Date of publication MMM-dd, yyyy, date of current version MMM-dd, yyy
www.computingonline.net/ computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.19.3.XX

Evaluating the Quality of Class Diagrams Generated by GPT-4 Model

KELETSO J. LETSHOLO
Faculty of Computer Information Science, Higher Colleges of Technology, Abu Dhabi, UAE
https://orcid.org/0000-0003-4355-7987, e-mail: kletsholo@hct.ac.ae

ABSTRACT
Automatically generating accurate and comprehensive class diagrams from natural language requirements can minimize human errors, improve accuracy, and streamline requirements analysis. OpenAI's GPT-4 model has made significant strides in this domain. For GPT-4 to gain traction within requirements engineering, the quality of its class diagrams is essential. This study evaluates GPT-4's class diagrams by comparing them to those created by experts and existing tools, using precision, recall, and F1 measures, which reveal significant variability. GPT-4's precision ranges from 0.61 to 0.88, reflecting a varied ability to correctly identify instances. Recall spans from 0.63 to 1.00, indicating differences in capturing all relevant instances. The F1 score, which balances precision and recall, ranges from 0.65 to 0.87, indicating a variety of effectiveness in different contexts. In particular, GPT-4 outperforms existing tools in precision, recall, and F1 score, showcasing its strong aptitude to generate accurate class diagrams from natural language. This paper evaluates the GPT-4 diagrams against expert benchmarks, compares them with four tools, and presents insights into GPT-4's capacity in requirements engineering.

KEYWORDS Generative Pre-trained Transformer (GPT); Requirements Engineering (RE); Natural Language Processing (NLP); Class Diagram.

I. INTRODUCTION

A Class diagram is important in the requirements engineering phase, as it provides a blueprint of the system structure by translating requirements into visual representations. It serves as an analysis model by visually representing the structure of system classes, their attributes, operations, and relationships [1]. It is a form of the Unified Modeling Language (UML) model. This paper focuses on a class diagram for two reasons: it documents the most important concepts in requirements analysis, which are classes, attributes, and relationships, and it is a fundamental analysis technique used in most software development methodologies [2], [3].

The automatic creation of class diagrams or analysis models can greatly enhance productivity in the software development lifecycle. By automating the production of these diagrams, consistency is maintained, and errors are minimized, facilitating the transition from informal requirements described in natural language to formal design components. This process reduces the manual modeling effort, allowing developers and analysts to focus more on high-level design and problem solving tasks. Although automated generation of class diagrams offers significant advantages, it also encounters certain challenges. A key challenge is the interpretation of natural language requirements, which are often ambiguous. Such an ambiguity can lead to misinterpretations and subsequent design inaccuracies. Therefore, the quality of the generated class diagrams is of utmost importance; poor quality diagrams can lead to defective models, which can worsen issues throughout the development lifecycle. Furthermore, it is essential to evaluate the accuracy and completeness of the diagrams produced by these tools to mitigate the risk of error propagation throughout the development process.

Advancements in deep learning have promoted the use of large language models (LLMs) such as BERT (Bidirectional Encoder Representations from Transformers) [4] and GPT (Generative Pre-trained Transformer) [5] to transform natural language requirements (NLRs) into various software artifacts, including class diagrams. An LLM is a deep neural network model that has been trained on large amounts of data, such as books, code, articles, and websites, to learn the underlying patterns and relationships in the language for which it was trained [6]. The GPT model is a specific type of LLM, built using a transformer architecture, and is pre-trained on a diverse dataset in an unsupervised manner. After this pre-training, the GPT model can be fine-tuned on smaller, specific datasets, reducing its dependency on extensive labeled data. The model's training process enables it to generalize effectively across various tasks, making it adaptable to different application scenarios such as requirement engineering. Furthermore, the GPT transformer architecture has the ability to capture context more effectively than traditional models [5], enabling it to automatically extract class diagram elements from implicit requirements. However, the quality of the GPT generated class diagrams is yet to be determined, as there is little to no published research that systematically evaluates the quality of these diagrams in terms of accuracy and completeness. These diagrams are at the start of several subsequent phases in software development; faults not detected at this stage can be costly to fix at later stages. Evaluating the quality of the class diagrams generated by GPT models is important to determine the effectiveness of automated tools in practical software development environments. Additionally, evaluating the performance of GPT models in generating class diagrams can lead to improvements in AI-driven development tools, ultimately enhancing productivity, reducing the likelihood of human error, and speeding up the requirements engineering process. This research evaluates the GPT-4 model, an advanced multimodal system that reaches human-level performance on numerous rigorous professional and academic benchmarks. It exceeds the capabilities of current large language models in multiple natural language processing tasks and surpasses the majority of cutting-edge systems, which frequently depend on task-specific fine-tuning [7].

The purpose of this study is to evaluate the quality of the class diagrams generated by the GPT-4 model by comparing them with those created manually by human experts, as well as with the diagrams produced by existing Natural Language Processing for Requirements Engineering (NLP4RE) tools. NLP4RE tools use natural language processing (NLP) techniques to improve requirements engineering (RE) tasks. To measure the quality of GPT-4 generated class diagrams, precision, recall, and F1 measures are used, and the study findings show notable variability in these scores. The precision ranges from a minimum of 0.61 to a maximum of 0.88, indicating inconsistency in GPT-4's ability to accurately identify relevant instances compared to the total instances it retrieved. Recall values range from 0.63 to a perfect 1.00, highlighting differences in the extent to which GPT-4 retrieves all relevant instances. Meanwhile, the F1 score, reflecting a balance between precision and recall, varies between 0.65 and 0.87, highlighting the diversity in the effectiveness of GPT-4 in different contexts of NLRs. Furthermore, compared to existing NLP4RE tools, the GPT-4 model significantly outperforms these tools in precision, recall, and the F1 score, indicating a robust ability to generate accurate and comprehensive class diagrams from NLRs. The contributions of this paper are threefold. First, it presents an evaluation of the quality of GPT-4 generated class diagrams by comparing them to reference diagrams created manually by human experts. Second, it provides an evaluation of how the GPT-4 generated diagrams compare to those produced by the four existing NLP4RE tools. Third, it offers insight to the requirements engineering community to understand how well the GPT-4 model can generate class diagrams from NLRs.

The remainder of the paper is organized as follows. Section II, reviews previous research focused on evaluating the performance of GPT-based models in generating analysis models. Section III, Methodology, discusses the processes, tools, and comparative strategy used when evaluating the quality of generated class diagrams. Section IV, Results presents the findings of the comparative analysis. Section V, Discussion, provides an in-depth analysis of the results, highlighting the implications for software development practices. Section VI, identifies potential limitations in the design, execution, or interpretation of this study. Finally, Section VII, Conclusion, summarizes the key findings of the study, reflecting on the research objectives and contributions.

II. RELATED WORK

De Bari et al. [8] conducted a systematic evaluation of LLMs, including those based on GPT, in UML class diagram modeling exercises. Their research highlights how LLM performance varies depending on prompt configuration and evaluation scenarios. They achieved human-level performance in certain tasks. In contrast, this study directly compares the generated diagrams with expert benchmarks using precision, recall, and F1 metrics. Unlike De Bari et al. [8], who focus on educational settings and qualitative analyzes, this paper uses expert-designed benchmarks and traditional information retrieval metrics. This approach provides a unique perspective that is more aligned with industry practices and automated tool evaluation.

Extended evaluations in other domains, such as those involving ChatGPT, have assessed its capabilities and limitations in software modeling [9]. Although LLMs have shown significant progress in code generation, their effectiveness in producing UML class diagrams and assisting modeling tasks is limited by syntactic and semantic complexities, inconsistent responses, and scalability challenges. Researchers advocate for incorporating LLMs into model-based systems engineering (MBSE) to potentially enhance the societal impact of MBSE. Marques et al. [10] analyzed the role of LLMs, such as ChatGPT-3.5, in software requirements engineering, using a comparative analysis. Their study highlights ChatGPT's efficiency in eliciting requirements and its accuracy in capturing user needs, improving communication among stakeholders. Research by Speth et al. [11] investigates ChatGPT's potential in creating software engineering modeling exercises using UML class and sequence diagrams, showcasing promise in educational settings. The study examines ChatGPT's interpretation of UML diagrams, both graphically and textually, to develop customized exercises that improve conceptual understanding in software engineering education. Although ChatGPT excels with textual UML diagrams, it struggles with graphical UML accuracy, producing quality comparable to earlier models like the GPT-3.5-turbo. Manual review remains necessary for optimal exercise creation. Ronanki et al. [12] explored the potential of ChatGPT to generate requirements and compared its output with that of human RE experts from academia and industry. The results suggest that ChatGPT can support requirements-elicitation processes by transforming raw requirements into high-quality specifications. Further exploration of ChatGPT's applications in various RE activities is recommended to leverage LLM capabilities and encourage broader adoption in RE tasks.

