Model transformations using LLMs out-of-the-box: can accidental complexity be reduced?

Gabriel Kazai[1,*], Ronnie Agyeiwaa Osei[1], Alessio Bucaioni[1,*] and Antonio Cicchetti[1]

[1]*Mälardalen University, Box 883, 721 23 Västerås*

Abstract

Model-driven engineering envisions an enhancement of software engineering by promoting automation through model transformations. However, the effective use of model-driven tools often requires significant expertise due to their reliance on custom domain-specific languages for transformations. This expertise gap, combined with challenges like inadequate tool support and the need for additional training, has meant that model-driven engineering sometimes struggled to reduce, and might have even increased, accidental complexity. Addressing this problem, our work investigates the use of large language models, specifically ChatGPT-4, to reduce accidental complexity in model transformation processes within model-driven engineering. We conducted a systematic literature review and designed an experiment to explore ChatGPT-4's efficacy in performing model transformations out-of-the-box. Using a semi-automated pipeline, we applied ChatGPT-4 to 99 UML class diagram models, generating Java programs and comparing them with ground truth programs created by a state-of-the-art modelling tool. Our findings indicate a cumulative success rate of 94% after three iterations, with most generation errors being resolved during the process. However, complex models presented a significant challenge, with a cumulative success rate of only 17%.

Keywords

Model-driven engineering, model transformation, accidental complexity, large language models

1. Introduction

Model-Driven Engineering (MDE) advances software engineering by shifting the focus from coding to modelling [1]. It relies on two pillars: models, which are well-defined abstractions of reality [2], and model transformations, which automate model manipulation [3]. Models simplify complexity by emphasizing relevant details [4], while transformations enhance automation by synchronizing development stages, generating code, and enabling early validation [5]. MDE provides powerful abstractions and automation, with studies highlighting benefits such as reduced defects, increased productivity, and improved understandability [6]. However, its reliance on custom domain-specific languages for models, queries, and transformations demands significant expertise. This, coupled with inadequate tool support and training requirements, has paradoxically struggled to reduce *accidental complexity* despite reducing inherent software complexity [6, 7]. Low-/no-code development builds on MDE principles to mitigate these challenges by offering pre-made components [8]. While this approach reduces accidental complexity, it still requires learning platform-specific tools and constrains solutions to predefined component compositions [9].

Large Language Models (LLMs) are powerful tools with broad applications in software engineering, spanning from requirements management to runtime monitoring. By enabling natural language interaction, they lower the skill barrier for AI-powered solutions, making them increasingly relevant for MDE-related tasks aimed at reducing accidental complexity. While research has explored using LLMs to extract models from requirements and informal diagrams or to create domain-specific languages from natural language specifications, their application to model transformations remains largely unexplored. Only preliminary attempts, such as those using ChatGPT[1], suggest that this field is still in its early stages.

Joint Proceedings of the STAF 2025 Workshops: OCL, OOPSLE, LLM4SE, ICMM, AgileMDE, AI4DPS, and TTC. Koblenz, Germany, June 10-13, 2025

*Corresponding author.

alessio.bucaioni@mdu.se (A. Bucaioni); antonio.cicchetti@mdu.se (A. Cicchetti)
0000-0002-8027-0611 (A. Bucaioni); 0000-0003-0416-1787 (A. Cicchetti)

[1]https://www-users.york.ac.uk/dimitris.kolovos/blog/metamodelling-with-chatgpt/episode-2/

To address this problem, this paper investigates the use of LLMs for performing model transformations out-of-the-box. Specifically, we conducted a systematic literature review and designed an experiment to evaluate the precision of ChatGPT-4 in translating UML class diagram models into corresponding Java programs. Our study is supported by an experimental pipeline that automates data collection, transformation execution, and result analysis. It is important to note that this work focuses on LLMs' ability to perform model transformations out-of-the-box, not on code generation. Translating UML diagrams into Java programs was used solely to establish the ground truth, as described in Section 4. Our findings show a cumulative success rate of 94% for transformed models out of 99 input cases, with most generation errors being resolved during the process. However, the experiment also highlights significant issues when dealing with complex models, for which the cumulative success rate drops to only 17%. By providing a systematic approach and a publicly available replication package in Section A, we enable the research community to experiment with ChatGPT-4 in additional transformation tasks. Furthermore, our experimental pipeline can be refined to support alternative LLMs, modelling languages, or datasets.

2. Research Process

Figure 1 illustrates our research process, which follows constructive research techniques commonly applied in computer science and engineering [10, 11]. These techniques involve developing an artefact to address a domain-specific problem, generating knowledge about potential solutions. By integrating existing theoretical knowledge with novel applications, constructive research aims to tackle practically significant issues that remain under-explored in the literature. We began by investigating whether LLMs could reduce the accidental complexity in model transformation processes by performing model transformations out-of-the-box. To assess prior research on this topic, we conducted a Systematic Literature Review (SLR) following Kitchenham et al.'s guidelines [12]. Finding no prior studies directly addressing this question, we designed and executed an experiment to explore our hypothesis. Finally, we analysed the experimental results to evaluate and refine our initial observations.

