et al. and the other 17 models were manually created. The `transform_models.script` Ruby script was used to convert the BPMN models into Mermaid.js models for use in the prompts to the LLMs. This script uses the CPEE framework(2), also used by Klievtsova et al. for their traditional M2T model transformations and model conversions.

A pipeline was implemented in `main.ipynb` to generate all M2T transformation outputs from each process model. The GPT-4o-mini model was called through OpenAI's API and the Llama 3.1 model was run locally. The pipeline calculated all automated metrics for each process model in addition to generating the `graph_stats.csv` with the statistics on how complex each process model is. As there was no reliable method to automatically check the instruction following, correctness, and writing quality metrics, these were rated manually after running the pipeline.

6.2 Evaluation

The evaluation of our generated M2T descriptions was conducted using a combination of quantitative metrics and visualization techniques, automatic metrics, manual reviews, and conflict analysis logic. The analyses measured the performance and quality of the generated texts across different LLM models, prompt variants, and text types.

Dataset Preparation. We analyzed the output data from the implemented pipeline in Python stored in the CSV file. Rows corresponding to the "Golden Description" reference texts were excluded to focus on comparing traditional and LLM-based M2T outputs. Column names and string values were cleaned by removing any leading or trailing whitespace to ensure consistent matching and prevent lookup errors during analysis, and relevant numeric fields were coerced into numeric format using `pandas` (16).

Metrics. Two automatic metrics were used:

- **Score** – a combined text similarity score based on trace similarity, calculated using the following weighted formula:

$$\text{Score} = 0.7 \cdot \text{TS-C} + 0.3 \cdot \text{TS-NC}$$

Where TS-NC was computed using a non-contextual Term Frequency-Inverse Document Frequency (TS-NC) and a cosine similarity approach, capturing lexical overlap without considering control-flow semantics, whereas contextual Term Frequency (TS-C) relied on contextual embeddings from the OpenAI text-embedding-ada-002 model, enabling semantic comparison that is sensitive to task order and process structure. This measurement was taken by comparing the LLM-generated model descriptions and the Golden descriptions.

- **FRE (Flesch Reading Ease)** – indicates how easy the text is to read.

Manual Review. Each LLM-generated text was manually reviewed based on three dimensions, rated on a scale from 1 to 5 where 1 is perfect and 5 represents a result that does not align with the prompt or is not useful.

- **Instruction Following** – how closely did the LLM follow the instructions from the prompt, i.e. generating a natural language description from a provided business process model given in Mermaid.js notation. 1 - Perfect, natural language text with no extra formatting. 2- Mostly natural language text with possible extra formatting or whitespace. 3 - A bulleted list or every task is its own line OR references model elements instead of describing the actual process. 4 - More of a chart than natural language. 5 - Not a natural language description of the model.
- **Correctness** – whether the natural language description accurate describes the provided Mermaid.js model. 1 - Perfectly describes the model entirely. 2 - Misinterprets a task name or a very minor detail. Overall correct. 3 - Makes a minor mistake that makes the description no longer fully correct. 4 - Makes a major mistake. 5 - Completely untruthful; does not describe the process model at all.
- **Writing Quality** – how clear, concise, and easy to follow the natural language description is. 1 - Perfect and easy to read. 2 - Minor stylistic issues, overall good. 3 - OK, but either too long or too short. 4 - Hard to read, not natural at all. 5 - Impossible to read or no useful information.

These human-assigned quality ratings were then visualized by prompt and LLM to analyze overall trends in generation performance.

Conflict Detection. To ensure alignment between model outputs and the tasks in the original process models, we implemented an LLM-based conflict detection procedure. First, tasks were extracted from each generated text via prompting:

```
Return a numbered list of activities (each 3-5 words maximum) one by one
    without any additional information. Respond with the following
   format:
"Number of tasks: <number of tasks>"
```

This list was then compared against the original description using a second LLM prompt:

```
Please fix the process description to resolve these conflicts. Return
   only the fixed process description.
```

The outputs were analyzed by the LLM to identify any semantic mismatches or missing tasks. If no conflicts were found, the conflict-flag was set to 0; if conflicts were present, it was set to 1.