Other researchers have considered the applications of LLM-based tools such as ChatGPT in automating various software engineering tasks [13], including code generation [14]–[16] and test generation [17]. However, it is necessary to assess whether AI-based tools offer higher success rates than standard and manual practices [18]. It is crucial to investigate whether AI tools can outperform human experts in delivering valuable results to the requirements engineering community.

A recent systematic review of the literature that examines advances in automated support for RE underscores the predominant use of controlled experiments for tool evaluation [19]. The study concludes that, while there has been substantial academic progress, limited industrial application and comparative evaluations pose challenges that need to be addressed to enhance automation in RE and software development practices. The mapping study conducted by Zhao et al. [20] also highlighted a general lack of evaluation in the results of the NLP4RE research. This mapping study recommends that practitioners and experts in the field of RE must participate in the evaluation of the results of the NLP4RE tools. This recommendation aligns with what Yue et al. [21] emphasized: the quality of an automatically generated analysis model should be evaluated by comparing it with one manually developed by human experts to determine how closely the automated analysis model matches the expert solution. However, this is not always feasible because experts often come at a higher cost and are not always accessible [22].

None of the studies explicitly included RE experts in the evaluation of the class diagrams produced by GPT models. Furthermore, some studies did not assess correctness and completeness using standard metrics such as precision, recall, and the F1 score. This indicates areas for improvement in evaluating AI tools for requirements engineering tasks and suggests an opportunity for future research to incorporate RE experts' assessments and quantitative measures.

III. METHODOLOGY

The methodology used in this study follows a systematic workflow to assess the quality of the class diagrams produced by the GPT-4 model based on the requirements of natural language. As shown in Figure 1, the process starts with selecting a diverse set of NLRs as input scenarios. Benchmark class diagrams for each NLR are independently crafted by human experts to establish accurate reference points. The GPT-4 model is then tasked with generating the corresponding class diagrams based on the same NLRs. Subsequently, key components, such as classes, operations, and relationships, are methodically extracted from both the expert and GPT-4 diagrams. These components form the foundation for a detailed comparison, where matches and discrepancies are thoroughly evaluated using standardized criteria. Metrics, including precision, recall, and the F1 score, are calculated to measure the accuracy and completeness of the automated method. The results from this comparative process are then analyzed to derive insightful conclusions on GPT-4's effectiveness in class diagram creation and its implications for requirements engineering practice.

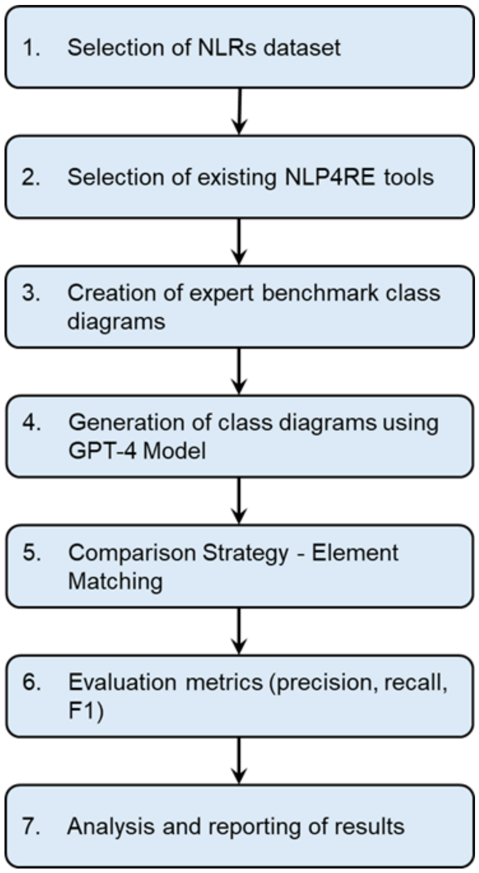
1. Selection of NLRs dataset
2. Selection of existing NLP4RE tools
3. Creation of expert benchmark class diagrams
4. Generation of class diagrams using GPT-4 Model
5. Comparison Strategy - Element Matching
6. Evaluation metrics (precision, recall, F1)
7. Analysis and reporting of results

Figure 1: Workflow of the Methodology

A. SELECTION OF NLRS DATASET

The data set used in this study includes five NLR specifications obtained from peer-reviewed publications and is widely recognized for its use in the evaluation of the NLP4RE tools. The NLR specifications range from 78 to 285 words in length, with an average sentence length of 12 words, and an average of 16 sentences per specification. The average lexical density is 43.23%. Lexical density is a measure that estimates the linguistic complexity of written or spoken communication by examining the ratio of content words to functional words. Thus, the lower lexical density implies that the specifications were relatively easy to understand. The NLR specifications are presented in the Appendix A and are briefly described as follows:

- *NLRs-1:* is from the library information systems domain and was presented and used in [23]. The specification has 219 words, 21 sentences, with an average length of 10 words per sentence and a lexical density of 46.58%.
- *NLRs-2:* gives a high-level description of the library system. The specification was used as an example in [24] and [25]. The specification has 78 words, 8 sentences, with an average length of 9 words per sentence and a lexical density of 41.03%.
- *NLRs-3:* focuses on an online shoe company that sells shoes to its customers through a website. The specification was originally published in [26]. The specification has 227 words, 15 sentences, with an average length of 15 words per sentence and a lexical density of 49.78%.
- *NLRs-4:* describes the interaction between a hypothetical salesperson and a sales order system. This specification was published and used in [27]. The specification has 169 words, 17 sentences, with an average length of 9 words per sentence and a lexical density of 46.15%.
- *NLRs-5:* is a hypothetical specification involving an in-flight missile control system. The specification was introduced and used in [28]. The specification has 285 words, 16 sentences, with an average length of 17 words per sentence and a lexical density of 32.63%.

B. SELECTION OF EXISTING NLP4RE TOOLS

Four existing NLP4RE tools were selected for this study. The tools were selected mainly because they have previously used one of the NLRs mentioned in Section III-A to generate a class diagram, and their solutions have been published in peer-reviewed conferences and journals. The four tools are briefly described below.

CM-Builder [23] is an NL-based tool that aims to support the requirements analysis process by identifying class, attributes and relationships of objects used to model a problem domain. CM-Builder takes software requirements text written in English as input and constructs an initial UML Class Model, either automatically or interactively with the analyst. The CM-Builder approach involves four stages: (1) Collect functional requirements or problem descriptions in natural language. (2) Use an NL processing system to syntactically and semantically analyze these requirements and create a world model as a knowledge base. (3) Extract object classes, attributes, and relationships based on predefined rules. (4) Generate an initial static structure model of the system and refine it using a graphical CASE tool.

CIRCE [28] is an environment to analyze and transform NLR specifications into different UML models, including a class diagram. It is based on the concept of successive transformations applied to requirements to obtain concrete, rendered views of models extracted from these requirements. The transformation process in CIRCE is divided into five steps. First, the natural language text of the requirements is parsed into a forest of parse trees. In the second step, these parse trees are encoded as tuples, providing extensional knowledge about the requirements. Next, an embedded expert system refines this knowledge by enriching the tuples, resulting in intentional knowledge about the software system's structure and behavior. In the fourth step, specific views of the requirements are modeled by extracting information from the shared tuple space, producing an abstract description. Finally, the abstract view is rendered into a concrete view. CIRCE has been implemented as a web-based system and uses a central server as a repository for requirements documents, allowing users to edit and request specific views through a standard web browser.

RACE (Requirements Analysis & Class Diagram Extraction) [24] is a method and tool designed to facilitate the requirements analysis process and class diagram extraction from textual requirements using natural language processing and domain ontology techniques. The RACE tool uses the OpenNLP[1] parser to obtain lexical and syntactic parses of textual requirements written in English. In addition, it uses WordNet to validate the semantic accuracy of the sentences generated during syntactic analysis. Concepts are identified according to the requirements document, and a domain ontology is used to enhance the performance of concept identification. The RACE tool identifies concepts based on analysis of nouns, noun phrases, and verbs. Heuristic rules are then used to extract class diagrams from the identified concepts. The RACE tool assists analysts by providing an efficient and fast way to produce class diagrams from their requirements. It supports good user interaction by offering a human-centered user interface that involves the user in the analysis process.

Shinde et al. System [25] uses natural language processing techniques to lexically analyze software requirement texts written in English and builds an integrated discourse model of the processed text, represented in a Semantic Network. This system automatically constructs UML diagrams (i.e., a class model representing the object classes mentioned in the text and their relationships) from the Semantic Network. It uses the WordNet dictionary to obtain the semantic relations between classes and attributes. In addition, the system includes an interactive user interface

[1]OpenNLP —http://opennlp.apache.org/

that allows the user to add, delete, and rename classes and relationships in the generated class diagram. Through this interface, the domain expert can help improve the results obtained by the system.