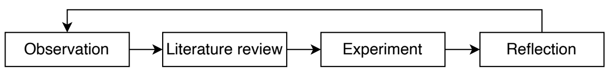

Figure 1: Research process

2.1. Systematic literature review

We conducted an automated search of peer-reviewed literature across four major scientific databases: IEEE Xplore, ACM Digital Library, SCOPUS, and Web of Science. To ensure both rigour and inclusiveness, we used the search string: *("llm" OR "large language model*") AND ("mde" OR "model-driven engineering")*, querying all fields without applying filters. This search yielded 35 potential studies[2]. After removing non-research articles and duplicates, we identified 10 primary studies (Table 1). These were analysed using the guidelines of Cruzes et al. [13]. While we omit details for brevity, all search and selection data, along with the full list of primary studies, are available in our public replication package in Section A. A discussion of the selected studies is presented in Section 3.

2.2. Experiment

Given the absence of prior research on using LLMs to mitigate accidental complexity in model transformation processes, we designed and conducted an experiment to explore this hypothesis. Inspired by recent studies on LLMs for code generation [14, 15], we developed a semi-automated experimental pipeline to streamline data collection, execution, and analysis

Dataset. The dataset needed to be independent, publicly available, and contain a large number of UML class diagram models. To satisfy these criteria, we selected the Lindholmen dataset[3], curated by

[2]Our review does not include any work published after May 2024
[3]http://models-db.com

the University of Rostock. This dataset links to GitHub repositories that use UML, offering access to over 93,000 potential UML-related artefacts[4].

Ground truth. To assess ChatGPT-4's performance in translating UML class diagrams into Java programs, we required a reliable ground truth dataset. For this, we leveraged the code generation capabilities of Modelio, a widely recognized modelling tool[5]. Our choice was driven by two key requirements: handling XMI files and automatically generating Java models. We evaluated alternative tools such as Astah, Papyrus, ArgoUML, and Visual Paradigm, but each presented limitations[6]. Using Modelio ensured that the Java programs serving as benchmarks were accurate translations of the UML class diagrams, mitigating potential threats to construct and internal validity.

Automation. The experimental pipeline was designed to minimize manual intervention. To achieve this, we integrated various tools and technologies to automate key steps. Python scripts handled multiple automation tasks, Modelio was used for generating ground truth models, AutoHotKey[7] automated repetitive keyboard and mouse interactions, and Beyond Compare[8] facilitated automated file comparisons. This automation streamlined the workflow, ensuring consistency and efficiency in executing the experiment.

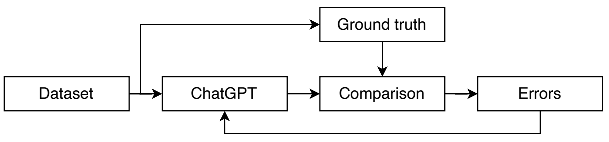

Figure 2: Simplified overview of the experimental pipeline

Reproducibility and Verifiability. To ensure that our experimental pipeline and results can be independently replicated and verified, we provide a public replication package containing all artefacts used in this work in Section A. Figure 2 presents a simplified overview of our experimental pipeline. The process starts with a dataset of UML class diagram models, which are input into ChatGPT-4 for translation into Java programs. The generated Java programs are then compared against ground truth Java models produced by Modelio. Identified discrepancies are used to iteratively refine the ChatGPT-4 output, with a maximum of two re-prompting cycles, as previous studies suggest that additional iterations beyond the third yield minimal improvements [14, 15]. Section 4 provides a detailed discussion on the pipeline's definition and execution.

2.3. Threats to validity

To mitigate threats to conclusion validity, we meticulously followed well-defined research processes and provided a public replication package to ensure reproducibility. A key limitation is the statistical validity of our dataset, which includes 99 models. However, the scarcity of large, high-quality datasets for software engineering research remains a well-known challenge, and previous studies have often relied on even smaller datasets [16]. Potential threats to internal validity stem from the use of Modelio to establish ground truth. However, this risk is minimal, as Modelio is an industry-standard tool for MDE and Java. Our selection was driven by two core requirements: XMI file handling and automatic Java model generation. Alternative tools, such as Astah, Papyrus, ArgoUML, and Visual Paradigm, had limitations preventing their integration into our pipeline. Additionally, while class diagrams may not fully represent all transformation scenarios, they are fundamental to modelling, as they define a system's structural backbone [17]. Construct validity threats relate to model selection and experimental design. Although we ensured variation in structural complexity, our dataset may not fully reflect the complexity of real-world transformations. Furthermore, since ChatGPT-4's training data is not publicly available, there is a potential risk that it may have been trained on similar UML models,

[4]It should be noted that these files may include various UML diagrams, not just class diagrams, and some may contain artefacts such as images rather than serialised UML class diagrams.