Self-Correction. If a conflict was detected, a follow-up prompt was issued to the LLM to revise the generated text and resolve the inconsistencies. The revised output was then re-evaluated for correctness and conflicts. This iterative self-correction logic enabled a second generation round, classified under the type "LLM-Generated Self-Corrected". Metrics for both original and corrected versions were stored and compared.

Integration with Structural Complexity. To investigate correlations between model structure and textual correctness, structural features (e.g., Density, Cyclomatic Complexity) were joined from a secondary file (`graph_stats.csv`). A scatter plot and regression analysis were used to visualize this relationship.

Tools. All evaluation logic was implemented in Python using `pandas` (16), `NumPy` (7), `Matplotlib` (8), and `Seaborn` (18).

7 Results

Traditional vs LLM-Generated Original. The LLM-generated original descriptions achieved a higher score ($M = 0.80$) compared to the traditional model-to-text outputs ($M = 0.66$), as shown in Figure 3. However, traditional outputs had higher readability based on the Flesch Reading Ease (FRE), indicating simpler sentence structures and vocabulary (Figure 4).

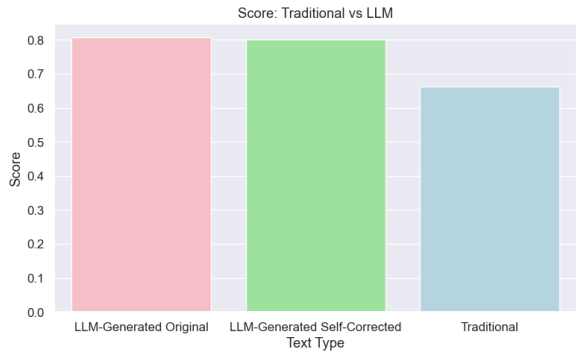

Abbildung 3: Average Score: Traditional vs. LLM-Generated Original

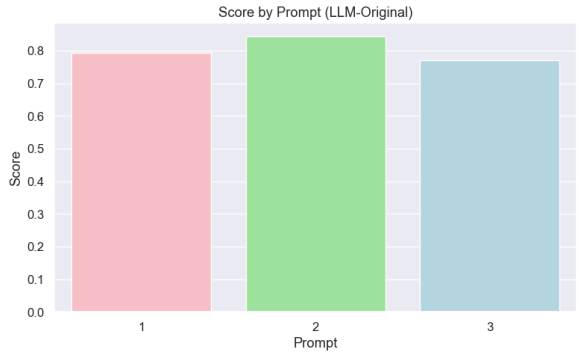

Abbildung 5: Average Score by Prompt (LLM-Generated Original)