The four tools were developed before the launch of GPT-4 and utilized supervised learning techniques to generate class diagrams from the NLR specification. Supervised learning techniques require a comprehensive set of labeled data, where each input is paired with the correct output or label. These labeled data must accurately represent the problem domain to ensure that the model can effectively adapt to new unseen data. The quality of the labels is critical, as inaccuracies can lead to poor model performance. In addition, the data set should be evenly distributed across different classes in classification tasks to prevent biases and promote fair learning. Acquiring and correctly labeling these data is time-consuming and expensive but is essential for the effectiveness of supervised learning models.

C. CREATION OF EXPERT BENCHMARKS

In this study, a benchmark refers to a high-quality class diagram created by human experts, which acts as a standard or point of reference. The benchmarks provide a reliable foundation for evaluating the quality of diagrams generated by automated tools. Two requirements engineering experts participated in the creation of benchmark class diagrams for the five NLR specifications discussed in Section III-A. Expert 1 has more than 25 years of experience in the field of requirements engineering, teaching, and research, while Expert 2 has more than 15 years of experience in the same areas. The experts voluntarily participated in creating the benchmark class diagrams without any financial compensation, and each dedicated about 1.5 to 2 hours to the task, as detailed in Table 1. No strict time limit was imposed on experts; they were asked to complete the task at their normal professional pace to reflect realistic modeling conditions. This approach ensures that the benchmark diagrams reflect the best judgment of experts without restrictions that could compromise quality. The actual time spent by each expert provides transparency in terms of the effort required.

Employing human experts to create benchmark class diagrams is important for several reasons. The experts ensure that the diagrams are of the highest quality and accurately represent the required classes, relationships, and attributes. Their extensive knowledge in the field can establish a solid standard for evaluating the effectiveness of automated tools. Human experts are competent in dealing with the complexities and intricacies of real-world systems, enabling them to make well-informed decisions that automated tools may struggle with. Their skill in interpreting various requirements and constraints leads to solutions that are more adaptable and context-specific. Using manually created benchmarks can help identify the shortcomings of automated tools, guiding efforts to improve the performance of these tools. The two experts received the same dataset with the following instructions:

1) For each NLR specification, create a class diagram or identify relevant elements (e.g., classes, attributes, operations, and relationships) needed for constructing a class diagram.
   A class element can be textually represented as *Class [Attribute] (Operation)*, and a relationship can be represented as *class -verb phrase- class*.
2) Record the time taken to complete task 1.

The evaluation focused on the core elements of the UML class diagrams, in particular:

- *Classes* - the core entities that form the foundation of the system.
- *Operations* - specify the various functions or services that are made available by each individual class.
- *Relationships* - describe the various types of connection, including associations, aggregations, and generalizations that occur between classes.

According to the UML 2.0 specifications [1], these components together comprise the fundamental structural and dynamic elements of a system. Therefore, the functions to measure the accuracy and completeness of class diagrams are based on the identified classes, operations, and relationships. This study does not limit the types of UML classes considered, including abstract classes, interfaces, or stereotypes, which underscores the aim of evaluating general-purpose automated modeling tools suitable for diverse requirements. Future work will involve refining the scope to focus on specialized diagrams or domain-specific classes.

Table 1 shows the total number of elements (that is, classes, operations, and relationships) identified by experts according to the NLR specification. This task was achieved by counting individual classes, operations, and binary relationships in the benchmark diagrams produced by the experts. The experts identified a similar number of relationships, with minor differences. Operation counts differed more, and Expert 2 typically identified more numbers. Expert 1 took 96 minutes, and Expert 2 took 106 minutes to manually construct benchmark class diagrams for the five NLRs. The benchmarks are shown in the Appendix C for the readers to independently verify the number of elements counted.

D. GENERATION OF GPT-4 CLASS DIAGRAMS

ChatGPT utilizing the GPT-4 model was used to generate class diagrams from the NLR specifications. ChatGPT offers an interface that allows users to input their requirements and receive automatically generated class diagrams based on their specifications. In addition, it can provide further explanations and context to help users understand the rationale behind the generated diagrams. For each NLR specification, a class diagram was generated using the following prompt:

		NLRs-1	NLRs-2	NLRs-3	NLRs-4	NLRs-5	Total Time
Expert 1	Classes	10	10	6	6	9	91 mins
	Operations	12	10	16	17	12	
	Relationships	5	5	5	4	7	
	Total	**27**	**25**	**27**	**27**	**28**	
Expert 2	Classes	8	8	7	7	8	106 mins
	Operations	11	12	16	18	15	
	Relationships	5	6	6	6	7	
	Total	**24**	**26**	**29**	**31**	**30**	

Table 1: Number of elements identified by experts per NLRs.

Given the following natural language requirements, create a class diagram or identify the relevant elements (e.g., classes, attributes, operations, and relationships) needed to construct a class diagram.
A class element can be textually represented as `Class [Attribute] (Operation)`*, and a relationship can be represented as* `class -verb phrase- class`*.*
{NLR specification}

For the purpose of this study, only textual representations of the ChatGPT-generated class diagrams were obtained. However, it is worth mentioning that you can use UML diagramming tools to visually render these class diagrams by copying and pasting the textual representations. When GPT-4 generates textual representations of NLR class diagrams, the output includes several key elements that mirror the components of a traditional class diagram. The text output is formatted to include the following key elements.

Class Definitions: Each class is clearly identified by its name and is typically accompanied by a description that captures its purpose and role within the system.

Attributes (Properties): For each class, the model lists the attributes with their respective data types. The attributes are typically enumerated in a structured manner, identifying essential properties of the class.

Methods (Functions): the methods associated with each class are specified, and this includes their names, return types, and parameters.

Relationships: the relationships between classes, such as associations, inheritances, and dependencies, are included in the model. These relationships are crucial to understanding how different classes interact within the system.

A summary of the number of elements of the class diagram generated by the GPT-4 model for each NLR is shown in Table 2. The time taken by GPT-4 to generate a class diagram from an NLR specification was a matter of seconds, well within a minute.

	NLRs-1	NLRs-2	NLRs-3	NLRs-4	NLRs-5
Classes	7	11	7	6	9
Operations	13	17	13	28	22
Relations	5	6	6	10	8
Total	**25**	**34**	**26**	**44**	**39**

Table 2: Number of elements produced by GPT-4 model for each NLR.

E. COMPARISON STRATEGY

The accuracy and completeness of class diagrams generated by the GPT-4 model was evaluated by comparing them with the benchmark class diagrams created manually by experts. Each element in the class diagram produced by GPT-4 on the same NLR specification is matched to a corresponding element in the benchmark. For each match found, the counter representing the relevance of the GPT-4 class diagram is incremented by one; thus, an element is classified as *relevant*, if it correctly matches an element in the benchmark. Due to limited resources, the comparison was carried out by the author of the article and independently verified by two post-graduate students. The author asked post-graduate students and gave them instructions on how to follow the comparison strategy as presented in the Appendix B. The two post-graduate students who assisted with the independent verification of the comparison process participated on a voluntary basis and were not financially compensated for their contribution.

Following the approach used in [23], the matching is done in two stages. First, the names of the matching elements must be plausibly close (for example, *loan item* can match *item* but not *loan*). Second, the context in which the element has been used is taken into consideration to find out if that is what its name suggests. ***Classes*** are matched through approximate matching of names, while their context is determined by evaluating their attributes, operations, and relationships. Attributes are not counted; however, they are used to evaluate the context of a class element. In cases where two response classes could be matched to one key class or vice versa, the score will not be affected, regardless of the choice. ***Operations*** matching is only limited to name matching; thus, parameters, return types and data types were not taken into consideration. An operation with a name matching that of the benchmark should also appear in a relevant class. ***Relationships*** are matched through context matching, and only binary relationships were considered. Thus, a relationship between class A and B is classified as *correct*, if it exists between matching classes of the benchmark. In this comparison, *associations, aggregations, compositions* and *generalizations* are counted together as relationships.

F. EVALUATION METRICS

The quality of the generated class diagrams was assessed using precision, recall, and the F1 score, following standard practice in both requirements engineering and model transformation research [21], [22], [29]. These metrics have been widely used to quantify the correctness and completeness of software artifacts generated from natural language specifications.

To assess the correctness of class diagrams generated by the GPT-4 model compared to those created manually by experts, the **precision** metric is used. This metric evaluates the correctness and relevance of elements within a GPT-constructed class diagram, highlighting how closely the model aligns with the real world or domain. The precision is calculated using the following formula:

$$\text{Precision} = \frac{\text{Number of correctly identified elements}}{\text{Total number of elements retrieved}} \tag{1}$$

A high precision rate confirms that most of the components included in the generated class diagram from the NLR specification are relevant and accurate. This reduces the chance of adding unnecessary or incorrect elements, which can cause confusion or inefficiencies in the later development or maintenance stages. In addition, high precision increases the credibility of the documentation and promotes clearer communication between various stakeholders by ensuring that the information provided is relevant and concise. High precision in a generated class diagram signifies that the diagram is more targeted and relevant, while low precision suggests that many of the included components are extraneous or incorrect, resulting in a chaotic and potentially misleading diagram.

