Learning to Transform: Evaluating LLMs on Model Transformation by Example

Duy Dao*, Alessio Bucaioni†, Antonio Cicchetti†
* Mälardalen University (Sweden), *khanh.duy.dao@mdu.se*
† Mälardalen University (Sweden), *name.surname@mdu.se*

***Abstract*—Large language models have shown promising potential in reducing the accidental complexity of model-driven engineering, particularly in automating model transformation tasks. However, existing research has shown that large language models often struggle with correctness, generalizability, and performance in complex transformation scenarios. This paper investigates the feasibility of applying large language models within the paradigm of model transformation by example, an intuitive technique that derives transformation logic from prototypical source-target model pairs, thereby reducing cognitive load and bypassing the need for formal transformation specifications.**

We empirically benchmark five LLMs, ChatGPT-4.5, DeepSeek V3, DeepHermes 3 LLaMA 3 8B, QwQ 32B, and OlympicCoder 32B, across three transformation scenarios of increasing complexity: RDBMS-to-UML, UML-to-Java, and SysML-to-AAS. Each scenario is evaluated under three configurations, varying the number of example pairs provided. In each configuration, the LLMs are tasked with directly generating target models given source models, example pairs, and initial mapping rules. Model outputs are assessed using correctness and weighted success metrics that account for structural and semantic fidelity.

Our findings reveal that LLMs perform well in syntactically regular transformations but struggle in semantically rich or complex scenarios. Additional examples do not consistently enhance performance and may even degrade it, highlighting the models' limitations in abstraction and semantic grounding. GPT-4.5 delivers the highest peak performance but suffers from instability, while smaller models like DeepSeek V3 and OlympicCoder 32B offer more stable, scalable results.

***Index Terms*—Model Transformation By Example, Large Language Models, Benchmarking.**

I. Introduction

In recent years, several studies have investigated the use of Large Language Models (LLMs) to reduce the accidental complexity commonly associated with Model-Driven Engineering (MDE), particularly in model transformation tasks [1], [2]. Accidental complexity refers to the non-essential difficulties that stem not from the intrinsic nature of the task, but from the limitations and intricacies of the tools, languages, and techniques used to perform it [3]. While preliminary results have shown the potential of LLMs in code generation [4] and automating model transformations [1], substantial challenges persist, particularly when addressing less-explored transformation scenarios and ensuring the correctness and generalizability of the generated transformations [1].

In this context, our research aims to address these limitations by investigating the feasibility and effectiveness of leveraging LLMs within the paradigm of Model Transformation by Example (MTBE). In MDE, MTBE is an intuitive approach that simplifies transformation tasks by allowing practitioners to derive transformation rules from a prototypical set of interrelated source and target models, specified in a purely declarative manner [5]. Our central intuition is that, when applied in the context of LLMs, MTBE can facilitate the definition of transformations by feeding concrete example pairs directly to the models. By alleviating cognitive load, this paradigm has the potential to enhance the correctness and reliability of transformations generated by LLMs.

In this research, we empirically benchmark five different LLMs, ChatGPT-4.5, DeepSeek V3, DeepHermes 3 LLaMA 3 8B, QwQ 32B, and OlympicCoder 32B, to evaluate their capability to autonomously perform MTBE. The evaluation spans three representative transformation scenarios of increasing complexity: Relational Database Schema to UML Class Diagram (RDBMS-to-UML), UML Class Diagram to Java (UML-to-Java), and System Modeling Language to Asset Administration Shell (SysML-to-AAS). These scenarios were selected to cover a range of transformation types, from structural to behavioral and cross-domain mappings. For each scenario, we define three experimental configurations based on the number of example pairs (i.e., source and target model pairs) provided to the LLM: one-pair input, two-pair input, and four-pair input. This design allows us to systematically assess the impact of example quantity on the models' transformation accuracy and generalization ability. Each LLM was given a single attempt to perform the MTBE task per configuration and directly generate target models from the given source models, example pairs, and initial mapping rules according to the scenario and configuration. Success was evaluated using two complementary metrics: the percentage of fully correct transformations and a weighted success score that accounts for partial correctness based on the structural and semantic fidelity of the generated models. We have made a public replication package available in Section VII.

Our evaluation reveals substantial performance variability across tasks, closely tied to transformation complexity. LLMs exhibit strong performance in structured scenarios like RDBMS-to-UML, where mappings are relatively deterministic

This work is supported by the Swedish Agency for Innovation Systems through the project "iSecure: Developing Predictable and Secure IoT for Autonomous Systems" (2023-01899), and by the Key Digital Technologies Joint Undertaking through the project "MATISSE: Model-based engineering of digital twins for early verification and validation of industrial systems" (101140216).

and rule-based. However, their effectiveness diminishes in semantically rich and syntax-intensive tasks, such as SysML-to-AAS and UML-to-Java, where abstraction and interpretation play a greater role. Notably, increasing the number of example pairs does not consistently improve performance and can, in some cases, degrade it, suggesting limitations in semantic grounding and a risk of cognitive overload within the models. GPT-4.5 stands out with superior yet volatile scalability, while smaller, fine-tuned models like DeepSeek V3 and OlympicCoder 32B demonstrate more stable results, emphasizing the value of targeted prompt engineering and task-specific adaptation.

In summary, this paper makes the following contributions:

- It provides early empirical evidence on the capability of LLMs to autonomously perform MTBE.
- It benchmarks five different LLMs across 45 curated configurations spanning three distinct model transformation scenarios.

The remainder of this paper is organized as follows. In Section II, we provide relevant knowledge for our paper and review existing studies that are related to this work. In Section III, we describe the methodology used to benchmark LLMs for MTBE methodology and detail its implementation. In Section IV, we present the results of our experiments. In Section V, we discuss these results along with the potential threats to validity. Finally, in Section VI, we conclude the paper and outline possible future research avenues.