[5]https://www.modeliosoft.com/en/products/modelio-sd-java.html

[6]Astah's free version lacked XMI import support, Papyrus required file renaming and additional steps for Java generation, ArgoUML failed to import XMI files, and Visual Paradigm lacked Java model generation capabilities. These constraints made them unsuitable for our automated pipeline.

[7]https://www.autohotkey.com

[8]https://www.scootersoftware.com

which could influence results. Our study intentionally adopted a zero-shot approach to evaluate LLMs' out-of-the-box capabilities, prioritising simplicity to minimise accidental complexity, though more advanced prompting strategies (e.g., Chain of Thought, Tree of Thought) could improve performance. External validity may be affected by the modelling and programming languages used. The dataset's size also poses a limitation, as curating high-quality models required a labour-intensive process involving ChatGPT-4, Modelio, and the Lindholmen dataset. Despite these constraints, we aimed to enhance generalisability by including models with varying structural complexities.

3. Related Work

This section reviews the primary studies identified through our SLR. While an exhaustive analysis is beyond the scope of this paper, Table 1 classifies these studies based on their research focus. Notably, no prior research has explored the use of LLMs to reduce the accidental complexity of model transformation processes, establishing our study as a pioneering contribution. Additionally, despite the increasing number of studies on LLM-based code generation, generating code from natural language is fundamentally different from performing model transformations out-of-the-box. Consequently, we do not include such works in this review.

Table 1
Classification of primary studies

Ref	Title	Authors	Chat-bot	LLMs for software engineering	Prompting	Domain modelling
[18]	Measuring and Clustering Heterogeneous Chatbot Designs	Cañizares et al.	x			
[19]	Building Domain-Specific Machine Learning Workflows: A Conceptual Framework for the State of the Practice	Oakes et al.		x		
[20]	Accelerating Software Development Using Generative AI: ChatGPT Case Study	Rajbhoj et al.		x	x	
[21]	Prompt Engineering: User Prompt Meta Model for GPT Based Models	Tamenaoul et al.			x	
[22]	Chat2Code: A Chatbot for Model Specification and Code Generation, the Case of Smart Contracts	Qasse et al.	x	x		
[23]	Model-Driven Smart Contract Generation Leveraging ChatGPT	Petrović et al.		x		
[24]	Model-Driven Prompt Engineering	Clariso et al.			x	
[25]	Extracting Domain Models from Textual Requirements in the Era of Large Language Models	Arulmohan et al.				x
[26]	Prompting or Fine-tuning? A Comparative Study of Large Language Models for Taxonomy Construction	Chen et al.			x	
[27]	Automated Domain Modeling with Large Language Models: A Comparative Study	Chen et al.				x

3.1. Large Language Models and Model-Driven Engineering

Cañizares et al. explored chatbot design measurement and classification, introducing a suite of metrics and clustering methods [18]. Their tool, Asymob, facilitates the translation of chatbot platforms into a neutral notation, improving comparability. Oakes et al. examined domain-specific machine learning workflows and identified six key challenges in workflow development [19]. Their study highlighted gaps in tool support and recommended future research to reduce accidental complexity, aligning with our study's focus. Rajbhoj et al. investigated systematic prompting strategies for software development life cycle tasks, validating their approach using ChatGPT [20]. Their results show that generative AI can significantly reduce skill barriers in MDE, supporting our premise that LLMs can simplify complex transformation tasks. Tamenaoul et al. developed a meta-model for user prompts, formalizing several prompt engineering patterns [21]. Similarly, Clariso et al. proposed Impromptu, a domain-specific language for platform-independent prompt generation and adaptation [24]. Qasse et al. explored chatbots as an interactive alternative for MDE, developing a framework that generates platform-independent code from conversational inputs, with a focus on smart contracts [22]. Their findings suggest that chatbot-based development can improve accessibility in software engineering. Petrović et al. investigated ChatGPT for automating smart contract generation, proposing a model-driven framework for treating smart contract creation as an interactive dialogue [23]. Their evaluation highlighted ChatGPT's adaptability but also noted challenges such as increased response times and costs. Arulmohan et al. examined LLMs for extracting domain models from textual documents like product backlogs [25]. They compared GPT-3.5 against a state-of-the-practice tool and a CRF-based NLP approach, finding that while GPT-3.5 outperformed standard tools, the CRF approach achieved higher accuracy with minimal training. Chen et al. developed a framework for automated taxonomy construction, comparing LLM prompting with fine-tuning [26]. Their results showed that prompting often outperforms fine-tuning, particularly for smaller datasets, though fine-tuning allows easier post-processing. Chen et al. also explored the automation of domain modelling using LLMs [27]. While GPT-3.5 and GPT-4 demonstrated strong understanding capabilities, they struggled with full automation, achieving only moderate F1 scores in class, attribute, and relationship generation. This complements our study, as it focuses on automating one pillar of MDE—modelling—while we focus on model transformations. In addition to the works identified through our opportunistic SLR, recent studies have begun exploring the use of LLMs for model transformation tasks, for example, the transformation of UML state diagrams into Rebeca models using a few-shot learning approach [28].