The completeness criterion evaluates GPT-4's ability to identify all model elements that align with those created by human experts. To assess the completeness of the class diagrams generated by the GPT-4 model, the **recall** metric is used. Recall measures the ratio of correctly identified relevant elements to the total number of relevant elements in the benchmark. Specifically, it focuses on how well the generated model captures all relevant information defined in the requirements. The recall measure is expressed as:

$$\text{Recall} = \frac{\text{Number of correctly identified elements}}{\text{Total number of elements in the Benchmark}} \tag{2}$$

The high recall rate guarantees that the majority of key components in the NLR specification are captured in the produced class diagram. This reduces the likelihood of overlooking vital elements, which could result in mistakes or misinterpretations during later development or maintenance phases. In addition, high recall contributes to the creation of more precise documentation and facilitates better communication between different stakeholders. A class diagram generated that demonstrates high recall suggests that it is more comprehensive. In contrast, low recall indicates that a significant number of essential elements are absent, leading to an incomplete diagram.

Precision and recall were initially created to assess information retrieval systems and have become widely used to evaluate software tools that generate software artifacts from NLR specifications [29], [30], [23]. In addition, the **F1 score**, which combines precision and recall in a single metric, is used for a more balanced evaluation. Provides a harmonic mean of precision and recall, offering a comprehensive measure of the accuracy of a model. The F1 score is particularly useful when assessments need to consider both the accuracy of retrieved elements and the completeness of the model generated from NLR specifications. This metric is widely adopted in the evaluation of software tools to ensure robust performance in the generation of software artifacts.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

IV. RESULTS

A. GPT-4 AGAINST EXPERT BENCHMARKS

To evaluate the consistency between two experts tasked with creating benchmark class diagrams, an inter-rater agreement analysis was performed using Cohen's kappa statistic. For each NLR, the presence or absence of every uniquely identified class element, as aggregated from the responses of both experts, was coded as 1 (present) or 0 (absent) for each expert. The observed agreement, calculated as the proportion of class elements for which both experts agreed on presence or absence, was 65.4%. The expected agreement by chance, based on the prevalence of identified classes per expert, was 63.2%. The resulting Cohen kappa coefficient was 0.06, typically indicating a slight agreement beyond chance [31].

The outcome highlights the inherent ambiguity and subjectivity in the analysis of requirements and the modeling of the UML class, where even experienced experts can interpret and abstract the same requirements of natural language differently. Despite experts receiving the same data set and instructions, there are variations in the number of identified elements. This variability demonstrates that there is no single benchmark class diagram for any given NLRs. Consequently, variations in class diagrams emerge, leading to an understanding that diagrams are not simply correct or incorrect, but judged as good or bad. Therefore, the use of multiple expert benchmarks is warranted to reflect the feasible solution space and provide a comprehensive evaluation framework for automated methods.

The performance of the GPT-4 model is evaluated against benchmark class diagrams from two experts, each providing a unique perspective on its effectiveness. Table 3 presents an assessment of the performance of GPT-4 using benchmarks created by Experts 1 and 2. This evaluation involves precision, recall, and F1 score metrics in five benchmark diagrams, labeled NLRs-1 to NLRs-5. Against

		NLRs-1	NLRs-2	NLRs-3	NLRs-4	NLRs-5
Expert 1	Precision	0.68	0.65	0.88	0.61	0.69
	Recall	0.63	0.88	0.85	1.00	0.96
	F1 score	0.65	0.75	0.87	0.76	0.81
Expert 2	Precision	0.76	0.71	0.77	0.64	0.77
	Recall	0.79	0.92	0.69	0.90	1.00
	F1 score	0.78	0.80	0.73	0.75	0.87

Table 3: Results of GPT-4's Performance Metrics on Expert Benchmark Diagrams

Expert 1 benchmarks, the GPT-4 model obtained precision rates ranging from 0.61 to 0.88, with optimal precision observed in NLRs-3, while recall scores range from 0.63 to a perfect 1.00, notably achieving full recall for NLRs-4. Against Expert 2 benchmarks, the GPT-4 model obtained precision rates between 0.64 and 0.77, and recall scores range from 0.69 to 1.00, achieving full recall in NLR-5. The F1 score, which synthesizes precision and recall, is between 0.65 and 0.87 for Expert 1 and between 0.73 and 0.87 for Expert 2. With both experts responsible for benchmark constructions, the variability in observed performance metrics underscores their differing perspectives or standards of analysis, indicating how the perceived consistency and reliability of GPT-4 may fluctuate with the complexity of benchmarks and variances in expert-designed criteria.

B. GPT-4 MODEL VS EXISTING NLP4RE TOOLS

The comparison strategy discussed in Section III-E was followed to objectively compare the class diagrams generated by the GPT-4 models to those generated by the existing NLP4RE tools. Thus, the published class diagrams produced by these tools were compared with the benchmark class diagrams created by experts to calculate their precision and recall rates. CM-Builder has a published class diagram for NLRs-1; when compared to Expert 1's benchmark for NLRs-1, 11 out of 27 elements were relevant, while compared to Expert 2's benchmark, 9 out of 24 elements were relevant. RACE and the system by Shinde et al. both have class diagrams published for NLRs-2. Compared to Expert 1's benchmark, RACE had 14 out of 25 relevant elements, while compared to Expert 2's benchmark, 12 out of 26 elements were relevant. The system of Shinde et al. had 17 out of 25 relevant elements when compared to Expert 1's benchmark, and 18 out of 26 elements were relevant when compared to Expert 2's benchmark. CIRCE has a class diagram published for NLRs-5 and when compared to Expert 1's benchmark, 22 out of 28 elements were relevant, while compared to Expert 2's benchmark, 21 out of 30 elements were relevant. Since we do not have access to the tools to generate class diagrams for the other NLR specifications, the comparison is limited to the published class diagrams. The evaluation focused on how well these tools aligned with class diagrams developed by experts on NLR-1, NLR-2, and NLR-5 benchmark diagrams.

Table 4 shows the precision, recall and F1 metrics for four NLP4RE tools: CM-Builder, RACE, Shinde et al., and CIRCE. CIRCE consistently demonstrated superior performance, achieving high precision and recall, leading to higher F1 scores. In contrast, CM-Builder and RACE demonstrated higher precision with lower recall, resulting in moderate F1 scores. Shinde et al. achieved a balance between precision and recall. These variations in precision, recall, and F1 scores underscore the various levels of performance of the tools when assessed against expert-generated diagrams.

The performance of GPT-4 was then compared with the NLP4RE tools using graphs. This sequence of graphs provides a comprehensive analysis of GPT-4's precision, recall, and F1 score compared to various NLP4RE tools, including CM-Builder, RACE, Shinde et al., and CIRCE, in the context of generating class diagrams from NLRs. As depicted in Figure 2, GPT-4 demonstrates higher precision compared to other tools on different benchmarks, especially excelling in NLRs-2 and NLRs-5. According to Figure 3, GPT-4 significantly outperforms other tools in terms of recall, especially excelling in NLRs-1, NLRs-2, and NLRs-5, where it achieves high or nearly perfect scores. This signifies the strong ability of GPT-4 to retrieve all relevant elements from expert-created diagrams. Figure 4, which harmonizes precision and recall, shows that GPT-4 consistently achieves high F1 scores. It surpasses other tools, especially in NLRs-2 and NLRs-5, demonstrating its effectiveness in handling NLRs. In general, GPT-4 shows high recall and precision on expert-created benchmarks, resulting in higher F1 scores compared to other tools. The evaluations underscore GPT-4's dependability and proficiency in producing class diagrams from NLR specifications.