II. Background

This section outlines the foundational concepts necessary to understand the remainder of the paper and reviews relevant related work

A. Model Transformation by Example

MTBE is an MDE technique that defines model transformations using concrete example models rather than traditional metamodel-based transformation rules [5]. It supports the (semi-)automatic generation of transformation rules from prototypical pairs of source and target models, allowing essential mapping cases to be specified without requiring in-depth expertise in transformation languages [6]. This approach lowers the barrier for domain experts and modelers, who are typically more familiar with the concrete syntax, enabling them to define model transformations without needing extensive knowledge of the underlying metamodels [6].

The MTBE process begins with the *manual setup of prototype mapping models*, where designers create an initial set of interrelated source and target model pairs. These serve as concrete examples from which the system can learn. Following this, the system performs an *automated derivation of transformation rules* by analyzing the relationships within the prototype mappings. These initial rules are capable of correctly transforming at least the provided source models into their corresponding target models. Once generated, the rules undergo a phase of *manual refinement*, during which designers adjust them to introduce generalizations or impose necessary constraints. Finally, the process concludes with the *automated execution of the transformation*, where the refined rules are applied to new source models to produce target models. These new cases serve both to validate correctness and to inform the creation of further prototype mappings, thus continuing the iterative refinement cycle. Since the final set of transformation rules is unlikely to be fully derived from the initial set of prototype models, the MTBE approach is inherently highly iterative and interactive [5]. Designers can revise or refine the generated rules at any point, and correctness is assessed continuously by treating each prototype mapping model as a test case. In the MTBE studies [5], [6], both the initial prototype mappings and derived rules are based on specific assumptions, such as treating source and target models as graph-like structures and enforcing one-to-one mappings between source and target nodes.

The MTBE approach offers several advantages for modelers. Since it is model-centric, it presents a low entry barrier by allowing designers to work with familiar modeling constructs when specifying transformations. It also reduces the manual effort typically required in traditional model transformation methods, as many transformation rules can be synthesized automatically. Being inherently iterative, MTBE supports incremental refinement of mapping rules and offers flexibility for designers to override or generalize rules as needed. However, the approach also has notable limitations. It still depends heavily on human intuition for defining the initial prototype models and for generalizing transformation rules, which constrains its level of automation. Moreover, applying MTBE at an industrial scale may prove challenging, as real-world systems often involve complex, heterogeneous models with numerous elements and types, potentially hindering validation and scalability.

B. Large Language Model

LLMs are a type of artificial intelligence based on advanced deep neural networks trained on massive volumes of textual data, including books, source code, articles, and websites, to learn complex language patterns and relationships [7]. These models are capable of generating coherent and grammatically correct human-like text or syntactically accurate code snippets. However, their outputs are inherently probabilistic and may still contain semantic errors [7]. Multiple versions of LLMs have been released in recent years, including GPT, DeepSeek, and LLaMA, each offering various improvements in capability and efficiency. These advancements open up a wide range of opportunities for applying LLMs in software engineering. For instance, Ozkaya [8] highlights potential use cases such as automated code generation and assistance, developer feedback, automated testing, and natural language translation within software systems.

This paper investigates the effectiveness of leveraging the capabilities of LLMs for MTBE. To conduct this study, five diverse LLMs were selected: ChatGPT-4.5, DeepSeek V3, DeepHermes 3 LLaMA 3 8B, QwQ 32B, and OlympicCoder 32B. Each model exhibits unique architectural characteristics

and training configurations, offering a broad spectrum of generative behaviors.

ChatGPT-4.5 is one of the strongest models for complex reasoning, structured problem-solving, and detecting subtle patterns, all of which are critical in model transformation tasks that require fine-grained rules and mappings. Its superior few-shot learning capabilities make it well-suited for in-context learning scenarios[1].

DeepSeek v3 is known for its excellence in structured reasoning and generating programmatic outputs such as code, data models, and structured transformations. Its quality is close to that of GPT-4, offering flexibility for potential fine-tuning on MTBE-specific tasks. DeepSeek's abstraction and schema reasoning capabilities are especially relevant, as MTBE often involves interpreting instance models[2].

DeepHermes 3 (LLaMA 3 8B) is a smaller, more efficient model that allows for faster experimentation cycles. It specializes in instruction-following and transformation tasks, making it well-suited for converting example pairs into generalizable transformation logic[3].

QwQ 32B has been trained across a wide range of tasks, including both code generation and complex text reasoning. This broad tuning makes it a balanced model in terms of quality and versatility, particularly strong in tasks involving code transformation[4].

OlympicCoder 32B is specifically optimized for code generation and logical reasoning, aligning closely with the requirements of MTBE. It is well-suited for tasks that demand interpretation of formal syntax, code rewriting, and transformation, making it a highly relevant model for this study[5].

C. Related Work

Recent research has increasingly explored LLMs as tools for automating and simplifying complex tasks within MDE, particularly addressing the accidental complexity arising from traditional model transformation methods. Kazai et al. provided early evidence of using LLMs such as ChatGPT-4 to perform model transformations directly from UML diagrams to Java code, achieving a cumulative success rate of 94% for simpler models, but facing significant challenges with complex transformations, where performance sharply decreased to 17% [1]. This highlights both the potential and the limitations of LLMs in handling domain-specific and structurally intricate transformations.

Moezkarimi et al. explored the use of ChatGPT-4 to automate the transformation of UML state diagrams into Rebeca models by using few-shot learning, where GPT-4 was provided with example pairs of UML state diagrams and corresponding Rebeca models [2]. The research showed GPT-4 achieving up to 85% syntactic correctness and 69% weighted success on select examples, but also revealed incorrect instantiation, omitted semantics, or hallucinated constructs. Their findings underscore the importance of metadata augmentation, prompt design, and dataset structure in improving LLM-based transformations and advocate for combining LLMs with model-checking tools like Afra to achieve practical semi-automatic verification pipelines, providing a blueprint for LLM-assisted model transformations in formal domains and emphasizing the need for structured prompts and tailored metadata.

