Evaluating Large Language Models in Exercises of UML Class Diagram Modeling

Daniele De Bari, Giacomo Garaccione, Riccardo Coppola, Luca Ardito, Marco Torchiano
first.last@polito.it
Politecnico di Torino
Torino, Italy

Abstract

Large Language Models (LLM) have rapidly affirmed in the latest years as a means to support or substitute human actors in a variety of tasks. LLM agents can generate valid software models, because of their inherent ability in evaluating textual requirements provided to them in the form of prompts.

The goal of this work is to evaluate the capability of LLM agents to correctly generate UML class diagrams in activities of Requirements Modeling in the field of Software Engineering. Our aim is to evaluate LLMs in an educational setting, i.e., understanding how valuable are the results of LLMs when compared to results made by human actors, and how valuable can LLM be to generate sample solutions to provide to students.

For that purpose, we collected 20 exercises from a diverse set of web sources and compared the models generated by a human and an LLM solver in terms of syntactic, semantic, pragmatic correctness, and distance from a provided reference solution.

Our results show that the solutions generated by an LLM solver typically present a significantly higher number of errors in terms of semantic quality and textual difference against the provided reference solution, while no significant difference is found in syntactic and pragmatic quality.

We can therefore conclude that, with a limited amount of errors mostly related to the textual content of the solution, UML diagrams generated by LLM agents have the same level of understandability as those generated by humans, and exhibit the same frequency in violating rules of UML Class Diagrams.

CCS Concepts

• **Software and its engineering → Requirements analysis**; • **Computing methodologies → Natural language processing**.

Keywords

Software Modeling, Class Diagrams, Large Language Models, Artificial Intelligence

ACM Reference Format:
Daniele De Bari, Giacomo Garaccione, Riccardo Coppola, Luca Ardito, Marco Torchiano. 2024. Evaluating Large Language Models in Exercises of UML Class Diagram Modeling. In *Proceedings of the 18th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24), October 24–25, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3674805.3690741

ESEM '24, October 24–25, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-1047-6/24/10
https://doi.org/10.1145/3674805.3690741

1 Introduction

In the ever-evolving landscape of Software Engineering, the design phase of the software process is still one of the most crucial to ensure the quality of the final product [7]. In such a phase, in fact, the foundation of the software product is conceptualized, designed, and documented, setting the foundation for the subsequent development process. Among the most utilized instruments for this phase is the Unified Modeling Language (UML), a standardized visual language that has been largely adopted to make clear and concise documentation of software design activities. More specifically, UML class diagrams are a vital tool in defining the structure of the software system, offering a visual representation of classes, their attributes, methods, and the relationship between them. However, the realization of accurate and comprehensive class diagrams is a challenging task, especially during the requirement-gathering stages, due to the complexity of translating abstract requirements into detailed visual models which demands a deep understanding of the system's domain and a good knowledge of UML's syntax and semantics.

The rapid diffusion in the past years of Large Language Models (LLMs) in several domains hints at a possible solution to this challenge. LLMs, thanks to their deep learning algorithms and big training datasets, have shown proficiency in understanding and generating human-like text, opening the possibilities for automation in tasks that traditionally needed human expertise. In Software Engineering, the potential of LLMs to automate the generation of UML class diagrams is particularly intriguing, because it could enhance efficiency, and reduce errors in the design process, freeing human designers that, in this way, can focus on more strategic aspects of system development [9]. However, in the current state of the art, the application of LLMs in generating UML class diagrams has not been thoroughly explored, leaving a gap in the understanding of their effectiveness, especially if compared to traditionally human-driven methods.

The work described in this paper aims to assess the applicability of LLMs in generating UML class diagrams, aiming to evaluate whether LLMs can support or potentially enhance the traditional manual practices of UML diagram generation. The investigation is structured around seeking to compare the syntactic, semantic,

and pragmatic qualities of LLM-generated diagrams with those produced by humans, and to assess LLM's ability to replicate domain-specific knowledge within these diagrams. To that extent, we compared several quality aspects of UML diagrams generated by LLM agents and by human practitioners.

The experiment has been conducted with an educational connotation, i.e., the considered natural language requirements are part of educational programs for Software Engineering courses. The rationale for this decision is two-fold: first, to conduct a smaller-scale exploration in educational settings instead of assessing LLM agents directly on real-world large-scale software artefacts; second, to evaluate the capability of LLM agents to aid docents of Software Engineering and modeling courses, in providing sample diagrams and assist students with reference examples for software modeling.