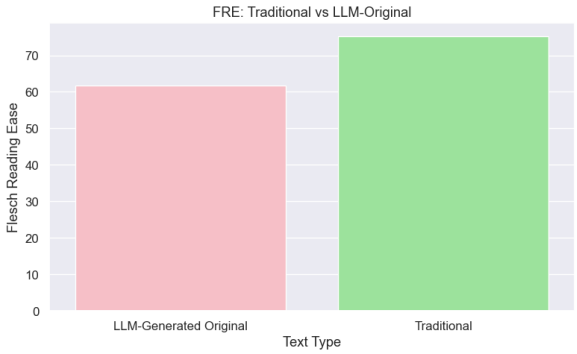

Abbildung 4: Readability (FRE): Traditional vs. LLM-Generated Original

Prompt Sensitivity in LLM Outputs. The prompts used in the experiment are below:

```
Prompt 1:
Read the following Mermaid.js model:
<model content>
Convert this model to a textual process description using simple
    language.

Prompt 2:
Read the following Mermaid.js model:
<model content>
Convert this model to a textual process description without mentioning
    types of the model elements (i.e., task, start event, end event,
    gateway, etc.). Return only the textual description. Do not use
    newlines or bullet points, unless paragraphs are needed.

Prompt 3:
Read the following Mermaid.js model:
<model content>
Convert this model to a textual process description without mentioning
    types of the model elements (i.e., task, start event, end event,
    gateway, etc.). Use the following traditional M2T transformation text
     as a reference:
<traditional M2T text>
The reference text uses a traditional M2T transformation method with
    minimal natural language processing and is therefore stale; this can
    be improved upon, use it only for reference. Return only the textual
    description. Do not use newlines or bullet points, unless paragraphs
    are needed.
```

Prompt 2 consistently yielded the highest average score among LLM-generated outputs (Figure 5), while Prompt 1 produced slightly more readable texts (Figure 6), however the differences in score and FRE were not very significant across all prompts.

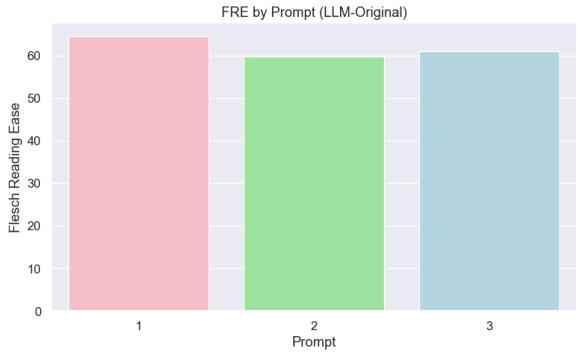

Abbildung 6: Flesch Reading Ease by Prompt (LLM-Generated Original)

Self-Correction. The results show that the self-correction did not significantly affect correctness. However, the relatively small changes suggest that better prompts should be used in the correction step.

Abbildung 7: Score Comparison: Original vs. Self-Corrected by Prompt

Impact of Model Complexity. To examine whether structural model characteristics influence the difficulty of generating accurate textual descriptions, we computed several complexity measures for each BPMN model:

- **Number of tasks:** counted as nodes in the process graph.
- **Number of gateways:** decision or merging points that increase control-flow branching.
- **Density:** calculated as $\frac{\#Edges}{\#Nodes \cdot (\#Nodes - 1)}$, measuring how densely connected the process graph is.
- **Cyclomatic Complexity:** computed as $\#Edges - \#Nodes + 2$, which quantifies the number of independent paths through the model. This metric reflects the minimum number of test cases needed to cover all control-flow variations and is widely used in software engineering.

Figure 9 shows a weak but positive correlation between cyclomatic complexity and the manual correctness score (lower = better). This indicates that more complex models tend to receive less accurate descriptions from LLMs.

To visualize this relationship, we fitted a linear regression line using `numpy.polyfit`, which calculates the best-fit line of the form $y = ax + b$ by minimizing the squared error between the predicted and actual values. Here, x represents cyclomatic complexity and y the correctness score. The resulting trend suggests a general pattern, although the variation among models remains considerable.

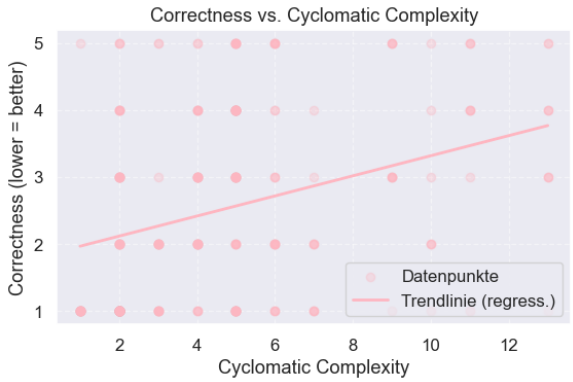

Abbildung 8: Correctness vs. Cyclomatic Complexity (lower = better)

The effect of complexity on readability — or what we referred to as writing quality in our manual review — was also examined. The effect turned out to be very weak, so it can be concluded that, fortunately, as the model becomes more complex, the readability of the LLM's output is not significantly affected.