Further relevant research by Chen et al. demonstrated the feasibility of using LLMs for automated domain modeling tasks, underscoring their potential in recognizing and generating domain-specific elements but also emphasizing challenges in achieving full automation and accurate semantic understanding [9]. Similarly, Arulmohan et al. explored the extraction of domain models from textual requirements using GPT-3.5, revealing that while LLMs can significantly streamline model generation processes compared to traditional NLP-based methods, they may still struggle with semantic precision and generalization [10].

Efforts by Rajbhoj et al. illustrated systematic prompting techniques for using ChatGPT to automate various tasks within the software development lifecycle, showing that appropriate prompt engineering could notably enhance the utility of generative AI in reducing the barriers to using complex transformation tools and methods [11]. Similarly, Clariso et al. presented a domain-specific language aimed at improving the adaptability and precision of prompts for LLMs in model-driven scenarios, which could potentially mitigate some of the inherent limitations observed in direct, zero-shot transformations [12].

These studies collectively underscore a growing interest and potential in leveraging LLMs to reduce the complexity associated with model transformation tasks in MDE. They highlight the promise of these AI-driven methods in making model transformations more accessible to users with limited expertise, while also clearly identifying current limitations, particularly in handling semantic complexity, structural intricacies, and the need for precise and effective prompting strategies.

III. Research Method

Our goal is to assess the feasibility and accuracy of LLMs in learning transformation logic from a limited set of example pairs and applying it to generate correct target models. To this end, we employed a structured methodology across three distinct transformation scenarios, each evaluated under three different configurations that vary in the number of input example pairs. The overall workflow consists of three main phases: (i) preparing transformation prompts that include initial mapping rules, metamodels, and several example source-target model pairs depending on the configuration; (ii) querying the LLM with a new source model and requesting it to generate the corresponding target model; (iii) validating the generated output with expected output based on ground truth examples using weighted evaluation framework [2].

[1]https://openai.com/chatgpt

[2]https://deepseek.com/blog/deepseek-v2-and-v3

[3]https://huggingface.co/collections/Darkstorm225/deephermes-llama3-8b-664ef3bbddf1cb05f38d37cc

[4]https://huggingface.co/QwQ/QWen-32B

[5]https://huggingface.co/OpenBMB/OlympicCoder-32B

A. Transformation Scenarios and Configurations

To evaluate the capabilities of LLMs in model transformation tasks, we selected three distinct scenarios of increasing complexity, encompassing a range of transformation types, from structural and behavioral to cross-domain mappings. Each scenario's dataset consists of four distinct model pairs as example pairs and one model acting as the source model.

RDBMS-to-UML This task involves transforming structured data representations into object-oriented models [5], [13]. It maps tables to UML classes, columns to attributes, and primary/foreign key relationships to associations. The datasets for this scenario were manually constructed based on information provided in Varro et al. [5]. Each of the RDBMS models is XML-based with varying complexity, ranging from four to eight tables per model, with each table consisting of three to eight columns. The UML class diagrams are serialized in XMI format with an equivalent number of classes, attributes, and associations. This transformation tests the model's ability to understand structural correspondences and enforce integrity constraints in UML notation.

UML-to-Java In this scenario, the model translates object-oriented design representations into executable code skeletons [1], [2]. It converts UML classes, attributes, operations (methods), visibility modifiers, and inheritance relationships into Java class definitions. The datasets found in this scenario were manually designed based on the research of Kzai et al. [1], including object-oriented UML class diagrams paired with equivalent Java code skeletons. The UML class diagram is serialized in XMI format, with each containing two to five classes with each consisting of a range from three to six attributes and two to four associations between classes. The corresponding Java code skeletons are provided in standard Java source file format, organized in packages with clearly defined class structure, attributes, getters and setters, method signatures, and associations. This task evaluates the LLM's capacity to bridge design and implementation-level abstractions, preserving both syntax and semantics.

SysML-to-AAS This transformation translates system-level models into digital twin representations compliant with the AAS meta-model. SysML constructs such as blocks, parts, ports, states, and attributes are mapped to AAS submodels, properties, and operations. The source and target example model pairs are derived from the publicly available dataset introduced by Enxhi Ferko et al. [14], which provides representative SysML and their corresponding AAS submodels. This dataset serves as a realistic benchmark for digital twin integration tasks. [14]. Each source model is defined using a textual SysML representation, each contains five to ten top-level *Definitions*, representing general concepts such as part, port, interface, and action [14]. These definitions serve as building blocks comprising more complex *Usage* elements. Each defined block contains a varying number of components, ranging from one to eight per block, including parts, subsets, redefinitions, interfaces, input/output ports, and attributes. The corresponding target AAS model is serialized using an XMI-based format, comprising entities, submodel elements, relationships, qualifiers, and file references. This scenario requires semantic interpretation of engineering concepts and formal conformance to an industrial standard.

To systematically study LLM performance under varying learning conditions, we defined three configurations for each transformation scenario. Combined, these scenarios and configurations provide a controlled experimental framework to assess how effectively LLMs can perform model transformations across different levels of supervision and abstraction.

One Example Pair The LLM is provided with a single example model pair (source and target) as a minimal input. This configuration assesses the model's ability to infer transformation rules from a single instance.

Two Example Pairs The LLM is given two example model pairs. This setup allows us to analyze whether limited diversity in training samples improves rule induction and generalization performance.

Four Example Pairs With four example model pairs, the LLM has broader exposure to transformation patterns and variations. This configuration serves as a measure of how performance scales with additional examples and provides insight into the model's capacity to learn reusable transformation logic.

B. Prompting and Model Generation via LLM-based MTBE

Our LLM-based MTBE process follows a structured, multi-phase methodology comprising three main stages: preparation and prompt engineering, providing example pairs and initial transformation rules to the LLMs, and output validation with conformance checking.

Each transformation scenario is initiated with a crafted prompt designed to instruct the LLMs. These prompts are tailored to leverage the linguistic capabilities of the models by blending natural language with domain-specific syntax, thereby facilitating a clearer understanding of the intended transformation logic.