3.2. Simplifying Model Transformations

Over the years, many studies have aimed to make model transformations more accessible to users unfamiliar with transformation languages and related technologies. One of the most notable approaches is Varrò's transformation by example method [29]. This method generalizes XML transformation techniques for model transformations, deriving transformation rules from initial interrelated source and target models. These rules can be refined iteratively by adding more source-target model pairs, eliminating the need to learn a dedicated transformation language. Building on Varrò's work, Kappel et al. surveyed early Model Transformation By-Example approaches [30]. Kessentini et al. extended this method with an optimization-based approach, using search-based algorithms (particle swarm optimization and simulated annealing) to determine the best transformation fragment combinations [31]. Among AI-driven solutions, Burgueño et al. proposed a neural network-based model transformation approach [32]. Their encoder-decoder LSTM architecture with an attention mechanism learns transformation patterns from input-output examples, automatically generating transformation outputs once trained.

4. Experimental pipeline

This section presents the experimental pipeline, aligning with the requirements outlined in Section 2. We detail its configuration, including the tools, technologies, and artefacts used, all of which are available in our public replication package in Section A.The pipeline consists of four main phases: cleaning, ground truth generation, generation, and comparison. Figure 3 provides a comprehensive overview, with execution flow indicated by black-circled numbers. We initiated our experimental pipeline with the publicly available Lindholmen dataset, which contained 93,608 files, including 3,722 XMI UML diagrams. To ensure data quality and manageability, we performed a thorough cleaning process (black-circled 1 in Figure 3). This involved removing duplicates, inaccessible files, incompatible character sets, and files exceeding 200KB—beyond the input limit of ChatGPT-4's web version. We also excluded corrupted files that could not be imported into Modelio, as they would prevent ground truth generation. After this screening, 99 usable XMI files remained, forming our initial dataset of textual UML class diagram representations. We imported the XMI files into Modelio to generate the corresponding Java programs, serving as the ground truth (black-circled 2 in Figure 3). During this process, some elements—such as names or attributes—were occasionally missing due to incomplete XMI data or Modelio's interpretation. In such cases, we inserted custom strings to ensure Java grammar compliance. Once exported, Modelio generated each class as a separate file. To streamline comparisons, we used a Python script to merge these files into a single Java file per XMI instance, preserving their content.

Notably, this Java output serves as a model for the eventual implementation code, which would be further refined or manually written in later stages.

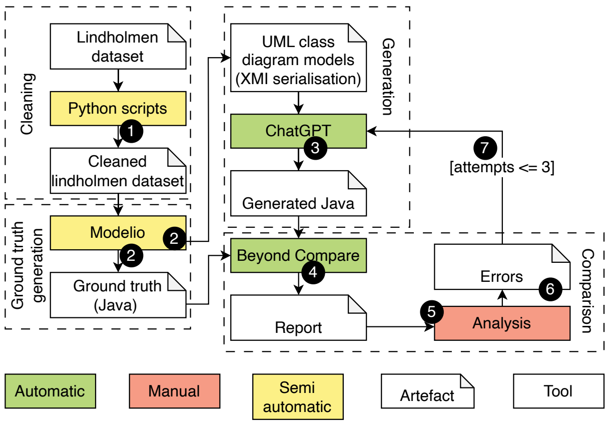

Figure 3: Detailed overview of the experimental pipeline

To minimize threats to validity, we fed ChatGPT-4 with XMI files exported from Modelio rather than using raw dataset files (black-circled 2 in Figure 3), reducing discrepancies between ChatGPT-4 and Modelio interpretations. We then prompted ChatGPT-4 to generate Java programs from these XMI files (black-circled 3 in Figure 3) using the instruction: `The following is an XMI serialization of a UML Class Diagram model. Generate the corresponding Java program. Do not implement get and set functions. Do not add any comments.` To automate this process, we used AutoHotKey for interacting with ChatGPT-4 via pre-programmed inputs and formatting responses. AutoHotKey also facilitated storing the generated outputs in our replication package. After generating the Java programs, we compared the ChatGPT-4-generated Java programs with the ground truth, Java programs created by Modelio (black-circled 4 in Figure 3). It is important to clarify that we do not compare ChatGPT-4 with Modelio regarding the transformation process. Instead, we used Modelio solely to generate the ground truth Java models. Before this comparison, we took additional steps to format the programs and eliminate impurities that might lead to false positives. For example, we removed IDs that Modelio adds to the Java programs using a python script or fixed the formatting using AutoHotKey. Additionally, we standardised the formatting by employing the Language Support for Java extension developed by Red Hat in Visual Studio Code, which helped us eliminate extra spaces and correct indentation errors. For the comparison, we employed Beyond Compare, a tool that facilitates side-by-side comparison of two files. It reads the files and highlights the differences in a detailed comparison report. After generating the comparison report with Beyond Compare, we manually analysed it (black-circled 5 in Figure 3) to filter out irrelevant differences, such as capitalization or ordering variations, while identifying substantial discrepancies like missing classes, incorrect attributes, and erroneous cardinality. We logged all significant differences in a separate error log (black-circled 6 in Figure 3) and used this to create a refined prompt for a subsequent ChatGPT-4 generation attempt (black-circled 7 in Figure 3) similar to this: `In the previous response, the following errors were discovered. The attribute` *`readWriteSingleValuedEnumerationAttribute`* `is missing an` *`enum`* `instance. The function` *`opParameters`* `has the wrong return type. Regenerate the response fixing the errors above.` This cycle was repeated up to two times, as previous research indicates that additional attempts rarely produce further improvements [14, 15].