Table 5 provides a comparison of the capabilities of the CM-Builder, CIRCE, Shinde et al., RACE, and GPT-4 tools to capture key elements of class diagrams. The elements assessed are Class, Operation, Attribute, Association, Aggregation, Composition, Generalization, Dependency, and Multiplicity. All tools evaluated can capture class, attribute and association elements, indicating a basic capabilities across the board. CM-Builder and RACE lack the ability to capture operations, whereas CIRCE, Shinde et al., and GPT-4 capture this element. Associations are well-supported across all tools. However, only GPT-4 captures dependencies, while all other tools fail to capture this element. This highlights the advanced capability of GPT-4 in identifying dependencies between classes, which is crucial to understanding interactions in complex systems. CM-Builder and GPT-4 capture multiplicity, but CIRCE, Shinde et al., and

		CM-Builder NLRs-1	RACE NLRs-2	Shinde et al NLRs-2	CIRCE NLRs-5
Expert 1	Precision	0.69	0.78	0.55	0.85
	Recall	0.41	0.56	0.68	0.79
	F1 score	0.51	0.65	0.61	0.81
Expert 2	Precision	0.56	0.67	0.58	0.81
	Recall	0.38	0.46	0.69	0.70
	F1 score	0.45	0.55	0.63	0.75

Table 4: Performance Evaluation of NLP4RE Tools on Benchmark Diagrams by Experts

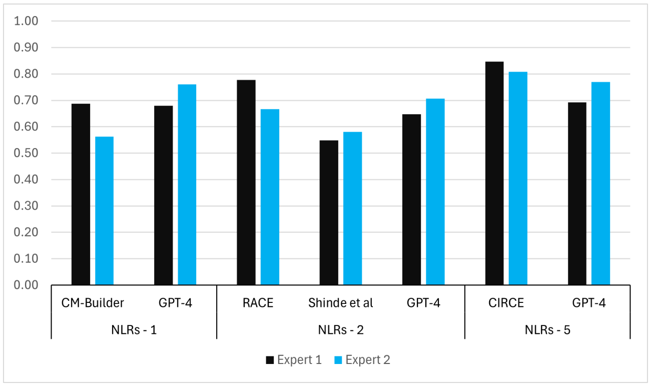

Figure 2: Precision Rate Comparison: GPT-4 and NLP4RE Tools with Expert Benchmarks

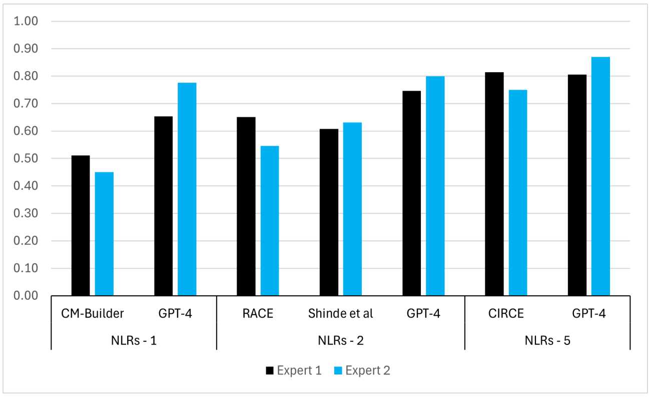

Figure 4: F1 Score Comparison: GPT-4 and NLP4RE Tools with Expert Benchmarks

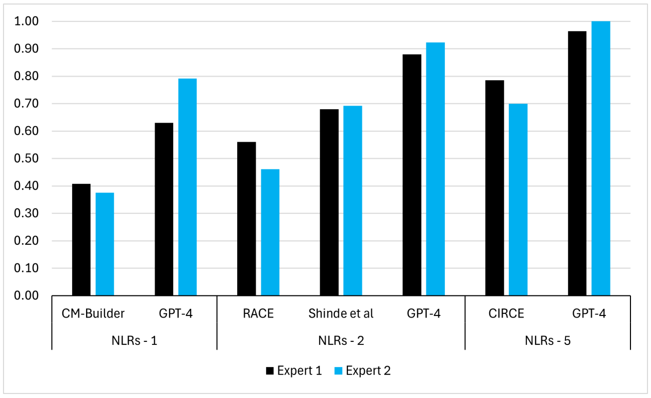

Figure 3: Recall Rate Comparison: GPT-4 and NLP4RE Tools with Expert Benchmarks

	CM-Builder	**CIRCE**	**Shinde et al.**	**RACE**	**GPT-4**
Class	✓	✓	✓	✓	✓
Operation	×	✓	✓	×	✓
Attribute	✓	✓	✓	✓	✓
Association	✓	✓	✓	✓	✓
Aggregation	✓	✓	×	✓	✓
Composition	✓	✓	×	✓	✓
Generalization	×	×	✓	✓	✓
Dependency	×	×	×	×	✓
Multiplicity	✓	×	×	×	✓

Table 5: Comparison of class diagram elements captured by the tools.

RACE do not. This could limit the detail and accuracy of the class diagrams produced. GPT-4 stands out as the tool with the most comprehensive set of capabilities, capturing all the class diagram elements listed. This shows that GPT-4 is robust and versatile in understanding and creating complex class diagrams.

V. DISCUSSION

This study evaluated the quality of the class diagrams generated by the GPT-4 model in two different settings: first, by comparing them to the benchmark diagrams created by experts manually, and second, by comparing them with the class diagrams produced by four existing NLP4RE tools. Manually constructed class diagrams contain implicitly added knowledge by the human modeler. Thus, they incorporate an element of human intelligence and are often aligned with the modeler's opinion of the system rather than with the actual requirements. The findings demonstrate that the GPT-4 model achieved higher recall rates compared to precision. This outcome implies that GPT-4 is capable of producing class diagrams with considerable comprehensiveness, indicating its proficiency in managing implicit knowledge. GPT-4 is capable of handling implicit knowledge through its extensive training in large and diverse data sets that include complex information [5]. It recognizes patterns within the data, allowing it to infer unstated details based on contextual similarities. By maintaining and utilizing context throughout text, GPT-4 can make associations between pieces of information and fill in the gaps. Additionally, it performs pragmatic inference by understanding meanings that go beyond literal word interpretations, leveraging its pre-trained exposure to common implications and unstated norms [5]. Consequently, GPT-4 is capable of identifying implicit classes, attributes, operations, and relationships similar to those identified by human experts.

The existing NLP4RE tools lack the capability to identify implicit knowledge of the given NLR specification, resulting in lower recall rates compared to the GPT-4 model. Furthermore, RACE and CM-Builder did not identify operations in their diagrams, resulting in lower recall rates. RACE and the tool by Shinde et al. are semi-automatic and include interactive user interfaces that allow users to add, remove, and rename classes and relationships in the class diagram. However, even with human intervention, the GPT-4 class diagrams are still better. This implies that human expertise is still needed in requirements modeling, even when aided by tools, as it ensures proper understanding and complete interpretation of complex requirements.

It should be noted that the diagrams generated by GPT-4 contained additional elements and more detailed operations, including parameters compared to the benchmarks created by experts. The inclusion of detailed operations and parameters indicates that the GPT-4 model not only identifies the core structural elements, but also attempts to specify how each class will function. Additional elements in a generated class diagram can have both advantages and disadvantages. Capturing additional elements ensures that all essential elements are considered. This is an advantage in the early stages of the requirement analysis as it ensures that all essential elements are captured. A disadvantage of this is that not all additional elements are relevant, which may introduce noise and distract from the core requirements, potentially leading to inaccurate class diagrams.

Generating class diagrams from NLR specification using GPT-4 is significantly faster compared to the time taken by human experts. Although GPT-4 can produce a class diagram in a matter of seconds, well under a minute, human experts often need more time, as shown in Table 1. For human experts, this process involves carefully reading and interpreting the NLR specification, conceptualizing the structure of the system, and manually creating the diagram, which can take several hours depending on the complexity of the NLR specification. The efficiency of GPT-4 not only speeds up the generation of class diagrams but also allows quick iterations and refinements [7], enabling a more agile and responsive development process. However, human expertise is crucial for validating the accuracy and completeness of the generated diagrams.

GPT-based models have made significant improvements in the area of natural language processing, but it is important to exercise caution about their use [6]. In the context of NLP4RE research, GPT-4 models may lack the domain knowledge necessary to understand domain-specific requirements, resulting in incomplete or even inaccurate class diagrams. Ensuring completeness in automatically generated class diagrams requires domain experts' input. Other issues that highlight the need for cautious and informed implementation include bias, information hallucination, lack of explainability, vulnerability to attacks, reasoning errors, overreliance, and ethical considerations [6], [10]. These issues call for an extensive evaluation of the class diagrams or any analysis models generated by GPT-based models before wider adoption of these models.

The results of this study are consistent with those of De Bari et al. [8], who noted that LLM can match or approximate human performance in class diagram modeling, although with scenario-dependent variation. This highlights the potential and difficulties of applying LLMs in model-based requirements engineering. In contrast to De Bari et al. [8], the use of dual expert benchmarks and quantitative information retrieval style scores enables a finer assessment and exposes expert variability, adding depth to the educational perspective.

The findings of this study could benefit software engineers and practitioners by demonstrating how GPT-4-generated class diagrams can ensure accuracy and consistency in requirements analysis and speed up the process. These insights can guide the development of more advanced NLP4RE tools, which could lead to more robust and user-friendly software tools in the field. Theoretically, the findings contribute to the existing body of knowledge by providing new insights into the capabilities of the GPT-4 model to automate RE tasks traditionally performed by human experts. Furthermore, the results can help formulate best practices for incorporating GPT-based solutions into the requirements engineering process, leading to more efficient and reliable software development methodologies.

VI. THREATS TO VALIDITY

Concerning **internal validity**, first, the method used to compare GPT-4 generated class diagrams with benchmark diagrams might introduce bias. This limitation was mitigated by clearly defining the comparison strategy in Section III-E. This strategy, previously used by other researchers in the field, was applied consistently during this study. The comparison was carried out by the author of the paper and independently verified by two postgraduate students. The postgraduate students were requested by the author and given instructions on how to follow the comparison strategy.