For each of the three transformation scenarios: SysML-to-AAS, UML-to-Java, and RDBMS-to-UML, each configuration (one, two, and four example pairs) was executed once per scenario. This process was repeated across five different LLMs, resulting in a total of 45 runs.

This systematic execution enabled a consistent and comparative evaluation across all models and configurations. We

formalize the prompt structure as follows:

$$Prompt = \underbrace{\text{Mapping Rules}}_{M} + \underbrace{\text{Example Pairs}}_{E_n} + \underbrace{\text{Metamodels}}_{MM} \quad (1)$$

where M denotes a description of the transformation rules (i.e., transformation logic), provided either as a textual explanation or in a structured format; E_n represents n ($n \in \{1, 2, 4\}$) example source-target model pairs that demonstrate complete mappings; MM captures optional references to metamodels or type definitions to ensure semantic alignment. The LLM is then presented with a previously unseen source model S_{new} and asked to generate the corresponding target model T_{new} by analyzing information from $(M + E_n + MM)$.

An example of the general prompt used to generate the output model across all three transformation scenarios and their respective example pair configurations is provided below.

```
In a model-driven engineering context, I am
transforming "Your source type" models into "Your
target type" models, and I want you to help by
doing the translation. I will give you in input the
following artefacts: a set of mapping rules ("Your
mapping rules file") telling you how to translate
a "Your source type" model into a "Your target
type" model, "Number of example pairs" example
pair of "Your source type" model and "Your target
type" models ("Your source example file" and "Your
target example file") translated from such "Your
source type" model, a "Your source type" metamodel
and a "Your target type" metamodel ("Your source
metamodel file", "Your target metamodel file"),
and a new complete "Your source type" model ("Your
source file") that I want you to transform into
"Your target type" model. I want you to consider the
mapping rules and the example of "Your source type"
model elements already translated into "Your target
type" mode element, and based on them generate a new
"Your target type" model that is translated from the
new "Your source type" model that I provided you.
I want that the "Your target type" model that you
will generate to be conformant to the "Your target
type" provided example and the "Your target type"
metamodel If the new "Your source type" model that I
want you to translate contains elements not covered
by the mappings that I provided you, please alert
me, but try to still generate an "Your target type"
model compliant to the "Your target type" metamodel.
```

C. Validation Methodology

Each LLM is used to generate a single target model per prompt. The resulting outputs are then manually reviewed and benchmarked for comparative analysis.

Specifically, each target model undergoes a two-tier validation process: structural comparison, which verifies whether the generated model matches the expected output based on ground truth examples (e.g., compilable Java code, valid JSON, or well-formed XMI), and semantic validation, which assesses whether the output correctly preserves the structure, constraints, and intent of the original source model.

In particular, we adopted a weighted evaluation framework [2], which enables a nuanced and systematic analysis of the generated outputs. This framework assigns differential weights to various aspects of correctness, error, and completeness, thereby reflecting their relative importance in the context of model transformation quality and practical applicability.

- LOC (Lines of Code): The total number of lines in the ground truth model, serving as the normalization base for the metrics. These lines refer to individual elements or statements in the serialized output models, e.g., XML tags in XMI, Java source code lines, SysML textual statements, etc.
- Cr (Correct Lines): Lines in the LLM-generated model that exactly match the ground truth model; each such line is awarded +1 point.
- InCr (Incorrect Lines): Lines in the LLM-generated model that are syntactically or semantically incorrect with respect to the ground truth model; each incurs a penalty of -1 point. Syntactic errors include malformed syntax, tags, and semantic errors involve incorrect mappings or misrepresentation of model elements.
- Ad (Additional Lines): Lines in the LLM-generated model that are not present in the ground truth model. If such lines are neutral, i.e., they do not affect the correctness of the transformation, they receive 0 points. However, if they are detrimental (e.g., misleading, semantically invalid), they are penalized with -1 point.
- Miss (Missing Elements): Lines or elements that are expected but absent in the LLM-generated model. Due to their critical impact on completeness, each missing element results in a -2 point penalty.

Based on the above framework, we derived two metrics summarizing the overall transformation quality. Percentage of Correctness (%Correct) captures the proportion of correct lines relative to the total number of expected and missing lines, offering a normalized measure of accuracy.

$$\%Correct = \frac{Cr}{LOC + Miss} \quad (2)$$

Weighted Success Rate (%WSuccess) accounts for not only correct and incorrect lines, but also evaluates the impact of additions and omissions. It provides a comprehensive perspective on the semantic and structural adequacy of the transformation.

$$\%WSuccess = \frac{Cr - InCr + Ad \cdot (0 \text{ or } -1) - 2 \cdot Miss}{LOC} \quad (3)$$

IV. Result Evaluation

This section presents a comparative summary of the performance of all five LLMs (GPT-4.5, DeepSeek V3, DeepHermes 3 LLaMA 3 8B, QwQ 32B, and OlympicCoder 32B) across three model transformation scenarios. Each task was evaluated under three configurations, varying the number of provided example pairs (1, 2, and 4), to assess the models' effectiveness under different levels of supervision. Table I summarizes the results of our experiments. For each scenario and configuration, it reports the percentage of correct transformations (% Correct) and the weighted success score (% WSuccess), as computed using the evaluation framework introduced earlier.