5. Results

This section presents our study's findings, starting with the success rate of the ChatGPT-4 transformation process and its variation with the structural complexity of UML class diagram models. Additionally, we identify and discuss the most common errors encountered during transformation.

5.1. Success rate

In our experiment, we implemented the semi-automatic pipeline described in Section 2 and applied it to 99 UML class diagram models, allowing up to three iterations. Table 2 summarises the outcomes, reporting the absolute number of successfully and unsuccessfully transformed models, the single success rate per iteration, and the cumulative success rate (total percentage of models successfully transformed up to the given iteration.). In the first iteration, 67 out of 99 models were successfully transformed (67% single and cumulative success rate). The second iteration transformed 21 additional models (66% single success rate), raising the cumulative success rate to 89%. In the third and final iteration, 5 more models were transformed (45% single success rate), bringing the cumulative success rate to 94%. Table 2 also notes that 6 models remained untransformed after three iterations.

Table 2
Absolute value, single and cumulative success rate

	Absolute value	Single success rate	Cumulative success rate
1st iteration	67	67%	67%
2nd iteration	21	66%	89%
3rd iteration	5	45%	94%
Failed	6		
Total	99		

We categorized the XMI files by structural complexity to evaluate ChatGPT-4's performance across different model types. First, we quantified complexity by counting the various structural elements—Classes, Primitive Types, Enumerations, Interfaces, and Associations—in each XMI file and summing them. This metric revealed a wide range of complexity, from a few elements to over 80. To reduce categorization bias, we applied the K-means clustering algorithm [33] with $k = 3$ based on the total number of elements. This yielded three clusters: low complexity models with up to 9 elements (57 models), medium complexity models with 10 to 36 elements (36 models), and high complexity models with more than 36 elements (6 models).

Figures 4 and 5 show the transformation success rates for each cluster. In the low complexity group, 47 out of 57 models (82%) were successfully transformed in the first iteration. A further 9 models succeeded in the second iteration (90% success for that round), boosting the cumulative rate to 98%, and the final model succeeded in the third iteration, reaching a 100% cumulative success rate. In the medium complexity group, 19 of 36 models (53%) transformed in the first iteration. An additional 12 models succeeded in the second iteration (71% for that round), raising the cumulative rate to 87%, and 4 more models succeeded in the third iteration (80% for that round), with a final cumulative success rate of 97%. One model, however, failed to transform after three iterations. For the high complexity group, only 1 of 6 models (17%) transformed in the first iteration. Due to this low rate, we conducted further analysis, which indicated that a high number of associations—specifically, more than 40—was correlated with transformation failures, while the number of classes showed no direct correlation.

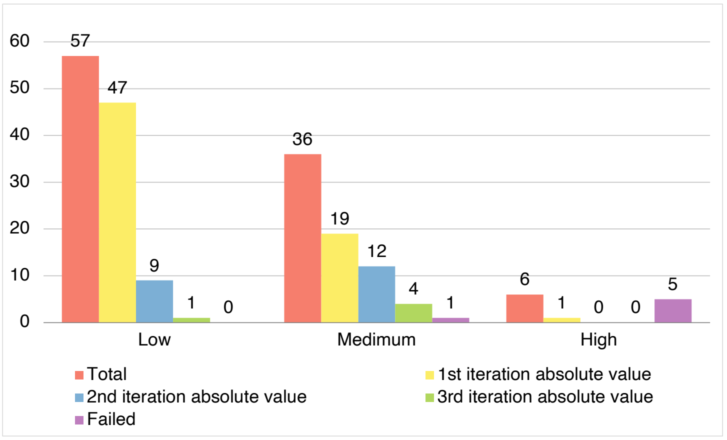

Figure 4: Bar chart showing the total number of models per cluster, the absolute number of model successfully transformed in each iteration, and the absolute number of failed models.

5.2. Errors

During the three executions, we collected all significant differences that emerged from comparing the ChatGPT-4-generated Java programs with the ground truth Java programs created by Modelio, as described in Section 4. We then categorised these differences into error types, which are summarised in Table 3, along with the count of individual occurrences.