The remainder of the paper is structured as follows: Section 2 contains a discussion regarding the state-of-the-art about the utilization of Large Language Models in software modeling; Section 3 describes the methodology applied in the current work; Section 4 describes the results of the experimentation; Section 5 discusses the results, along with the threats to the validity of the current research; Section 6 concludes the paper and provides future research direction.

2 Background and Related Work

Utilizing Large Language Models for requirement gathering in software engineering is an area which is being extensively explored in recent literature.

As highlighted by Belzner et al. [1], LLMs possess the ability to recognize and outline the fundamental components within a system. These fundamental components, known as core entities, constitute the primary elements that form the system. Understanding these entities and their interactions is pivotal for the efficient design and advancement of software systems.

Utilizing LLMs to identify and define core entities involves leveraging their capacity to comprehend and interpret natural language descriptions of a system. By providing a comprehensive description of the system's attributes, functionalities, and behaviours, an LLM could potentially highlight the central entities based on factors like their frequency of mention, contextual relevance, and significance in achieving the system's functionalities.

Moreover, LLMs can generate relational data models from contextual descriptions, which can be represented visually using domain-specific languages such as mermaid.js. Additionally, LLMs are capable of accommodating various levels of abstraction in data models, similar to those observed in ELT or ETL pipelines.

In their literature review from 2023 [3], Fan et al. analyze the principal application areas of Large Language Models in the Software Engineering discipline. Among the main areas for LLM use in SE, the authors highlight the existence and feasibility of design diagrams through automated generation by LLM-based agents, whilst identifying code generation, code documentation and software testing as the main areas for LLM application. The authors highlight as a result of their investigation the need for the definition of hybrid techniques, involving traditional human-based SE aided by LLM-based agents, to ensure reliability in software engineering tasks.

Table 1: GQM Template for the study

Object of study	Usage of LLM agents
Purpose	Comparing
Focus	Effectiveness and correctness
Context	Definition of UML Class Diagrams
Stakeholders	Software Modeling Instructors

Hirtreiter et al. [5] document the application of Large Language Models in the process of piping and instrumentation diagrams (P&IDs), using Language Models to implement control structures into Process Flow Diagrams (PFDs). With the use of a pre-trained model, they measure an accuracy of 89.2% over 100,000 generated PFDs, thereby confirming the potential of AI-assisted process engineering.

LLMs have also been evaluated by related literature in the field of education, and have been investigated in terms of the opportunities and challenges provided in that field [6]. One application in which ChatGPT has been evaluated is, as an example, the possibility to generate natural language assignments for students, as investigated by Xiao et al. for reading exercises in high school [13] and by Sarsa et al. for programming exercises [10]. To the best of our knowledge, no previous studies are available in the literature to evaluate the capability of LLM agents to solve existing exercises in compatibility with reference solutions, in the field of Software Modeling.

3 Experimental design

We designed an experiment to evaluate the feasibility of LLM-based construction of UML Class Diagrams starting from Natural Language requirements.

3.1 Experiment goal

We report the design, goal, research question and procedure by following the GQM template as summarized in Table 1. The goal of the experiment can be described as: *Analyze the usage of LLM agents for the purpose of comparing the quality and correctness of generated UML Class Diagrams from natural language requirements, from the point of view of instructors of Software Modeling.*

3.2 Research Questions and Metrics

We define the following set of research questions to frame the experiment design:

- **RQ1:** Does the use of LLM agents have an impact on the quality and correctness of UML Class Diagrams generated from natural language requirements?
- **RQ2:** What aspects of the natural language requirements have an impact on the quality of generated UML Class Diagrams?