A. RDBMS-to-UML

This scenario evaluates the transformation from relational database schemas to UML class diagrams, a task that is largely

TABLE I
PERFORMANCE EVALUATION OF LLMS IN MODEL TRANSFORMATION BY EXAMPLE

Scenario	Configuration	LLMs	LoC	Cr (+1)	Incr (-1)	Ad (0/-1)	Miss (-2)	% Correct	% WSuccess
RDBMS-to-UML	1 example pair	GPT_4.5	75	75	0	0	0	**100%**	**100%**
		DeepHermes 3 Llama 3 8B	91	83	0	8/0	0	91%	91%
		DeepSeek V3	83	83	0	0	0	**100%**	**100%**
		QwQ 32B	91	83	0	8/0	0	91%	91%
		OlympicCoder 32B	75	75	0	0	0	**100%**	**100%**
	2 example pairs	GPT_4.5	81	73	8	8/0	0	90%	80%
		DeepHermes 3 Llama 3 8B	111	103	0	8/0	2	91%	89%
		DeepSeek V3	83	83	0	0	0	**100%**	**100%**
		QwQ 32B	91	83	0	8/0	0	91%	91%
		OlympicCoder 32B	83	83	0	0	0	**100%**	**100%**
	4 example pairs	GPT_4.5	82	74	0	8/0	0	90%	90%
		DeepHermes 3 Llama 3 8B	93	82	1	8/2	1	87%	82%
		DeepSeek V3	83	83	0	0	0	**100%**	**100%**
		QwQ 32B	88	80	0	8/0	10	81%	68%
		OlympicCoder 32B	82	74	0	8/0	0	90%	90%
UML-to-Java	1 example pair	GPT_4.5	114	114	0	0	0	**100%**	**100%**
		DeepHermes 3 Llama 3 8B	220	197	23	0	28	79%	53%
		DeepSeek V3	130	118	0	0/12	0	90%	81%
		QwQ 32B	130	118	0	0/12	0	90%	81%
		OlympicCoder 32B	134	119	0	0/15	0	88%	77%
	2 example pairs	GPT_4.5	96	62	34	0	46	43%	0% (-66%)
		DeepHermes 3 Llama 3 8B	126	114	0	0/12	3	88%	76%
		DeepSeek V3	130	118	0	0/12	0	**90%**	**81%**
		QwQ 32B	126	114	0	0/12	3	88%	76%
		OlympicCoder 32B	130	118	0	0/12	0	**90%**	**81%**
	4 example pairs	GPT_4.5	83	65	9	0/9	31	57%	0% (-18%)
		DeepHermes 3 Llama 3 8B	98	98	0	0	12	89%	75%
		DeepSeek V3	132	120	0	0/12	0	**90%**	**81%**
		QwQ 32B	130	118	0	0/12	0	**90%**	**81%**
		OlympicCoder 32B	130	118	0	0/12	0	**90%**	**81%**
SysML-to-AAS	1 example pair	GPT_4.5	74	74	0	0	10	88%	73%
		DeepHermes 3 Llama 3 8B	165	73	3	29/60	2	43%	2%
		DeepSeek V3	144	104	0	40/0	0	72%	72%
		QwQ 32B	85	54	31	0	0	63%	27%
		OlympicCoder 32B	95	95	0	0	7	**93%**	**85%**
	2 example pairs	GPT_4.5	58	57	0	1/0	10	83%	63%
		DeepHermes 3 Llama 3 8B	74	44	0	0/30	25	44%	0%
		DeepSeek V3	41	40	1	0	22	63%	0% (-12%)
		QwQ 32B	90	76	14	0	0	**84%**	**69%**
		OlympicCoder 32B	92	23	30	39/0	5	23%	0% (-18%)
	4 example pairs	GPT_4.5	61	54	7	0	6	**80%**	**57%**
		DeepHermes 3 Llama 3 8B	64	47	12	2/3	13	61%	9%
		DeepSeek V3	40	24	0	12/4	28	35%	0% (-90%)
		QwQ 32B	88	51	0	0/37	13	50%	0% (-13%)
		OlympicCoder 32B	92	37	16	39/0	8	37%	5%

syntactic and structurally consistent, making it more straightforward than the others. The transformation mappings (e.g., Table to Class, Column to Attribute) follow well-established, rule-based patterns, allowing language models to effectively generalize transformation logic from minimal training examples. Our results confirm that, even with only one example pair provided, the models demonstrate remarkable accuracy, with GPT-4.5, DeepSeek V3, and OlympicCoder 32B all achieving 100% in both correctness and weighted success. Among the five models, DeepSeek V3 consistently outperforms the others across all configurations, maintaining perfect scores. GPT-4.5 and OlympicCoder 32B follow closely, though their performance occasionally fluctuates due to the generation of extraneous lines in the output. Conversely, DeepHermes 3 and QwQ 32B underperform across configurations, often generating irrelevant or incomplete elements, which negatively impact both correctness and weighted success.

a) One example pair : All models demonstrate near-perfect performance, with GPT-4.5, DeepSeek V3, and OlympicCoder 32B each achieving exactly 100% correctness and 100% weighted success, while QwQ 32B and DeepHermes 3 closely follow with 91% for both metrics but fail to reach a perfect result due to additional lines. This indicates a strong immediate performance from a single example, even across different model architectures.

b) Two example pairs : The models continue to maintain high accuracy. GPT-4.5 achieves 90% correctness, and 80% for weighted success. This is lower than the previous configuration because of additional and incorrect lines. OlympicCoder, DeepSeek V3, and QwQ32B are all maintaining their high performance, matching their previous results for both correctness for weighted success. DeepHermes 3 also achieves the same 91% correctness, but drops slightly in weighted success at 89% by missing several features compared to the ground truth model.

c) Four example pairs : DeepSeek V3 leads again with 100% in both correctness and weighted success, while other models fluctuate. GPT-4.5 shows consistency while OlympicCoder drops a bit due to additional lines, with both models achieving 90% in correctness and weighted success. DeepHermes 3 performs slightly lower with its correctness and weighted success are now at 87% and 82%. QwQ32B experiences a noticeable decline in weighted success (down to 68%) despite an acceptable correctness score at 81%, most of these coming from the more missing features and lines of the models within this configuration.