Second, human experts may introduce subjective bias when manually creating benchmark diagrams. Thus, they may include elements of human intelligence, often aligned with the expert's opinion of the system rather than with the actual requirements. To mitigate this limitation, the human experts who created the benchmark class diagrams used in this study have years of experience and expertise in the field of requirements engineering. The assumption is that experts with more experience can produce higher-quality benchmark diagrams, influencing the comparative results with GPT-4. Furthermore, the performance of the GPT-4 model was evaluated against two sets of benchmark class diagrams from both experts.

A third issue is the inconsistency in the results when GPT-4 generates class diagrams from the same NLRs, which can lead to potential variability in performance assessments. Although repeated iterations of diagram generation were not performed, an approach that might have yielded more constant results, the same set of instructions given to the experts was used as prompts for GPT-4 to ensure uniformity. The use of consistent instructions as prompts sought to minimize this variability. However, in the absence of result averaging or a controlled setting, this limitation significantly hinders the consistent evaluation of GPT-4's capabilities. Future research should adopt these measures to improve the reliability of the results.

In terms of **external validity**, the dataset used may not represent the full range of complexity and variability found in real-world software engineering projects, thus limiting the generalizability of the findings. To mitigate this limitation, the dataset used in this study included five NLR specifications from different problem domains, sourced from peer-reviewed publications and widely recognized for its use in evaluating NLP4RE tools. Another threat to external validity is related to the scope of the class diagram evaluation. There were no restrictions or distinctions imposed with respect to specialized UML class types such as abstract, interface, or stereotypes. Instead, the focus was on accurately identifying the elements of the core diagram. This approach aimed to ensure generalizability, but it might limit insight into tool performance in more specialized or domain-specific modeling tasks.

Construct validity issues can arise from the way accuracy and completeness are defined and interpreted when evaluating class diagrams. Narrow or vague definitions might not encompass all pertinent elements, potentially leading to incomplete evaluations of the model's quality. Furthermore, variations in interpretation can result in inconsistencies, compromising the validity of the findings. Definitions must also be in line with real-world requirements to ensure that the evaluation reflects practical usage. Consistency in applying these definitions throughout evaluations is essential to prevent misleading comparisons and to accurately reflect the tool's capabilities in the study. The definitions used in this research are derived from the literature and were initially formulated to evaluate information retrieval systems, now widely used to evaluate the performance of NLP4RE tools. The emphasis on accuracy and completeness should not imply the neglect of other critical elements, such as the usability and maintainability of the created class diagrams, which are equally significant and warrant further investigation.

VII. CONCLUSION

This study aimed to evaluate the quality of the class diagrams generated by the GPT-4 model. The GPT-4 generated class diagrams were compared with the benchmark diagrams created manually by two human experts. The performance of GPT-4 in generating class diagrams on Expert 1 benchmarks obtained a peak precision of 0.88 and Expert 2 a maximum of 0.77. Recall values are high, with perfect scores from the benchmarks of expert 1, NLRs-4, and expert 2 NLRs-5, which shows proficiency in element identification. F1 scores, which balance precision and recall, also support these claims, although they vary by expert benchmark. The conclusion is that GPT-4 is effective in class diagram generation, particularly in recall, but has room to improve precision for better overall performance. In addition, the performance results of GPT-4 were compared to those of four existing NLP4RE tools using the expert 1 and expert 2 benchmark diagrams. The findings demonstrated that GPT-4 generally outperforms these tools in precision, recall, and the F1 score, indicating a robust capability to generate accurate and comprehensive class diagrams from NLRs.

The contributions of this paper are threefold. First, it presents an evaluation of the quality of GPT-4 generated class diagrams by comparing them to benchmark diagrams manually created by human experts. Second, it provides an evaluation of how the GPT-4 generated diagrams compare to those produced by the four existing NLP4RE tools. Third, it offers insight to the requirements engineering community to understand how well GPT-4 can generate class diagrams from NLR specifications. The findings of this study suggest that GPT-based models have the potential to generate comprehensive class diagrams that can lead to more accurate and consistent analysis models during the requirements engineering phase. Despite its strengths, GPT-4's variability in outputs poses a threat to internal validity, as different executions can yield diverse diagrams. Although this study did not employ multiple iterations with average scores to mitigate this issue, the same instructions used by experts were consistently applied as prompts to control variability to some extent. This remains a limitation, emphasizing the need for further research using repeated trials and controlled conditions to achieve more stable assessments.

To further enhance the understanding of GPT-4's ability to generate class diagrams, future research could dive into specific task classes or focus on certain elements of UML, such as abstract classes, interfaces, or domain-specific stereotypes. Developing specialized evaluation functions for these focused tasks offers a promising pathway to obtain deeper insights and improve the quality of class diagram generation. Exploring these specific UML elements may lead to more accurate and domain-relevant class diagrams. This, in turn, could refine requirement engineering practices by providing tailored solutions to complex modeling tasks.

References

[1] B. Bruegge and A. H. Dutoit, Object-Oriented Software Engineering Using UML, Patterns and Java, 2nd ed. USA: Prentice Hall, 2004.

[2] C. Larman, Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development, 3rd ed. USA: Prentice Hall PTR, 2005.

[3] M. Fowler, UML Distilled: a brief guide to the standard object modeling language, 3rd ed. USA: Addison-Wesley Professional, 2004.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in North American Chapter of the Association for Computational Linguistics. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/N19-1423

[5] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI, San Francisco, CA, USA, Tech. Rep., 2018. [Online]. Available: https://www.mikecaptain.com/resources/pdf/GPT-1.pdf

[6] 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. [Online]. Available: https://doi.org/10.1109/MS.2023.3248401

[7] OpenAI, "Gpt-4 technical report," arXiv.org, San Francisco, CA, USA, Tech. Rep., 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08774

[8] D. De Bari, G. Garaccione, R. Coppola, M. Torchiano, and L. Ardito, "Evaluating large language models in exercises of uml class diagram modeling," ser. ESEM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 393–399. [Online]. Available: https://doi.org/10.1145/3674805.3690741

[9] J. Cámara, J. Troya, L. Burgueño, and A. Vallecillo, "On the assessment of generative ai in modeling tasks: an experience report with chatgpt and uml," Software and Systems Modeling, vol. 22, no. 3, pp. 781–793, 2023. [Online]. Available: https://doi.org/10.1007/s10270-023-01105-5

[10] N. Marques, R. R. Silva, and J. Bernardino, "Using chatgpt in software requirements engineering: A comprehensive review," Future Internet, vol. 16, no. 6, p. 180, 2024. [Online]. Available: https://doi.org/10.3390/fi16060180

[11] S. Speth, N. Meißner, and S. Becker, "Chatgpt's aptitude in utilizing uml diagrams for software engineering exercise generation," in 2024 36th International Conference on Software Engineering Education and Training (CSEE&T). Wurzburg, Germany: IEEE, 2024, pp. 1–5. [Online]. Available: https://doi.org/10.1109/CSEET62301.2024.10663027

[12] K. Ronanki, C. Berger, and J. Horkoff, "Investigating chatgpt's potential to assist in requirements elicitation processes," in 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Durres, Albania: IEEE, 2023, pp. 354–361. [Online]. Available: https://doi.org/10.1109/SEAA60479.2023.00061

[13] R. Khojah, M. Mohamad, P. Leitner, and F. G. de Oliveira Neto, "Beyond code generation: An observational study of chatgpt usage in software engineering practice," Proc. ACM Softw. Eng., vol. 1, no. FSE, jul 2024. [Online]. Available: https://doi.org/10.1145/3660788

[14] A. Mastropaolo, L. Pascarella, E. Guglielmi, M. Ciniselli, S. Scalabrino, R. Oliveto, and G. Bavota, "On the robustness of code generation techniques: An empirical study on github copilot," in 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). Melbourne, Australia: IEEE, 2023, pp. 2149–2160. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00181

[15] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," Proc. ACM Program. Lang., vol. 7, no. OOPSLA1, apr 2023. [Online]. Available: https://doi.org/10.1145/3586030

[16] N. Nguyen and S. Nadi, "An empirical evaluation of github copilot's code suggestions," in Proceedings of the 19th International Conference on Mining Software Repositories, ser. MSR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–5. [Online]. Available: https://doi.org/10.1145/3524842.3528470

[17] C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, "Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models," in 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). Melbourne, Australia: IEEE, 2023, pp. 919–931. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00085

[18] N. Nascimento, P. Alencar, and D. Cowan, "Artificial intelligence vs. software engineers: An empirical study on performance and efficiency using chatgpt," in Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering, ser. CASCON '23. USA: IBM Corp., 2023, p. 24–33. [Online]. Available: https://doi.org/10.5555/3615924.3615927

[19] M. A. Umar and K. Lano, "Advances in automated support for requirements engineering: a systematic literature review," Requirements Engineering, vol. 29, no. 2, pp. 177—207, 2024. [Online]. Available: https://doi.org/10.1007/s00766-023-00411-0