To measure the quality of generated class diagrams, we focus on the quality aspects defined by Bolloju et al. [2]:

- *Syntactic Quality:* assess if the class diagrams follow the syntactic structure of UML Class Diagrams. The rules verified for syntactic quality are:
  - Missing cardinality details;
  - Inappropriate naming of classes and associations;

  - Incorrect use of UML symbols.
- *Semantic Quality:* evaluate the accuracy and completeness of the diagrams in representing the intended domain. It is measured in terms of: (i) Validity: all elements and relationships in the diagrams should accurately represent the domain; (ii) Completeness: the diagrams should include all necessary elements and relationships. The rules verified for semantic quality are:
  - Incorrect Cardinality;
  - Aggregation in place of association;
  - Wrong location of attributes or operations;
  - Operations cannot be realized using existing attributes and relationships.
- *Pragmatic Quality:* focus on the understandability of the diagrams from the perspective of stakeholders. The rules verified for pragmatic quality are:
  - Redundant attributes and associations;
  - Specialization with no distinction among subclasses;
  - Inconsistency in styling and conventions.

It is worth underlining that the three metrics described above are *intrinsic* for any Class Diagram, i.e. they are not evaluated in comparison with a reference solution but are computed against a set of general rules applicable to every diagram. For each of the three dimensions, we count the number of errors in each produced or generated diagram.

To measure the *correctness* of the generated diagrams against the natural language requirements provided, we focus on a comparison of the semantic content of elements and relationships in the diagrams. The elements that are analyzed to this end are classes and interfaces, class attributes, methods, and relationships between elements.

To this end, we adopt the approach defined by Nikiforova et al. for the comparison of class diagrams based on semantical features of their elements [8]. The semantical distance defined by Nikiforova et al. is based on the computation of the length of four distinct distance vectors, analyzing the differences between the diagram under analysis and a reference (solution) diagram based on four main aspects: classes and interfaces, class attributes, methods, relations between elements. The highest the resulting distance, the more distant is the proposed solution from the reference solution; a distance equal to 0 indicates a diagram that is equal to the reference solution.

To evaluate the aspects of the natural language requirements that may have an influence on the generated solutions, we take into consideration three different aspects:

- *Size:* the size of the reference solution, in terms of the number of classes, attributes and methods, and associations;
- *Estimated Difficulty:* an estimation of the difficulty of the exercises expressed in a Likert Scale (1-5). The evaluation of the difficulty of the exercises was made by three authors of this paper independently and then averaged, before any attempt at generating the corresponding UML diagram.
- *Readability:* the readability of the natural language exercise description was measured by computing the Flesch-Kincaid Ease Score [11], a readability metric used to determine how difficult a text is to understand based on the length of words and sentences in the text. We resorted on an online tool to measure this metric for each exercise[1].

Table 2: Details about the selected exercises

Exercise	Classes	Attr+Ops	Assoc	AVG ED	FK
1	6	17	7	2.33	45.9
2	6	35	5	1.33	55.2
3	7	12	8	4.67	67.5
4	6	31	7	3.33	41.1
5	8	18	11	2.33	48.4
6	9	12	12	3.67	66.4
7	6	5	7	1.00	58.6
8	7	11	8	2.00	49.4
9	9	13	10	2.67	54.7
10	7	15	7	2.67	59.1
11	6	11	9	3.33	55.4
12	6	11	5	2.33	53.8
13	8	19	8	1.67	58.0
14	5	12	5	2.67	59.6
15	8	6	8	4.33	74.5
16	11	13	13	3.00	60.0
17	8	5	8	2.33	53.1
18	6	10	6	1.67	60.9
19	9	20	9	4.00	54.5
20	10	28	10	4.67	60.8

3.3 Collection of experimental material

For the analysis conducted in this paper, 20 exercises were collected from several online sources. The exercises are natural language descriptions of different domains, with the purpose of giving different representations of the challenges that can be encountered in the realization of UML class diagrams.

The exercises were collected by one author of this paper from various educational online sources, by applying the following inclusion criteria: (i) the exercises were in a language understandable by the author (English or Italian); (ii) the exercises were provided with a reference solution.

The details about the 20 exercises are reported in Table 2. The interested reader can find all the natural language descriptions of the exercises in an online appendix[2].

3.4 Experiment Conduction

To conduct the experiment, it was required to obtain for each natural language description of a problem (i.e., an exercise) a human-generated solution and an LLM-generated solution.

All the natural language description of systems were translated into UML Class Diagrams by one of the authors of this paper, a master's student, during the work towards his master's thesis. The diagrams were made by using the Microsoft Visio tool.