B. UML-to-Java

This scenario examines the transformation of UML class models into Java, a transformation scenario that bridges high-level design abstractions with executable representations. This task introduces additional complexity related to language-specific syntax, boilerplate generation, and preserving the semantic integrity of the original model in a compilable format. The results highlight the challenges LLMs face in generating syntactically correct and semantically faithful code. GPT-4.5 performs exceptionally well with a single example pair, achieving 100% correctness and weighted success. However, its performance significantly declines as more examples are introduced, dropping to 0% weighted success in the four-example configuration, primarily due to missing features and semantic inconsistencies. In contrast, smaller models such as DeepSeek V3, QwQ 32B, and OlympicCoder 32B demonstrate more stable and scalable performance across configurations, with only minor issues arising from unnecessary or extraneous code lines.

a) One example pair : GPT-4.5 demonstrates exceptional performance with perfect results (100%) for both correctness and weighted success. DeepSeek v3 and QwQ32B also perform strongly, with both scoring 90% for correctness and 81% for weighted success; they fail to achieve a perfect result due to additional lines. OlympicCoder follows closely with 88% correctness and 77% weighted success due to the same problem. DeepHermes dips slightly behind due to a huge number of missing features and incorrectness in the output model, resulting in the model only achieving 79% correctness and 53% weighted success.

b) Two example pairs : GPT-4.5's performance experiences a significant drop (43% correctness, 0% WSuccess), stemming from the incorrectness and additional lines of the output model. Meanwhile, DeepHermes 3 shows a strong recovery with correctness increases to 88% and 76% for weighted success; the model no longer has any incorrectness, but still has harmful additional and missing features. DeepSeek V3 remains consistent, mirroring its previous scores (90% and 81%) while QwQ32B drops slightly to 88% correctness and 76% weighted success this time by having missing features. OlympicCoder's correctness and weighted success improve marginally to 90% and 81% respectively, thanks to a lower number of harmful additional lines compared to the previous configuration.

c) Four example pairs : GPT-4.5 remains affected by scalability issues, scoring only 57% correctness and 0% for weighted success, this low number coming from the high number of missing features, harmful additional lines, and incorrectness. Conversely, the other models, DeepSeek V3 and OlympicCoder 32B, exhibit remarkable consistency, and QwQ 32B also achieves the same result, this time with all of them remaining stable at 90% correctness and 81% weighted success. DeepHermes 3 slightly improves 89% correctness and 75% weighted success thanks to no longer having incorrectness and additional features.

C. SysML-to-AAS

This scenario, which involves transforming SysML into AAS models, is the most complex among those evaluated. It requires a deep semantic understanding that goes beyond simple syntactic mapping, posing a significant challenge for LLMs. The results underscore the intrinsic difficulty of the SysML-to-AAS scenario, as it demands robust structural reasoning and semantic grounding capabilities that many

LLMs struggle to demonstrate consistently. OlympicCoder 32B achieves the highest performance in the single example configuration, with 93% correctness and 85% weighted success. However, GPT-4.5 stands out for its overall stability and scalability across all configurations. In contrast, the other models exhibit erratic behavior, particularly as the number of example pairs increases, leading to higher rates of errors, extraneous content, and omissions. DeepHermes 3, in particular, consistently delivers the lowest performance across scenarios.

a) One example pair : GPT-4.5 and OlympicCoder 32B lead in the performance, with each achieving 88% and 93% correctness and 73% and 85% weighted success, respectively. Both models generalize well with no incorrectness, but only minimal missing lines. OlympicCoder 32B attains the highest weighted success (85%), showing strong performance while using only a single pair of examples. In contrast, DeepHermes 3 and QwQ32B have the lowest correctness and weighted success, with only 43% and 2% for DeepHermes 3 and 63% and 27% for QwQ32B. DeepHermes 3 suffers heavily from a large number of harmful and non-harmful additional lines and other problems, such as 3 incorrectness and 2 missing features, while QwQ 32 B's low performance results from a large number of incorrectness. Meanwhile, DeepSeek V3 gets 72% in both correctness and weighted success due to non-harmful additional lines.

b) Two example pairs : In this configuration, the results are mixed. GPT-4.5 maintains a solid but slightly degrading performance with 83% correctness and moderate 63% weighted success due to a non-harmful additional line. QwQ 32B improves in correctness to 84% and reaches 69% weighted success thanks to a lower number of incorrectness compared to the previous scenario. On the other hand, DeepHermes and OlympicCoder show steep declines (44% and 23% in correctness, respectively, and 0% for both weighted success), this coming from the rising number of missing features for DeepHermes and the large number of incorrectness and non-harmful additional lines for OlympicCoder. DeepSeek V3 also drops significantly, scoring only 63% correctness and 0% weighted success due to a high number of missing features.

c) Four example pairs : Overall performance trends downward. GPT-4.5 shows minor degradation (80% correctness and 57% weighted success) due to incorrectness and missing features, but still maintains robustness compared to other models. Meanwhile, OlympicCoder sees a sharp drop at 37% correctness and 5% weighted success because of the high number of incorrectness, additional lines, and missing features. DeepSeek V3 and QwQ32B fall further with 35% correctness and 0% weighted success for DeepSeek and 50% correctness and 0% weighted success for QwQ32B. These two models suffer from a large number of additional and missing lines. DeepHermes 3 slightly improves compared to its previous two configurations, scoring 61% correctness and 9% weighted success, but these are still considered underperformed. DeepHermes also suffers from incorrectness, additional lines, and missing features.

V. Discussion

In this work, we benchmarked five different LLMs across three distinct transformation scenarios, each tested under three configurations, to evaluate their capability in performing MTBE. Our core intuition was that MTBE could serve as an effective paradigm to improve the transformation performance of LLMs, an area where traditional prompting techniques have shown limitations [1], [2]. By leveraging concrete example pairs instead of complex transformation rules, MTBE has the potential to reduce the accidental complexity typically associated with MDE, thus making model transformations more accessible to domain experts who are familiar with models but not with transformation languages. This, in turn, can improve the overall effectiveness of the development process by simplifying the transformation workflow.

Our collected results provide evidence that LLMs, particularly GPT-4.5, DeepSeek V3, and OlympicCoder 32B, perform well with only a single example pair, achieving results that surpass the transformation methods using LLMs as reported by previous works [1], [2]. Specifically, the structured scenarios (RDBMS-to-UML and UML-to-Java) showed high initial accuracy, demonstrating the LLMs' capability for effective few-shot learning in well-defined transformation tasks.