[20] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural language processing for requirements engineering: A systematic mapping study," ACM Computing Surveys (CSUR), vol. 54, no. 3, pp. 1–41, 2021. [Online]. Available: https://doi.org/10.1145/3444689

[21] T. Yue, L. Briand, and Y. Labiche, "A systematic review of transformation approaches between user requirements and analysis models," Requirements Engineering, vol. 16, no. 2, pp. 75–99, 2011. [Online]. Available: https://doi.org/10.1007/s00766-010-0111-y

[22] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo, "Empirical software engineering experts on the use of students and professionals in experiments," Empirical Software Engineering, vol. 23, pp. 452–489, 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9523-3

[23] H. Harmain and R. Gaizauskas, "Cm-builder: A natural language-based case tool for object-oriented analysis," Automated Software Engineering, vol. 10, no. 2, pp. 157–181, 2003. [Online]. Available: https://doi.org/10.1023/A:1022916028950

[24] M. Ibrahim and R. Ahmad, "Class diagram extraction from textual requirements using natural language processing (nlp) techniques," in Computer Research and Development, 2010 Second International Conference on. Kuala Lumpur, Malaysia: IEEE, 2010, pp. 200–204. [Online]. Available: https://doi.org/10.1109/ICCRD.2010.71

[25] S. K. Shinde, V. Bhojane, and P. Mahajan, "Nlp based object oriented analysis and design from requirement specification," International Journal of Computer Applications, vol. 47, no. 21, pp. 30–34, June 2012. [Online]. Available: https://doi.org/10.5120/7475-0574

[26] K. Lunn, Software Development with UML, 1st ed. London, United Kingdom: Palgrave Macmillan, 2002.

[27] P. Harmon and M. Watson, Understanding UML: The Developer's Guide: with a Web-based Application in Java. Massachusetts, United States: Morgan Kaufmann Publishers Inc., 1997.

[28] V. Ambriola and V. Gervasi, "On the systematic analysis of natural language requirements with circe," Automated Software Engineering, vol. 13, no. 1, pp. 107–167, 2006. [Online]. Available: https://doi.org/10.1007/s10515-006-5468-2

[29] V. B. R. Vidya Sagar and S. Abirami, "Conceptual modeling of natural language functional requirements," Journal of Systems and Software, vol. 88, pp. 25–41, 2014. [Online]. Available: https://doi.org/10.1016/j.jss.2013.08.036

[30] M. Elbendak, P. Vickers, and N. Rossiter, "Parsed use case descriptions as a basis for object-oriented class model generation," Journal of Systems and Software, vol. 84, no. 7, pp. 1209–1223, 2011. [Online]. Available: https://doi.org/10.1016/j.jss.2011.02.025

[31] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," Biometrics, vol. 33, no. 1, pp. 159–174, 1977. [Online]. Available: https://doi.org/10.2307/2529310

KELETSO J. LETSHOLO holds a PhD in Computer Science from the University of Manchester, United Kingdom, specializing in Software Engineering. In addition, he holds an MSc in Computer Science from the University of Wollongong, Australia. He is currently serving as an Assistant Professor at the Higher Colleges of Technology (HCT) in the United Arab Emirates. His research focuses on the convergence of Requirements Engineering and Natural Language Processing, with a significant emphasis on Machine Learning. He seeks to refine NLP algorithms to more accurately handle complex requirements and exploit deep learning to enhance automation tools.

.

APPENDIX A NLRS DATASET

A. NLRS-1: LIBRARY INFORMATION SYSTEMS I

"A library issues loan items to customers. Each customer is known as a member and is issued a membership card that shows a unique member number. Along with the membership number, other details on a customer must be kept such as a name, address, and date of birth. The library is made up of a number of subject sections. Each section is denoted by a classification mark. A loan item is uniquely identified by a bar code. There are two types of loan items, language tapes, and books. A language tape has a title language (e.g. French), and level (e.g. beginner). A book has a title, and author(s). A customer may borrow up to a maximum of 8 items. An item can be borrowed, reserved or renewed to extend a current loan. When an item is issued the customer's membership number is scanned via a bar code reader or entered manually. If the membership is still valid and the number of items on loan less than 8, the book bar code is read, either via the bar code reader or entered manually. If the item can be issued (e.g. not reserved) the item is stamped and then issued. The library must support the facility for an item to be searched and for a daily update of records" [23].

B. NLRS-2: LIBRARY INFORMATION SYSTEMS II

"The library System is used by the Informatics students and Faculty. The Library contains Books and Journals. Books can be issued to both the Students and Faculty. Journals can only be issued to the Faculty. Books and Journals can only be issued by the Librarian. The deputy-Librarian is in-charge of receiving the returned Books and Journals. The Accountant is responsible for receiving the fine for over due books. Fine is charged only to students and not to the Faculty" [24], [25].

C. NLRS-3: ONLINE SHOE COMPANY

"Customers will need to register with the Odd Shoe Company to make orders. On registration, they need to provide name and address, payment details (credit card, etc), shoe sizes, gender, and any special details. To order, customers will select from the shoe range. The system will tell them if the shoes are in stock, or need to be ordered from a supplier. If the shoes are in stock, then the system will tell them how quickly the shoes can be delivered. Customers will want to track their orders online. An order can either be waiting for delivery to the Odd Shoe Company, waiting for dispatch, waiting for credit clearance, or dispatched. Before dispatch, the customer should have an option for cancelling the order. John would like a weekly report, detailing numbers of customers, statistics on shoe sizes by left and right foot, orders, stocks, and cancelled orders. Every month customers will be sent a statement by email, together with a list of special offers. Offers will only be for shoe pairs that are available in suitable sizes for that customer. John wants to keep pictures, prices, stock levels, and sizes of all his shoes in a database. When supplies arrive the database will need updating. When goods are ordered, the stock levels should be deducted. If an order is cancelled, the stock levels should be updated accordingly" [26, p. 377].

D. NLRS-4: SALES REPORT APPLICATION

"Salesperson turns on laptop, brings up the SaleWeb program, and chooses Report Sales Order from menu. Salesperson enters name, employee number, and ID. Sales Order checks to see if name, number and ID are valid. Salesperson enters customer name and address on sales order form. Salesperson checks customer information to find customer status. CustInfo checks Accounting to determine customer status. Accounting approves customer information and supplies customer credit limit. CustInfo accepts customer entry on Sales Order. Salesperson enters first item being ordered on sales order form. Salesperson enters second item being ordered, etc. When all items have been entered Items ordered are checked to determine availability and to check pricing. Items ordered checks with Inventory to determine availability and to check pricing. Inventory supplies all availability dates (when it can ship), approves prices, adds shipping and taxes, and totals order. Complete order appears on salesperson?s screen. Salesperson can print order, check with customer, etc. Salesperson submits the approved Sales Order. Sales Order is submitted to Accounting and Inventory" [27, p. 121].

E. NLRS-5: MISSILE CONTROL SYSTEM

"A launch control center manages a number of missiles. Each missile is controlled by an on-board FMCS unit. The FMCS is initially idle. When the FMCS receives a launch command from its base, if it is idle, the FMCS enters confirmation waiting mode. If the FMCS is in confirmation waiting mode, it receives a launch confirmation command from its base, and the token of the launch confirmation command is equal to the token of the launch command, then the FMCS enters operation mode. When the FMCS receives a launch cancellation command from its base, if it is in confirmation waiting mode, it becomes idle. When the FMCS enters operation mode, it turns on the main engine. A main engine is part of a missile. While the FMCS is in operation mode: every 100 milliseconds, it reads the height from the altimeter, and if the height is greater than 30 meters, it enters cruise mode. If the FMCS is still in operation mode 5 seconds after it entered that mode the FMCS turns off the main engine, and it enters idle mode. During flight, every 20 milliseconds: the FMCS reads the height from the altimeter, it reads the heading from the gyroscope, and it computes appropriate navigation commands, based on the target position. During flight, if the navigation commands are significant, the FMCS sends these commands to the navigation engines. During flight, the FMCS logs position, height, and heading every 2 seconds. The Launch Control Center sets the target position for the FMCS of its missiles. During flight, the FMCS reads the current position from its

GPS device every 5 seconds. A missile has as components: a main engine, a set of navigation engines, a gyroscope, an altimeter, and a GPS device" [28].

APPENDIX B COMPARISON TASK INSTRUCTIONS FOR STUDENT VERIFIERS

The postgraduate students who assisted in this study were provided with the following instructions to ensure consistency and reliability in the comparison process between the GPT-4-generated and expert benchmark class diagrams.