The following rules were followed while drawing the UML Class Diagrams:

- Time limitation: the time given to complete a UML class diagram is 30 minutes. This is necessary to evaluate the ability to work under time pressure and to avoid prolonged deliberation.
- No access to external sources: to ensure that the diagram is made only with personal knowledge, access to the internet,

[1] https://goodcalculators.com/flesch-kincaid-calculator/
[2] http://doi.org/10.6084/m9.figshare.25434550

Table 3: Maximum, Minimum, Average metrics for human and LLM actors

	Human			LLM		
	Min	Max	Mean (SD)	Min	Max	Mean (SD)
Sem. Errors	0	7	1.75 (1.80)	0	14	4.85 (3.28)
Syn. Errors	0	4	0.5 (1.14)	0	2	0.9 (0.79)
Prag. Errors	0	4	1.1 (1.12)	0	3	1.6 (0.94)
Distance	0.87	10.20	5.01 (2.36)	2.24	11.54	7.25 (2.86)

notes, textbooks, or any other external sources was prohibited during the execution of the exercise.
- No interaction with others: the exercises were completed individually, without help from other people.

The constraints for the UML class diagram exercise were carefully chosen to create a challenging yet fair environment that mimics real-world conditions.

To realize the UML class diagrams the Large Language Model chosen was ChatGPT-4, which is the latest version of OpenAI's generative pre-trained transformer series.

For this research, the prompt used is the same for all the exercises and it is *"Create a UML Class Diagram of the given exercise and give me the PlantUML code"*, followed by the text of the exercise to solve. This prompt was selected because in a simple way it ensures that the LLM agent understands the specific task (creating a UML diagram), the context (the provided exercise), and the expected output format (PlantUML code).

After the generation of the PlantUML code by the LLM agent, the online tool PlantText.com is used to generate UML diagrams starting from a PlantUML textual description.

Once the diagrams were generated and visualized with PlantText.com, they were manually inspected and compared with the reference solutions, to compute the quality and correctness metrics previously defined. The human and LLM-generated diagrams, as well as the related quality and correctness measures, are reported in full as an online-only appendix of this manuscript[3].

3.5 Analysis Method

To answer RQ1, we resorted on applying an ANOVA statistical test on the pairs of results (human produced vs. LLM-generated) for each diagram. We considered as independent variable the fact that the diagram was generated by a human actor or by a LLM-agent, and as dependent variables the semantic, syntactic and pragmatic quality of the diagram, as well as as correctness measured against the provided reference solution. The null hypotheses for the statistical test are reported later in the results section (Table 4). Since conducting multiple univariate ANOVAs may increase the chance of Type I error (false positives), we adjusted the significance level using Bonferroni correction to account for multiple comparisons.

Based on the results of RQ1, we aim to check – in case of significantly different qualities of the diagram – whether the size and readability characteristics of the exercises have an influence on the dependent variable. To do so, we first apply a covariance check between the independent size and difficulty variables, and we exclude one variable for each covariant pair. Then we apply a linear regression between all independent size and difficulty factors and dependent quality and correctness factors. For this analysis we are mostly interested in the effect sizes and not on the resulting p-values because of the low statistical power of the prompt. We however perform an analysis of the p-value obtained by applying the linear regression, and apply also in this case the Bonferroni Correction to cope with the increased chance of Type I error.

[3]http://doi.org/10.6084/m9.figshare.25434550

4 Results

In Table 3 we report the maximum, minimum, and average number of errors and distance over the 20 produced exercises for both human and LLM agents. The distributions of such variables are reported graphically in the violin/box plots in Fig. 1.

We can notice that LLM agents committed on average a higher number of errors for all categories (with a maximum amount of errors equal to 14 for Semantic Errors).

The highest difference can be noticed for Semantic Quality (4.85 errors on average against 1.75 made by the human solver), whereas the smallest difference is measured for Syntactic Quality (0.9 errors against 0.5). Regarding Syntactic and Pragmatic quality, it is worth underlying that – while having a higher average number of errors – the LLM agent performed at most, respectively, 2 and 3 errors, against a maximum number of errors equal to 4 for both qualities for the human agent.

The distance against the text-based vectors of the solution was higher for the LLM solver against the human solver (7.25 against 5.01). While the human-generated solutions had a minimum distance from the reference solution of 0.87, the LLM-generated solutions had a minimum distance that was three times higher (2.24). The difference in the maximum distance from the reference solution was smaller (10.20 for the human agent, and 11.54 for the LLM agent). We can thereby conclude that the LLM solver performed worse in any quality aspect with respect to the human solver.