A critical insight from our study is that, contrary to expectations, providing more example pairs to LLMs does not always result in improved performance. In fact, our data indicate a notable performance degradation in complex semantic transformations like SysML-to-AAS and in syntactically intricate tasks such as UML-to-Java when multiple examples were used. This decline in performance might be due to cognitive overload caused by conflicting transformation patterns or confusion arising from variations in the prompt structure, challenging the models' ability to generalize effectively.

Specifically, in the SysML-to-AAS scenario, even sophisticated models like GPT-4.5 exhibited considerable variability and instability when scaling beyond a single example pair, likely due to difficulties in semantic grounding and structured alignment. However, smaller or fine-tuned models like DeepSeek V3 and OlympicCoder 32B demonstrated more robust performance, especially in syntax-driven scenarios, highlighting the effectiveness of targeted fine-tuning and architectural optimization.

The observed drop in performance with additional examples suggests that improving LLM-assisted model transformation capabilities requires careful prompting rather than extensive example sets. In general, LLMs demonstrate high potential for MTBE, particularly in structured and syntactically regular domains. However, their effectiveness diminishes in semantically complex or multi-pattern scenarios, especially when overloaded with examples. GPT-4.5 remains the most capable model overall but shows signs of scaling instability. Smaller models often exhibit more stable behavior, especially when properly aligned with the task domain.

While our research setup may appear similar to few-shot learning, we frame it as MTBE to emphasize the concep-

tual methodological differences. Few-shot learning involves providing natural language tasks accompanied by labeled input/output examples, often with explicit instructions or task formulations [15], [16]. In contrast, our approach operates more implicitly and is structurally similar to MTBE: the LLM is presented with source-target model pairs, without any explicit labeling or natural language explanation of transformation rules, but initial abstract transformation rules in machine-readable XML format, and lets the LLM infer the rest. The goal is not to follow annotated examples, but to induce transformation logic by recognizing and generalizing from structural correspondences across the models. Moreover, MTBE's scope is more specialized as it assumes the presence of metamodel-conformant source and target models, and the task is framed within the semantics of MDE.

Our results provide an initial indication that LLM-based MTBE may offer a better success rate compared to related works. For instance, Kazai et al. [1] explored model UML class diagrams to Java model transformation using a zero-shot approach with ChatGPT-4, achieving 72% cumulative success rate within the first iteration. However, their performance sharply dropped to 17% for the high-complexity scenario. In contrast, our structured MTBE setup, where representative source-target model pairs are included as examples, appears more robust. In the same UML-to-Java scenario, in our experiments, DeepSeekV3 and OlympicCoder 32B maintained consistent performance as complexity increased. While GPT-4.5 initially performed strongly with one example pair, its success rate declined with additional pairs. It is important to note that these approaches are not directly comparable due to methodological differences. Kazai et al. evaluate the model-level success using a binary correctness metric aggregated over multiple prompting iterations, while our study employs a weighted evaluation framework that captures partial correctness and penalizes missing or incorrect outputs in a single-shot setting.

Similarly, a prior work by Zahra et al. [2] investigated the use of LLMs for transformation scenarios involving domain-specific languages, such as from UML state diagrams to Rebeca models. Their few-shot prompting approach differs from our MTBE-based approach, which relies on structured model pairs to guide the transformation. Nonetheless, their results provide a reference point showing the difficulty LLMs face in less-known modeling languages. The best result achieved by GPT-4 in that study was 85% Cr and 69%. In comparison, our SysML-to-AAS transformation task, also involving domain-specific languages, produced higher and more consistent performance, particularly in early configurations, although some models exhibit noticeable decline with increasing examples.

Based on our findings, LLM-based MTBE shows strong promise in increasing the success rate of model transformation tasks, even without fine-tuning or task-specific training. Its ability to operate effectively in structured and syntactically regular transformation scenarios demonstrates the potential of LLMs as accessible, general-purpose assistants for MDE.

However, several challenges remain that currently prevent LLMs from being used in isolation for transformation tasks. These include reduced performance in underexplored or highly domain-specific scenarios, limited reproducibility due to output variability across identical prompts, and difficulties in the pathway toward integrating LLMs into hybrid transformation workflows, where automation is augmented by human oversight or formal validation, to enhance productivity while preserving correctness and reliability.

A. Threats to Validity

We acknowledge several threats to the validity of our findings, which we organize according to the widely adopted framework of construct, internal, external, and conclusion validity [17].

Regarding construct validity, our evaluation relies on correctness and weighted success metrics derived through manual comparison of the generated models against reference models. While efforts were made to ensure consistency, this inherently subjective assessment introduces the possibility of evaluator bias or interpretive variability. Moreover, our validation approach focuses on structural and semantic equivalence with handcrafted reference models, which may reflect implicit assumptions not necessarily aligned with broader industry practices or community standards.

One threat to internal validity stems from the high sensitivity of LLM outputs to prompt formulation, formatting, and the structure of provided example pairs. Minor changes in prompt phrasing or example configuration can lead to substantial variability in the generated outputs, thereby reducing the consistency and reproducibility of results. This issue was particularly evident with GPT-4.5, whose performance deteriorated significantly when additional example pairs were introduced, highlighting potential scalability and generalization limitations of LLM-based MTBE as task complexity increases. Furthermore, given that the evaluated LLMs were primarily pretrained on general textual and programming corpora, rather than domain-specific transformation datasets, their learned representations may reflect biases toward generic programming constructs or informal language patterns. Such biases could compromise the internal validity of our findings by undermining the fidelity of LLMs in capturing precise, formal transformation logic.

Regarding external validity, our experiments were confined to three specific transformation scenarios, each characterized by distinct data properties and domain-specific contexts. For instance, the SysML-to-AAS dataset, adapted from the work of Ferko et al. [14], is rooted in assumptions specific to digital twin applications and may not generalize to other SysML-based transformations or domains. Similarly, the manually constructed UML-to-Java and RDBMS-to-UML datasets were tailored for controlled evaluation and may not capture the full complexity, heterogeneity, and variability found in real-world industrial model transformation tasks. Consequently, the performance outcomes observed for the evaluated LLMs may not directly translate to broader transformation settings, alternative modeling languages, or industry-scale use cases, thereby limiting the generalizability of our findings.