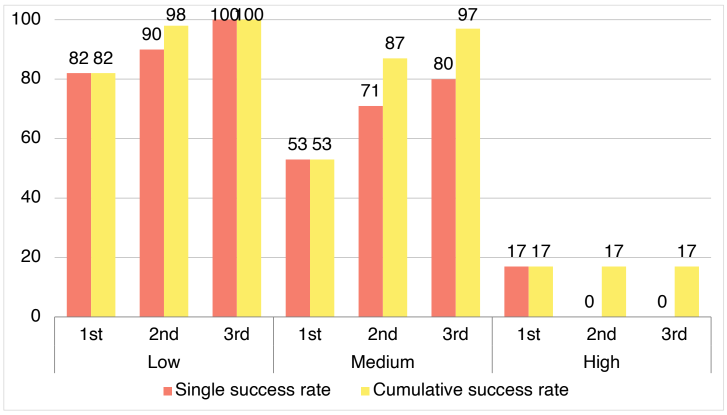

Figure 5: Bar chart showing the single and cumulative success rates per iteration, per cluster.

The data show that wrong type is the most common error, accounting for 61% of the total errors. Examples of these errors include attributes being generated as single objects rather than lists, or attributes whose type was set to Object rather than String, as shown below. The second most common error is incorrect extends relationships among classes, accounting for 16% of the total errors. Other common errors include wrong type, missing classes, and missing attributes. To gain deeper insights into potential correlations between errors and iterations, we analysed the frequency of errors across the three iterations. While no strong correlation emerges between specific errors and iterations—since the most common errors maintain a consistent ranking—most errors are progressively resolved. Exceptions include incorrect extends and missing enumeration instance. The former decreases after the second iteration but remains unresolved in over 40% of cases, while the latter is introduced in the second iteration and only partially corrected in the final one. Additionally, we examined error distribution across structural complexity clusters. Models in the low-complexity cluster exhibit fewer distinct errors, with only three error types exceeding three occurrences and relatively few instances per type. In contrast, models in the medium and high-complexity clusters show a broader range of errors and higher instance counts. This suggests that model complexity is a key factor contributing to errors.

Table 3
Errors

Error	Occurrences	Error	Occurrences
Incorrect extends	60	Missing enum	4
Wrong type	222	Missing parameter	2
Missing class	31	Wrong name	2
Missing attribute	28	Extra attributes generated	1
Lacking enum instance	10	Missing constructor	1
Missing interface	5	Wrong enum instance	1

6. Discussion

Choosing UML and Java notations may be seen as restrictive, particularly since our case focuses on model-to-text transformations [34]. While Czarnecki and Helsen treated model-to-model and model-to-text transformations uniformly, differentiating them mainly by the availability of mature tool support [34], focusing on a single transformation type may limit the broader applicability of our findings. However, UML class diagrams hold a special role in modelling as the most widely used and structurally significant diagrams [35], suggesting a degree of generalizability. Additionally, their selection was instrumental in ensuring a well-sized dataset and ground truth models, reducing potential threats to validity (see Section 2). The lack of large-scale benchmark datasets for software engineering and MDE research remains a well-documented challenge [36].

Our experimental pipeline, based on prior studies [14, 15], establishes ground-truth models using Modelio, an industry-standard tool, but allows for substitution with other tools if needed. The choice of a zero-shot learning strategy may be viewed as a limitation. While advanced prompting techniques like Chain of Thought and Tree of Thought could improve reasoning, we prioritized a zero-shot approach to assess LLMs' out-of-the-box capabilities, minimizing additional complexity. Few-shot learning could further improve success rates, particularly for models with more than 36 elements. Additionally, our experiment shows that just 12 error types emerged during generation, with four accounting for the majority. Addressing these through refined prompting could improve both single and cumulative success rates. Our decision to limit the process to three iterations aligns with prior research [14, 15]. Our findings suggest that ChatGPT-4 can help reducing accidental complexity in model transformation by eliminating the need for transformation languages and related tools. However, it has limitations in handling complex models with many associations. Notably, even for a widely used language like Java—on which ChatGPT-4 has likely been trained—generating fully correct target models remains challenging.

As a pioneering study on LLM-driven model transformation, some aspects beyond this work warrant further investigation. Transformation testing remains an open research area, particularly beyond functional testing [37]. Our black-box testing approach, based on an oracle, could be extended with mutation techniques to improve evaluation. More advanced checks, such as semantic correctness, would require explainability features within LLMs.

Our findings have promising implications for MDE and automated software engineering. They show the potential to simplify model transformations, making them more accessible to non-experts. Notably, all generated target models conformed to the target metamodel, and in some cases, ChatGPT-4 correctly inferred missing information, such as attribute names, improving resilience against domain evolution. However, LLMs are not yet mature enough for unsupervised transformation tasks, particularly with structurally complex models. While they significantly reduce accidental complexity, mechanisms are needed to check for errors and omissions in generated results. Few-shot and multi-shot learning may improve performance for complex models, but systematic research is required to characterize and address potential generation issues. Interestingly, adopting a model transformation chain, as in this study, may simplify such analysis compared to tasks like generating code from natural language, as intermediate steps provide better traceability of patterns.