In Table 4 we report the results of the ANOVA test for the dependent variables used to answer RQ1. By analyzing the p-values we can reject the hypotheses $H0_{sem}$ and $H0_{corr}$, meaning that we observe a statistically significant difference in Semantic Quality and Distance with respect to the exercise solution. We cannot instead reject the hypotheses related to Syntactic and Pragmatic quality, underlining thereby that the difference between human and LLM solvers is limited – in the setting of this experiment – for these aspects.

In the dumbbell plot reported in Fig. 2 we compare the distances against the reference solution for the results generated by the human and the LLM solvers. With this comparison, we notice that the LLM solver performed better than the human solver only in two occasions (Exercise 11 and Exercise 17) while obtaining an equal distance in a third case (Exercise 7). The exercises share no similarity in terms of average estimated difficulty and Flesh-Kinkaid ease score, whilst being among those with the smallest-sized reference solutions.

Once noticing a statistically significant difference in Semantic Quality and Correctness between solutions made by a human or by an LLM agent, we perform a linear regression on a series of

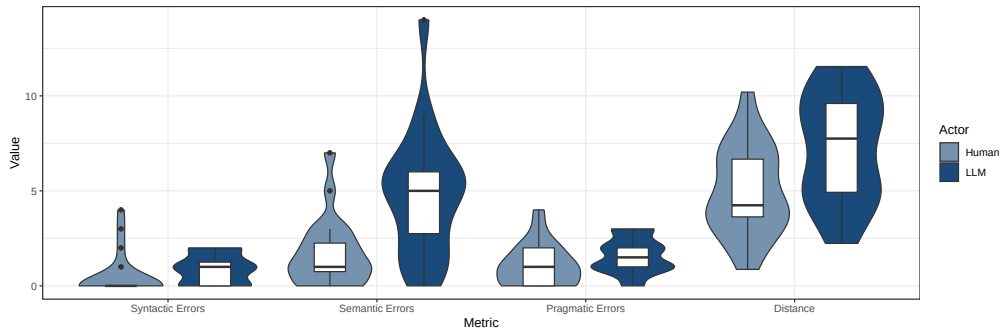

Figure 1: Violin plots for Quality metrics

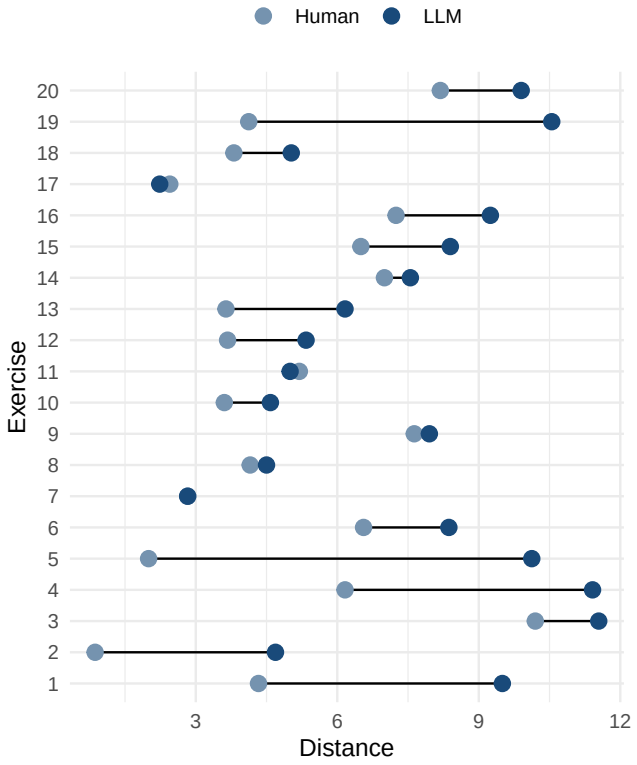

Figure 2: Distances from the reference solution for Human- and LLM-generated solutions

independent variables to understand the impact that they may have on such aspects, considered as dependent variables.

Based on the correlation matrix computed between the independent variables, we exclude the number of *Associations* from our set of independent size variables since it is highly correlated with the *Classes* independent variable (correlation coefficient of 0.85), suggesting that they share a strong linear relationship.

The results of the linear regression performed between each couple of independent and dependent variables are reported in Table 5: we report for each pair the estimate, the standard deviation and the p-value (in bold if below the significance threshold).