Finally, with respect to conclusion validity, although the MTBE process was executed systematically across 45 configurations (5 LLMs × 3 configurations × 3 transformation tasks), the sample size may still be insufficient to support strong statistical inferences or detect subtle performance differences with high confidence. As such, conclusions about the relative superiority or inferiority of specific models should be interpreted with caution, recognizing the influence of random variance and contextual factors. Moreover, our reliance on manual evaluation introduces additional uncertainty, potentially affecting the reproducibility and robustness of our findings. These limitations underscore the need for broader experimentation and objective evaluation frameworks in future work.

VI. Conclusion and Future Work

To the best of our knowledge, this is the first comprehensive benchmarking study evaluating LLMs for MTBE across three structurally diverse transformation tasks. Our results confirm that LLMs, particularly GPT-4.5 and OlympicCoder 32B, hold substantial promise for automating model transformations with minimal human effort. While structured transformations like RDBMS-to-UML achieved near-perfect results, more semantically intensive tasks, such as SysML-to-AAS, revealed notable performance inconsistencies and scaling challenges. These findings illustrate the transformative potential of LLMs in engineering MDE workflows, while also surfacing important limitations related to prompt saturation, generalization boundaries, and syntactic fidelity.

Building on these findings, future work should explore hybrid strategies to improve robustness. One promising direction is the design of ensemble systems, where multiple LLMs contribute through voting or consensus-based outputs, dynamically prioritized based on context, model confidence, or past accuracy. Additionally, structured pipelines could orchestrate LLMs in a multi-stage transformation process, e.g., parsing, reasoning, and generation, supported by feedback loops and validation layers. Co-operative processing between models using chain-of-thought or inter-model communication protocols may also enhance semantic grounding and transformation consistency. Another research avenue involves integrating retrieval-augmented generation techniques and enriching prompts with domain-specific metadata. This could improve translation accuracy, especially in semantically rich domains. Fine-tuning LLMs on transformation-specific datasets, especially those involving actor-based languages or formal verification targets, may yield further improvements.

To advance toward scalable, intelligent transformation agents, we envision LLM-based assistants that continuously observe modeling activity, learn from transformation history, and autonomously generate and refine models. Such agents could enhance productivity, ensure compliance with domain-specific constraints, and support real-time model validation. Expanding this approach to accommodate various modeling languages, e.g., BPMN, SysML v2, ArchiMate, would also improve its generalizability and industrial applicability.

VII. Data Availability

We have made a public replication package available at the following repository: https://zenodo.org/records/15703812

References

[1] G. Kazai, R. A. Osei, A. Bucaioni, and A. Cicchetti, "Model transformations using llms out-of-the-box: Can accidental complexity be reduced?" in *First Workshop on Large Language Models For Generative Software Engineering*, June 2025. [Online]. Available: http://www.es.mdu.se/publications/7201-

[2] Z. Moezkarimi, K. Eriksson, A. A. Johansson, A. Bucaioni, and 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*. [Online]. Available: http://www.es.mdu.se/publications/7130-

[3] P. Mohagheghi, W. Gilani, A. Stefanescu, M. A. Fernandez, B. Nordmoen, and M. Fritzsche, "Where does model-driven engineering help? experiences from three industrial cases," *Software & Systems Modeling*, vol. 12, pp. 619–639, 2013.

[4] A. Bucaioni, H. Ekedahl, V. Helander, and P. T. Nguyen, "Programming with chatgpt: How far can we go?" *Machine Learning with Applications*, vol. 15, p. 100526, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666827024000021

[5] D. Varró, "Model transformation by example," in *Model Driven Engineering Languages and Systems*, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 410–424.

[6] G. Kappel, P. Langer, W. Retschitzegger, W. Schwinger, and M. Wimmer, *Model Transformation By-Example: A Survey of the First Wave*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 197–215. [Online]. Available: https://doi.org/10.1007/978-3-642-28279-9{_}15

[7] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," 2023. [Online]. Available: https://arxiv.org/abs/2307.03109

[8] I. Ozkaya, "Application of large language models to software engineering tasks: Opportunities, risks, and implications," *IEEE Software*, vol. 40, no. 3, pp. 4–8, 2023.

[9] N. Gong, C. K. Reddy, W. Ying, H. Chen, and Y. Fu, "Evolutionary large language model for automated feature transformation," 2024. [Online]. Available: https://arxiv.org/abs/2405.16203

[10] S. Arulmohan, M.-J. Meurs, and S. Mosser, "Extracting domain models from textual requirements in the era of large language models." IEEE Press, 2023, p. 580–587. [Online]. Available: https://doi.org/10.1109/MODELS-C59198.2023.00096

[11] A. Rajbhoj, T. Sant, A. Somase, and V. Kulkarni, "Leveraging generative ai for accelerating enterprise application development: Insights from chatgpt," in *2024 31st Asia-Pacific Software Engineering Conference (APSEC)*, 2024, pp. 412–421.

[12] R. Clarisó and J. Cabot, "Model-driven prompt engineering," in *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2023, pp. 47–54.

[13] M. Chochlik, J. Kostolny, and P. Martincova, "Metamodel describing a relational database schema," *Central European Researchers Journal*, vol. 1, no. 1, pp. 94–102, 2015.

[14] E. Ferko, L. Berardinelli, A. Bucaioni, M. Behnam, and M. Wimmer, "Towards interoperable digital twins: Integrating sysml into aas with higher order transformations," in *3rd International Workshop on Digital Twin Architecture (TwinArch) and Digital Twin Engineering (DTE)*, June 2024. [Online]. Available: http://www.ipr.mdu.se/publications/6929-

[15] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," 2021. [Online]. Available: https://arxiv.org/abs/2107.13586

[16] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[17] D. Cruzes and L. Ben Othmane, *Threats to Validity in Empirical Software Security Research*, 11 2017, pp. 275–300.