In conclusion, LLMs like ChatGPT-4 present a novel alternative to traditional DSL-based model transformations. While DSLs ensure precise, deterministic, and reproducible transformations with mature debugging tools, they require significant expertise and upfront effort. In contrast, LLMs lower entry barriers, enabling natural language interaction and flexibility, but they lack the precision, transparency, and scalability of DSLs, particularly for complex models. LLMs excel in adaptability, handling evolving requirements and inferring missing details, yet their black-box nature complicates error tracing. DSLs remain preferable for high-precision, repeatable transformations. The choice between LLMs and DSLs depends on the use case: LLMs enable rapid prototyping and reduce accidental complexity, while DSLs offer fine-grained control. Future research should explore hybrid approaches, integrating DSL precision with the accessibility and adaptability of LLMs.

7. Conclusion and Future Work

Many studies have aimed to reduce the accidental complexity in model transformation processes, yet no research prior to April 2024 has systematically explored the use of LLMs for performing model transformations out of the box. This work investigates ChatGPT-4's potential to address this challenge. We conducted a systematic literature review and designed an experiment to assess ChatGPT-4's effectiveness in automating model transformations. Using a semi-automated pipeline, we applied ChatGPT-4 to 99 UML class diagram models, generating Java programs and comparing them against ground truth programs from a state-of-the-art modeling tool. Our findings indicate a cumulative success rate of 94% after three iterations, with most errors resolved. However, complex models remained a challenge, achieving a cumulative success rate of only 17%.

Future research should explore LLMs in different transformation scenarios, including model-to-model and model-to-text transformations. Expanding the dataset with more diverse models and leveraging few-shot and multi-shot learning strategies could improve success rates, particularly for complex models. Additionally, developing advanced error handling and correction mechanisms [38] will be essential to enhancing accuracy and reliability.

Acknowledgments

This research work has been funded by Vinnova through the iSecure (202301899) and AIDA (202402068) projects, and by the KDT Joint Undertaking through the MATISSE project (101140216).

Declaration on Generative AI

During the preparation of this work, the author(s) used Chat-GPT-4 to support the improvement of grammar and spelling checking. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] D. C. Schmidt, et al., Model-driven engineering, Computer-IEEE Computer Society- 39 (2006).
[2] J. Bézivin, O. Gerbé, Towards a precise definition of the omg/mda framework, in: Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001), IEEE, 2001, pp. 273–280.
[3] S. Sendall, W. Kozaczynski, Model transformation: The heart and soul of model-driven software development, IEEE software 20 (2003) 42–45.
[4] A. Bucaioni, A. Cicchetti, F. Ciccozzi, S. Mubeen, M. Sjödin, A metamodel for the rubus component model: extensions for timing and model transformation from east-adl, IEEE Access 5 (2016) 9005–9020.
[5] A. Bucaioni, S. Mubeen, A. Cicchetti, M. Sjödin, Exploring timing model extractions at east-adl design-level using model transformations, in: 2015 12th international conference on information technology-new generations, IEEE, 2015, pp. 595–600.
[6] G. Liebel, N. Marko, M. Tichy, A. Leitner, J. Hansson, Assessing the state-of-practice of model-based engineering in the embedded systems domain, in: Model-Driven Engineering Languages and Systems: 17th International Conference, MODELS 2014, Valencia, Spain, September 28–October 3, 2014. Proceedings 17, Springer, 2014, pp. 166–182.
[7] P. Mohagheghi, W. Gilani, A. Stefanescu, M. A. Fernandez, B. Nordmoen, M. Fritzsche, Where does model-driven engineering help? experiences from three industrial cases, Software & Systems Modeling 12 (2013) 619–639.
[8] A. Bucaioni, A. Cicchetti, F. Ciccozzi, Modelling in low-code development: a multi-vocal systematic review, Software and Systems Modeling 21 (2022) 1959–1981.
[9] D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, M. Wimmer, Low-code development and model-driven engineering: Two sides of the same coin?, Software and Systems Modeling 21 (2022) 437–446.
[10] G. D. Crnkovic, Constructive research and info-computational knowledge generation, in: Model-Based Reasoning in Science and Technology, Springer, 2010, pp. 359–380.
[11] K. Lukka, The constructive research approach, Case study research in logistics. Publications of the Turku School of Economics and Business Administration, Series B 1 (2003) 83–101.
[12] B. Kitchenham, P. Brereton, A systematic review of systematic review process research in software engineering, Information and software technology (2013).
[13] D. S. Cruzes, T. Dyba, Recommended steps for thematic synthesis in software engineering, in: Procs of ESEM, IEEE, 2011, pp. 275–284.
[14] S. Lertbanjongngam, B. Chinthanet, T. Ishio, R. G. Kula, P. Leelaprute, B. Manaskasemsak, A. Rungsawang, K. Matsumoto, An empirical evaluation of competitive programming ai: A case study of alphacode, 2022. ArXiv preprint arXiv:2208.08603.
[15] A. Bucaioni, H. Ekedahl, V. Helander, P. T. Nguyen, Programming with chatgpt: How far can we go?, Machine Learning with Applications 15 (2024) 100526.
[16] S. Yang, H. Sahraoui, Towards automatically extracting uml class diagrams from natural language specifications, in: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, 2022, pp. 396–403.
[17] B. Combemale, J. Gray, B. Rumpe, Model-based code generation works: But how far does it go?—on the role of the generator, Software and Systems Modeling (2024) 1–2.
[18] P. C. Canizares, J. M. López-Morales, S. Pérez-Soler, E. Guerra, J. de Lara, Measuring and clustering heterogeneous chatbot designs, ACM Transactions on Software Engineering and Methodology 33 (2024) 1–43.
[19] B. J. Oakes, M. Famelis, H. Sahraoui, Building domain-specific machine learning workflows: A conceptual framework for the state of the practice, ACM Transactions on Software Engineering and Methodology 33 (2024) 1–50.
[20] A. Rajbhoj, A. Somase, P. Kulkarni, V. Kulkarni, Accelerating software development using generative ai: Chatgpt case study, in: Proceedings of the 17th Innovations in Software Engineering Conference, 2024, pp. 1–11.
[21] H. Tamenaoul, M. E. Hamlaoui, M. Nassar, Prompt engineering: User prompt meta model for gpt based models, in: The International Conference on Artificial Intelligence and Smart Environment, Springer, 2023, pp. 428–433.
[22] I. Qasse, S. Mishra, B. þór Jónsson, F. Khomh, M. Hamdaqa, Chat2code: A chatbot for model specification and code generation, the case of smart contracts, in: 2023 IEEE International Conference on Software Services Engineering (SSE), IEEE, 2023, pp. 50–60.
[23] N. Petrović, I. Al-Azzoni, Model-driven smart contract generation leveraging chatgpt, in: International Conference On Systems Engineering, Springer, 2023, pp. 387–396.