Table 4: Null hypotheses for RQ1 and p-values of the applied ANOVA univariate tests (alpha = 0.0125)

Hypothesis	Description	p-value	
H0_syn	The human or LLM-based generation of UML diagrams has no impact on the resulting syntactic quality	0.206	accept
H0_sem	The human or LLM-based generation of UML diagrams has no impact on the resulting semantic quality	**6.7e-4**	**reject**
H0_prag	The human or LLM-based generation of UML diagrams has no impact on the resulting pragmatic quality	0.134	accept
H0_corr	The human or LLM-based generation of UML diagrams has no impact on the correctness of the diagram	**0.0102**	**reject**

Both the semantic errors and distance variables have a positive correlation with the four independent variables, with the exception of human semantic errors and distance against the number of attributes and operations, and the LLM agent distance against the Flesch-Kincaid score.

Regarding the estimates of the impact of each independent variable, we see that the number of classes in the reference solution has an impact on all four dependent variables, albeit not significant in size. This aspect may suggest that the size in terms of classes increases the frequency of errors only slightly, suggesting that both the human and LLM agents are good at capturing the main elements of a concept model even when the number of such elements increases.

The number of attributes and operations (methods) of the reference solution instead has a negative correlation with the number of semantic errors and the distance from the solution for human agents. Our interpretation of this result is two-fold: first, the human agents may have a tendency to capture many attributes from textual documentation; second, the reference solutions can be sketched by reducing the number of attributes inside classes. The combination of these two attitudes leads inevitably to an increase in the distance from the reference solution.

It is worth underlining that the average estimated difficulty (computed as a manual investigation by the authors of the paper) was a decent predictor of the semantic error and distance for both human

Table 5: Results of the linear regression between size and complexity variables and semantic correctness and distance from the reference solution

		Classes			Attr./Ops			Avg. ED			FK		
		estimate	SD	p-value	estimate	sd	p-value	estimate	sd	p-value	estimate	sd	p-value
Human	Semantic errors	0.37	0.25	0.16	-0.08	0.05	0.14	0.83	0.35	0.03	0.13	0.05	0.01
	Distance	0.49	0.33	0.15	-0.03	0.058	0.72	1.67	0.34	**1.0e-4**	0.13	0.07	0.06
LLM	Semantic errors	0.33	0.48	0.50	0.03	0.09	0.78	1.58	0.59	0.01	0.12	0.10	0.23
	Distance	0.63	0.39	0.13	0.13	0.08	0.08	1.84	0.46	**8.1e-4**	-8.0e-4	0.09	0.99

and LLM solvers. More specifically, we measured a significant correlation (p-value < 0.003125 after Bonferroni Correction) between the Human-agent distance, the LLM agent distance and the average estimated difficulty for the exercises. Even though the sample is limited in size and therefore the p-value is not dependable, the estimate provided by the linear regression is much higher than that measured for the number of classes. On the other hand, the Flesch-Kincaid ease score had hardly any influence on the dependent variables. We, therefore, conclude that the evaluation of the difficulty of modeling exercises cannot be performed by using standard textual ease scores, while the evaluation performed by domain experts proves to be dependable.

5 Threats to Validity

We discuss in this section the potential threats to the validity of the study according to the four categories defined by Wohlin et al. [12].

Threats to Internal Validity concern internal factors that may affect a dependent variable that the study did not consider. The internal validity of the study is limited by the solutions provided by the sources of exercises, which are regarded as the primary benchmark for correctness. The reference solutions have inherent potential limitations, e.g. being incomplete, overly simplistic or overly detailed, thus influencing all the correctness metrics measured for the solutions provided by human and LLM solvers. The internal validity of the study is also threatened by the specific prompts used to solve the exercises with the LLM agents, since there is a chance that other prompts would have provided better results in terms of correctness, by favoring specific correctness metrics over others. Potential bias and unreliability of LLM can be an additional factor impacting the results in generating UML Class Diagrams. Finally, all the exercises were solved by a single author of this manuscript, introducing a bias in the correctness of the exercises.

Threats to External Validity concern whether the results of the study can be generalized rather than be applicable only to the specific sample of participants involved. The described study was focused on a sample of UML modeling exercises gathered from online literature and books. Albeit effort was spent to ensure a certain variation of the semantic content and difficulty of the exercises considered, the limited sample of selected artefacts does not ensure that the results are generalisable to any UML modeling exercise.