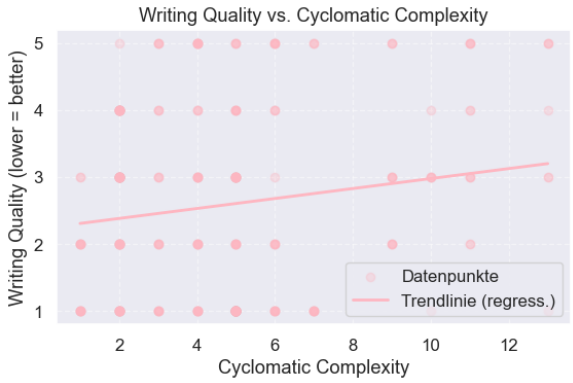

Abbildung 9: Readablity vs. Cyclomatic Complexity (lower = better)

Manual Quality Ratings. Human evaluations reveal nuanced differences across prompts (Figure 10) and LLMs (Figure 11). Prompt 2 scored highest in instruction following and writing quality, while Prompt 1 achieved the best correctness. Among LLMs, GPT-4o-mini slightly outperformed Llama 3.1 in correctness and instruction following, whereas writing quality was comparable.

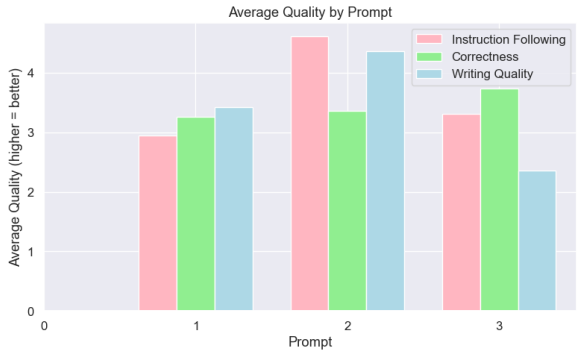

Abbildung 10: Manual Ratings by Prompt: Instruction Following, Correctness, Writing Quality

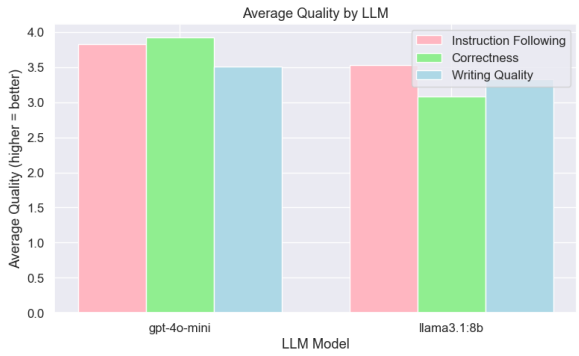

Abbildung 11: Manual Ratings by LLM: GPT-4o-mini vs. Llama 3.1

8 Discussion

Our findings show that LLM-generated descriptions outperform traditional M2T transformations in terms of writing quality, or resembling natural language. Moreover, the LLM-generated descriptions achieved higher non-contextual and contextual text similarity to the Golden Descriptions (see Fig.3). This suggests that LLMs, when guided by targeted revision prompts, can potentially produce more useful process descriptions, as the traditionally-generated descriptions, especially with larger models, were difficult to understand despite having a fixed template for each node in the process model.