1) For each Natural Language Requirement (NLR) and its corresponding class diagrams (GPT-4 and expert benchmark), systematically compare all classes, operations, and relationships.
2) For each unique element appearing in the GPT-4 or expert benchmark diagrams, indicate the presence (1) or absence (0) in each diagram.
3) Count an element as a correct match if its name closely matches in both diagrams (e.g., "LoanItem" and "Item" may count as a match if justified by a similar context).
    - Consider the context, such as associated attributes, operations, or relationships, to support your judgment of element matches, particularly in ambiguous cases.
    - *Operations* - Only the operation's name and its presence within the correct class are considered; ignore parameters, data types, or return types.
4) *Relationships* - Limit the matching to binary relationships (between two classes). Relationships are considered matched if both endpoint classes correspond, regardless of relationship type (association, generalization, etc.).
5) Attributes should not be scored individually, but may be considered to clarify the context of class elements.
6) Use the provided spreadsheet/template to record the results for each element.

APPENDIX C BENCHMARK CLASS DIAGRAMS CREATED BY EXPERTS

NLRs 1: Library Information Systems I

Class [Attribute](Operation):

Library [] (issuesItem(), issuesCard())
Item [barCode, name] (reserveItem(), renewItem(), returnItem())
Customer [name, address, dateOfBirth] (login(), borrowItem(),returnItem())
MembershipCard [memberNo] ()
Subject_Section [classificationMark] ()
Book [title, author]()
Language_Tape [title, level] ()
BarCode_Reader [title, author] (scanBarCode(), enterBarCode())
Current_Loan [] (extendLoan())
Record [] (updateRecords())

Relationships:
Library -issues- Loan Items
Customer -issued- Membership Card
Library -made of- Subject Sections
Customer -borrow- Items
Membership card -scanned by- BarCode_Reader

Time spent: 16 mins

NLRs 2: Library Information Systems II

Class[Attribute](Operation):

Library_System [] (searchItem())
Student [] (borrowItem(), returnItem())
Faculty [] (borrowItem(), returnItem())
Item [status] (getStatus())
Book [isbn, title, author, year, publisher] ()
Journal [title, author] ()
Librarian [] (issueItem())
Deputy_Librarian [] (receiveItem())
Accountant [] (calculateFine())
Fine [] (checkoverdueItems())

Relationships:
Students -can borrow- Books
Faculty -can borrow- Books & Journals
Librarian -issues- Items
Deputy Librarian ---receives- Items
Accountant -charges- Fines

Time spent: 10 mins

NLRs 3: Online Shoe Company

Class [Attribute] (Operation):

Customer [name, address, payment_details, shoe_size, gender] (register(), login(), makeOrder(), trackOrder(), cancelOrder()),
OddShoeCompany [] (checkStock(), deliverOrder(), dispatchOrder(), sendStatement()),
Order [customer, status, shoe, date] (getOrderStatus())
Shoe [picture, price, stock_level] (findShoe(), getShoeList(), updateStockLevel())
Weekly_Report [Customers, ShoeSizes, Orders, StockLevels, cancelledOrders] (generateWeeklyReport())
Statement [] (getOffers(), sendEmail())

Relationships:
Customer -makes- Order
Customer -registers with- OddShoeCompany
OddShoeCompany -creates- Weekly_Report
OddShoeCompany -sends- Statement
Order -contains- Shoes

Time spent: 17 mins

NLRs 4: Sales Report Application

Class [Attribute] (Operation):

Salesperson [name,employee_number,ID] (login(), submitCustInfo(), checkCustInfo(), printOrder(), checkOrder(), submitOrder())
Order [availability, price, shipping, tax] (checkSalesPerson(), checkAvailability(), checkPricing())
Customer [name, address, status] (checkCreditLimit())
Item [name, price] (addItem())
Inventory [date, price, shipping, tax](approvePrice(), addShipping(), addTaxes(), totalOrder())
Accounting [] (approveCustInfo(), supplyCustomerCreditLimit()),

Relationships:
Salesperson ---submits--- Order
Order ---includes--- Items
Inventory ---checks--- Order
Accounting ---approves--- Order

Time spent: 23 mins

NLRs 5: Missile Control System

Class [Attribute] (Operation):

Missile [mode] ()
FMCS [command, height, position] (enterMode(), setMainEngineMode(), readHeight(), readHeading(), readPosition(), sendNavigationCommand(), computeCommand())
Main_Engine [mode: on, off] ()
Navigation_Engine [command] ()
Gyroscope [] (getHeading())
Altimeter [] (getHeight())
GPS [](getPosition())
Command [token] (getCommand())
Launch_Control_Center [] (setTarget())

Relationships:
Launch_Control_Center -manages- Missiles
FMCS_Unit -controls- Missile
Missile -has- Main_Engine
Missile -has- Navigation_Engine
Missile -has- Gyroscope
Missile -has- GPS
Missile -has- Altimeter

Time spent: 25 mins

Figure 5: Expert 1 benchmark class diagram elements for the five NLRs.

NLRs 1: Library Information Systems I

Class [Attribute](Operation):
Customer [name, address, dateOfBirth] (login(), borrowItem(),returnItem())
Library [] (searchItem(), updateRecords())
Section [classificationMark]()
LoanItem [barCode, name] (issueItem(), reserveItem(), renewItem(), extendLoan())
types of LoanItem: Book [title, author]();
LanguageTape[title, level]()
MembershipCard [memberNo] ()
BarCodeReader [title, author] (scanBarCode(); enterBarCode())

Relationships:
Customer -issued- MembershipCard
Library -made up- Section
Customer -borrows- LoanItem
BarCodeReader -scans- MembershipCard
Section -contain- LoanItem

Time spent: 18 mins

NLRs 2: Library Information Systems II

Class [Attribute](Operation):
Book [isbn, title, author, year] (updateStatus())
Journal [title, author, year](updateStatus())
Student [id, name, major] (borrowBook(), returnBook())
Faculty [id, name, position] (borrowBook(), borrowJournal())
Librarian [id, name] (issueBook(), issueJournal())
DeputyLibrarian[id, name](receiveReturns())
Accountant [id, name] (receivePayment())
Fine [amount, date] (payAmount(), calculateAmount())

Relationships:
Books -issued to- Students.
Books -issued to- Faculty.
Journals -issued to- Faculty.
Librarian -issues- Books.
Librarian -issues- Journals.
Accountant -receives- Fines

Time spent: 14 mins

NLRs 3: Online Shoe Company

Class [Attribute] (Operation):
Customer [cust_number, name, address, payment_details, shoe_sizes, gender, age] (register(), placeOrder(), trackOrder(), cancelOrder())
Order [order_id, status] (getStatus(), updateStatus())
Stock/Shoe [shoe_id, size, price, quantity, picture, status] (checkStatus(), updateQuantity())
Report [cust_number, order_id, shoe-id] (generateReport(), sendReport())
Supplier [name, address] (fulfilOrder())
ShoeCompanySys [] (createPurchaseOrder(), receiveGoods(), cancelPurchaseOrder())
SpecialOffer [shoe_id, price, shoe_size] (getOffers(), sendOffers())

Relationships:
Customer -makes- Order
Order -contains- Stock/Shoes
Supplier -supplies- Stock/Shoes
ShoeCompanySys -order from- Supplier
ShoeCompanySys -generates- Report
Stock/Shoe -have- SpecialOffer

Time spent: 25 mins

NLRs 4: Sales Report Application

Class [Attribute] (Operation):
Salesperson [emp_number, name, ID] (login(), createOrder(), printOrder(), submitOrder())
SaleWebProgram [] (checkLoginDetails(), checkCustInfo())
Customer [name, address, status](checkCustStatus ())
Accounting [] (approveCustInfo(), approveOrder())
SalesOrder [id, date, totalPrice] (addItem(), checkItemAvailabity (), calculateTotalPrice())
Inventory [] (checkItemAvailabity(), checkItemPricing(), checkDeliveryDate(), calculateShipping(), updateInventory())
Item [item_id, name, price] (searchItem())

Relationships:
Saleperson -uses- SaleWebProgram
SaleWebProgram -creates- SalesOrder
SalesOrder -includes- Items
Inventory -contains- Items
SalesWebProgram -checks- Customer
Accounting -approves- SalesOrder

Time spent: 28 mins

NLRs 5: Missile Control System

Class [Attribute] (Operation):
LaunchControlCenter [] (setTarget())
Missile [status] ()
FMCS_Unit [height, heading, tartgetPosition] (receiveCommand(), updateStatus(), turnOnEngine(), readAltimeter(), readGyroscope(), turnOffEngine(), readGPS()), computeNavCommand())
MainEngine [mode] (receiveCommand(), updateMode())
NavigationEngine [currentPosition, targetPosition] (receiveNavCommand())
Gyroscope [] (getHeading())
GPS_Device [] (getCurrentPosition())
Altimeter [] (getHeight())

Relationships:
LaunchControlCenter -manages- Missiles
FMCS_Unit -controls- Missile
Missile -has- MainEngine
Missile -has- NavigationEngine
Missile -has- Gyroscope
Missile -has- GPS_Device
Missile -has- Altimeter

Time spent: 21 mins

Figure 6: Expert 2 benchmark class diagram elements for the five NLRs.