[24] R. Clarisó, J. Cabot, Model-driven prompt engineering, in: 2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, 2023, pp. 47–54.
[25] S. Arulmohan, M.-J. Meurs, S. Mosser, Extracting domain models from textual requirements in the era of large language models, in: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2023, pp. 580–587.
[26] B. Chen, F. Yi, D. Varró, Prompting or fine-tuning? a comparative study of large language models for taxonomy construction, in: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2023, pp. 588–596.
[27] K. Chen, Y. Yang, B. Chen, J. A. H. López, G. Mussbacher, D. Varró, Automated domain modeling with large language models: A comparative study, in: 2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, 2023, pp. 162–172.
[28] Z. Moezkarimi, K. Eriksson, A. A. Johansson, A. Bucaioni, M. Sirjani, Harnessing chatgpt for model transformation in software architecture: From uml state diagrams to rebeca models for formal verification, in: 4th International Workshop of Model-Driven Engineering for Software Architecture, ???? URL: http://www.es.mdu.se/publications/7130-.
[29] D. Varró, Model transformation by example, in: Model Driven Engineering Languages and Systems: 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006. Proceedings 9, Springer, 2006, pp. 410–424.
[30] G. Kappel, P. Langer, W. Retschitzegger, W. Schwinger, M. Wimmer, Model transformation by-example: a survey of the first wave, Conceptual Modelling and Its Theoretical Foundations: Essays Dedicated to Bernhard Thalheim on the Occasion of His 60th Birthday (2012) 197–215.
[31] M. Kessentini, H. Sahraoui, M. Boukadoum, O. B. Omar, Search-based model transformation by example, Software & Systems Modeling 11 (2012) 209–226.
[32] L. Burgueno, J. Cabot, S. Li, S. Gérard, A generic lstm neural network architecture to infer heterogeneous model transformations, Software and Systems Modeling 21 (2022) 139–156.
[33] A. Likas, N. Vlassis, J. J. Verbeek, The global k-means clustering algorithm, Pattern recognition 36 (2003) 451–461.
[34] K. Czarnecki, S. Helsen, et al., Classification of model transformation approaches, in: Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, volume 45, USA, 2003, pp. 1–17.
[35] G. Reggio, M. Leotta, F. Ricca, Who knows/uses what of the uml: A personal opinion survey, in: Model-Driven Engineering Languages and Systems: 17th International Conference, MODELS 2014, Valencia, Spain, September 28–October 3, 2014. Proceedings 17, Springer, 2014, pp. 149–165.
[36] F. Bozyigit, T. Bardakci, A. Khalilipour, M. Challenger, G. Ramackers, Ö. Babur, M. R. Chaudron, Generating domain models from natural language text using nlp: a benchmark dataset and experimental comparison of tools, Software and Systems Modeling (2024) 1–19.
[37] J. Troya, S. Segura, L. Burgueño, M. Wimmer, Model transformation testing and debugging: A survey, ACM Computing Surveys 55 (2022) 1–39. URL: http://dx.doi.org/10.1145/3523056. doi:10.1145/3523056.
[38] A. Bucaioni, G. Gualandi, J. Toma, Benchmarking large language models for autonomous run-time error repair: Toward self-healing software systems, in: International Conference on Evaluation and Assessment in Software Engineering, 2025. URL: http://www.es.mdu.se/publications/7185-.

A. Online Resources

The replication package is available at: https://anonymous.4open.science/r/ModelTransformationWithLLMs-FD3F/