Overall, Prompt 2 provided the best balance between instruction following, correctness, and writing quality. This is likely because Prompt 2 was designed to make the LLM avoid using modelling-specific language, so the generated process descriptions contained information more relevant to the process at hand. Prompt 1 received lower instruction following and writing quality scores because many of these descriptions contained more irrelevant information or references to specific model elements. Prompt 3 provided the highest accuracy, likely on the account of having both the Mermaid.js model and the traditionally-generated process description as context. However, this slight gain in accuracy came with a substantial decrease in writing quality. Because the traditionally-generated descriptions were so poorly written, the prompt provided the LLM a low quality starting point to improve from, and the writing quality suffered as a result (see Fig.10). Providing two representations of the original process model, both in Mermaid.js and as a traditionally-generated text description, appeared to allow the LLM to produce a more accurate

description.

In contrast to the LLM-generated descriptions, traditional outputs generally achieved higher Flesch Reading Ease scores (see Fig. 4). The reason for this lies in the simplified sentence structures produced by template-based methods. Nevertheless, this advantage is not decisive for traditional approaches, since in our use cases non-technical teams and stakeholders are typically well-educated adults who have no difficulty handling more complex texts as long as they remain clear. Additionally, in practice, reading and understanding the original process model only by reading the traditionally-generated descriptions was found to be very difficult. The more natural way of writing provided by the LLMs was decisively easier to follow.

Manual evaluation suggest that GPT-4o-mini slightly outperformed Llama 3.1 with 8 billion parameters in correctness and instruction following, though both models produced similar results in writing quality.

The analysis of structural complexity revealed a weak but positive correlation between cyclomatic complexity and correctness, where correctness had better scores when the cyclomatic complexity of the model decreased. Although the correlation is not consistent across all process model descriptions, the results indicate that more complex process models are generally harder for LLMs to describe accurately. This aligns with previous observations in MDE, where models with a greater number of independent control-flow paths tend to require more sophisticated handling to ensure faithful representation in text.(14)

Overall, these results suggest that LLMs can complement — and in some cases perform worse than — traditional M2T approaches. If the prompt does not explicitly instruct the model to avoid certain words, it may generate unnecessarily complex texts or add extra sentences.

8.1 Limitations and Future Work

While the results of this study demonstrate the potential of LLM-driven M2T transformations, several limitations affected the scope and depth of our evaluation.

First, the lack of relevant resources and standardized tools for natural language M2T transformation in the domain of enterprise modeling significantly limited our options. Many existing model transformation projects are either outdated or not maintained, with dependencies and documentation no longer functional for modern environments. In some cases, tools were so old that large portions of their functionality were unusable without substantial adaptation work.

Second, the absence of standardized languages and evaluation pipelines for M2T in natural language hindered comparability across approaches. Each selected tool required individual setup, manual adjustments, and, in some cases, custom data preprocessing, making a fully automated benchmark impractical.

Third, resource constraints played a notable role. The iterative nature of our evaluation meant that each execution of the complete pipeline required several hours. As a result, even minor changes in the experimental design or the addition of new metrics incurred substantial time costs, thereby limiting the number of feasible iterations.

For future work, several directions emerge. First, integrating with tools and deployment platforms for improved documentation, versioning, or even containerization of the pipeline for natural language M2T evaluation would make results more reproducible and comparable across studies. Second, adopting more efficient evaluation pipelines — for example, through parallel

processing or by leveraging reserved cloud resources (e.g., CPUs) — could significantly reduce iteration time and enable broader experimentation. Third, expanding the evaluation to cover additional model types and domains, as well as incorporating user studies with non-technical stakeholders, would provide a more holistic assessment of the practical usefulness of the generated text.

Another promising avenue is the exploration of more creative and targeted prompts, both for initial text generation and for the self-correction step. Future work could also introduce more precise and domain-specific metrics to better capture semantic fidelity and clarity, treating the trade-off between writing quality and accuracy as an optimization problem, analogous to how the F1-score balances recall and precision. Finally, while human ratings remain valuable, they represent a significant bottleneck in terms of time and resources; automated quality assessment techniques could supplement or even replace them.