Threats to Construct Validity concern the extent to which the measures selected for the study actually represent the observed construct. Regarding the constructs used in this study, we resorted to using a set of inherent correctness metrics that are reported in related literature [2][8]. It is indeed possible that such measures are not the optimal choices in evaluating UML diagrams, especially since they are defined based on human solvers and are not tailored for LLM agents.

Threats to Conclusion Validity concern the ability to conclude from the results of the study. The biggest conclusion validity threat of this study is related to the limited size of the sample (only 20 exercises considered). Therefore the conclusions drawn in the present research should be validated with a more extensive set of artefacts.

6 Conclusion and Future Work

In this manuscript, we evaluated the capability of LLM-based agents to solve exercises for the generation of UML diagrams when provided with natural language descriptions of software characteristics in different domains.

The results of this paper highlight that LLM agents still achieve worse results overall in generating UML diagrams from requirements against human solvers. The difference is however limited for several quality and correctness aspects that were evaluated, therefore hinting at a possible utilization of LLM agents for the generation of sample solutions for students or preliminary sketches of diagrams to accompany requirements in software projects. All the results in this study were gathered after a single-pass interaction with LLM agents, therefore there is important room for improvement of the results provided by such agents.

As our future work, we envision the possibility of evolving the basic and static prompts used to generate the diagrams and consider the possibilities of using an architecture of LLM agents cooperating to improve a generated diagram iteratively. This type of architecture, referred to as Cognitive Architecture, is being explored in Software Engineering literature and applied to other fields of the discipline like generation of test cases in web software testing [4].

Moreover, in the context of a separation of tasks and concerns in an architecture of LLM-based agents, we plan to evaluate the capability of agents to generate meaningful and consistent natural language exercises for learners of Software Modeling courses, and the ability in evaluating solutions to existing exercises. In the context of this paper, in fact, all the quality aspects were measured manually by one of the authors; the creation of LLM-based systems able to automatically and reliably evaluate the provided solution to an exercise would be a significant aid for docents in an educational setting.

References

[1] Lenz Belzner, Thomas Gabor, and Martin Wirsing. 2023. Large language model assisted software engineering: prospects, challenges, and a case study. In *International Conference on Bridging the Gap between AI and Reality*. Springer, 355–374.

[2] Narasimha Bolloju and Felix SK Leung. 2006. Assisting novice analysts in developing quality conceptual models with UML. *Commun. ACM* 49, 7 (2006), 108–112.

[3] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. *arXiv preprint arXiv:2310.03533* (2023).

[4] Robert Feldt, Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Towards autonomous testing agents via conversational large language models. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1688–1693.

[5] Edwin Hirtreiter, Lukas Schulze Balhorn, and Artur M Schweidtmann. 2024. Toward automatic generation of control structures for process flow diagrams with large language models. *AIChE Journal* 70, 1 (2024), e18259.

[6] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and individual differences* 103 (2023), 102274.

[7] Mayuram S Krishnan, Charlie H Kriebel, Sunder Kekre, and Tridas Mukhopadhyay. 2000. An empirical analysis of productivity and quality in software products. *Management science* 46, 6 (2000), 745–759.

[8] Oksana Nikiforova, Konstantins Gusarovs, Ludmila Kozacenko, Dace Ahilcenoka, and Dainis Ungurs. 2015. An approach to compare UML class diagrams based on semantical features of their elements. In *The Tenth International Conference on Software Engineering Advances*. 147–152.

[9] Ipek Ozkaya. 2023. Application of large language models to software engineering tasks: Opportunities, risks, and implications. *IEEE Software* 40, 3 (2023), 4–8.

[10] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 27–43.

[11] Marina Solnyshkina, Radif Zamaletdinov, Ludmila Gorodetskaya, and Azat Gabitov. 2017. Evaluating text complexity and Flesch-Kincaid grade level. *Journal of social studies education research* 8, 3 (2017), 238–248.

[12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[13] Changrong Xiao, Sean Xin Xu, Kunpeng Zhang, Yufang Wang, and Lei Xia. 2023. Evaluating reading comprehension exercises generated by LLMs: A showcase of ChatGPT in education applications. In *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*. 610–625.