Finally, our pipeline is designed as a modular framework in which new functions, metrics, and processing steps can easily be integrated. This openness allows for continuous improvement and adaptation to emerging evaluation methods, generation strategies, and model capabilities, ultimately enabling a flexible and extensible approach to LLM-based M2T transformation.

Literatur

[1] Actifsource. https://www.actifsource.com/, accessed: 2025-06-05

[2] Cpee. https://cpee.org/, accessed: 2025-06-05

[3] etm/prmptgen repository. https://github.com/etm/promptgen, accessed: 2025-05-20

[4] Llama 3.1 model. https://ollama.com/library/llama3.1, accessed: 2025-06-05

[5] Modelio. https://www.modeliosoft.com/en/, accessed: 2025-06-01

[6] Xtend. https://eclipse.dev/Xtext/xtend/index.html, accessed: 2025-06-05

[7] Harris, C.R., Millman, K.J., van der Walt, S.J., et al.: Array programming with numpy. Nature **585**(7825), 357–362 (2020)

[8] Hunter, J.D.: Matplotlib: A 2d graphics environment. Computing in Science & Engineering **9**(3), 90–95 (2007)

[9] ul Islam, S.O., Lauscher, A., Glavaš, G.: How much do llms hallucinate across languages? on multilingual estimation of llm hallucination in the wild (2025), https://arxiv.org/abs/2502.12769

[10] Kahani, N., Bagherzadeh, M., Cordy, J.R., Dingel, J., Varr
'o, D.: Survey and classification of model transformation tools. Software & Systems Modeling **18**(4), 2361–2397 (2019)

[11] Klievtsova, N., Benzin, J.V., Kampik, T., Mangler, J., Rinderle-Ma, S.: Conversational process modeling: Can generative ai empower domain experts in creating and redesigning process models? (2024), https://arxiv.org/abs/2304.11065

[12] Klievtsova, N., Mangler, J., Kampik, T., Rinderle-Ma, S.: Utilizing process models in the requirements engineering process through model2text transformation. In: 2024 IEEE 32nd International Requirements Engineering Conference (RE). pp. 205–215. IEEE (2024). https://doi.org/10.1109/RE59067.2024.00028, https://doi.org/10.1109/RE59067.2024.00028

[13] de Lara, J., Vangheluwe, H., Alfonseca, M.: Meta-modelling and graph grammars for multi-paradigm modelling in atom3. Software & Systems Modeling **3**(3), 194–209 (2004). https://doi.org/10.1007/s10270-003-0047-5

[14] Leopold, H., Mendling, J., Polyvyanyy, A.: Generating natural language texts from business process models. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) Advanced Information Systems Engineering. pp. 64–79. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[15] Leopold, H., Mendling, J., Polyvyanyy, A.: Supporting process model validation through natural language generation. IEEE Transactions on Software Engineering **40**(8), 818–840 (2014). https://doi.org/10.1109/TSE.2014.2327044

[16] McKinney, W.: Data structures for statistical computing in python. Proceedings of the 9th Python in Science Conference pp. 51–56 (2010)

[17] Rose, L.M., Matragkas, N., Kolovos, D.S., Paige, R.F.: A feature model for model-to-text transformation languages. In: 2012 4th International Workshop on Modeling in Software Engineering (MISE). pp. 57–63 (2012). https://doi.org/10.1109/MISE.2012.6226015

[18] Waskom, M.: Seaborn: Statistical data visualization. https://seaborn.pydata.org (2021)

9 Eidesstattliche Versicherung

Ich versichere eidesstattlich durch eigenhändige Unterschrift, dass ich die Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht und ist in gleicher oder ähnlicher Weise noch nicht als Studienleistung zur Anerkennung oder Bewertung vorgelegt worden. Ich weiß, dass bei Abgabe einer falschen Versicherung die Prüfung als nicht bestanden zu gelten hat.

Rostock

(Abgabedatum)	(Vollständige Unterschrift)