

AUTO-GENERATION OF UML SEQUENCE DIAGRAMS FROM NATURAL LANGUAGE REQUIREMENTS USING GPT-4 MODEL

Ji Yong Loo^{ID}¹ and Moon Ting Su^{ID}^{2*}

^{1,2} Department of Software Engineering, Faculty of Computer Science and Information Technology,
Universiti Malaya, 50603 Kuala Lumpur, Malaysia.

Emails: loojiyong@gmail.com¹, smting@um.edu.my^{2*} (Corresponding author)

Received: 22nd December 2025

Revised: 24th June 2026

Accepted: 14th August 2026

ABSTRACT

Automating the generation of Unified Modelling Language (UML) sequence diagrams from software functional requirements (FRs) can improve productivity. However, existing methods often require user intervention and therefore are not fully automated. To address this gap, this research proposed a novel LLM-based method for automating the generation of UML sequence diagrams from FRs written in English for the web application domain, without requiring human intervention. The proposed method leverages OpenAI's Generative Pre-Trained Transformer-4 (GPT-4), extracts key information from FRs and constructs corresponding PlantUML statements, which are rendered into UML sequence diagrams. A proof-of-concept prototype implementing the proposed method was developed. The correctness of the artefacts generated by the prototype was evaluated by calculating cosine similarity between them and ground truth ("oracle") created by third-party experts. An average similarity score of 86.14% across 10 FR sets was obtained, demonstrating a high level of correctness of the generated artefacts. This result shows the potential of LLM, particularly GPT-4 in automatically generating UML sequence diagrams from English FRs.

Keywords: *UML Sequence Diagram Generation; Software Requirement Engineering; Software Design; Large Language Model; GPT; Natural Language Processing (NLP).*

1.0 INTRODUCTION

The software development life cycle (SDLC) refers to the life cycle that covers all stages of software development from its inception with requirement definition through to deployment and maintenance [1]. Within the SDLC, requirement analysis, software architecture and software design are distinct yet interrelated phases [2]. According to SWEBOK 4.0 [2], requirement analysis focuses on defining and analysing stakeholder needs, while software architecture deals with the macro-level structure and key constraints, software design is concerned with the micro-level realization of that architecture, detailing exactly how components function and communicate internally. To transition smoothly between these phases, Sommerville [3] notes that "system models are used during the requirements engineering process to help derive the detailed requirements for a system, during the design process to describe the system to engineers implementing the system."

Among the various types of system models, behavioural models are required to capture the dynamic behaviour of a system as it responds to user inputs and external events. It is within this context that the Unified Modelling Language (UML) sequence diagram emerges as a critical tool. Serving as the primary notation for interaction modelling, the sequence diagram maps out the explicit temporal ordering and sequential flow of messages between objects. Consequently, it acts as the vital analytical bridge between initial stakeholder needs and formal software design.

It is critical to perform thorough requirement analysis and software design as these can help in identifying potential gaps and minimising errors, thereby saving time, money, and manpower [4]. However, finding adequate time to (re-) do them is challenging in today's fast-paced software development projects that demand shorter time-to-market and continuous improvement of the software to satisfy stakeholders' changing and evolving requirements.

Natural language (NL) has served as the prevalent notation for documenting and specifying system requirements [5]. Researchers in requirement engineering have explored the use of various natural language processing (NLP) techniques since the 1990s. The review studies by Abdelnabi et al. [6] and Ahmed et al. [7] found that most of the existing studies primarily focused on the generation of static models such as UML class diagrams and package diagrams instead of behavioural models like UML sequence diagrams or activity diagrams. Static models represent the static aspect of the system, focusing on the elements present in the system being modelled, whereas, behavioural models represent the dynamic aspect, focusing on the interaction among objects and the changes of an object in the domain over time [8].

Existing solutions for generating software models from NL requirements can be categorised into conventional rule-based methods and machine learning approaches, and they both face notable limitations. Abdelnabi et al. [6] and Ahmed et al. [7] highlighted the limitations of the conventional rule-based methods, which are, high complexity, constraints on NL inputs, and the need for constant user intervention. On the other hand, existing solutions that leverage machine learning models, particularly large language models (LLMs), face challenges in ensuring consistent and accurate outputs [9]. Additionally, most of the current LLM-based methods [9, 10, 11, 12] are in fact using ChatGPT directly, which may require users to refine the results repetitively to get the desired result. While there are some studies [6, 7, 13, 14] on automating the transformation of NL requirements into UML models, currently, there is no method or tool that can fully automate the transformation process without requiring constant user intervention.

The previous two paragraphs show that existing solutions have not achieved a fully automated process for the transformation of NL requirements into UML models. This gap not only hinders the automation of model generation but also perpetuates the manual effort required during the requirements analysis phase of the SDLC. Addressing this gap has the potential to enhance productivity in software development processes.

This research aimed to address the gap above by proposing a novel LLM-based method for automating the generation of UML sequence diagrams from software functional requirements (FRs) written in English for the web application domain, without requiring human intervention. The proposed method leverages OpenAI's Generative Pre-Trained Transformer-4 (GPT-4) via Application Programming Interface (API). The proposed method extracts key information from FRs and constructs PlantUML statements which are rendered into UML sequence diagrams. To demonstrate the technical feasibility of the proposed method, a proof-of-concept prototype that incorporates the proposed method was developed using the Python programming language. The correctness of the PlantUML statements generated by the prototype was evaluated by calculating cosine similarity between them and ground truth ("oracle") created by third-party experts.

As mentioned earlier in this section, there is a scarcity of studies on generating behavioural models from NL requirements. Therefore, this research focused on generating behavioural models, in particular UML sequence diagrams. According to the UML specification [8], sequence diagram is the most widely used interaction diagram among all types of UML behavioural models. As a one-size-fits-all solution may not be appropriate in all cases, this research focused on the domain of web application. To the best of the researcher's knowledge, no previous related study has specifically targeted this domain.

The main contribution of this paper is a fully automated method that generates UML sequence diagrams directly from FRs written in NL. The proposed method integrates OpenAI's GPT-4 model via API, using fine-tuned prompts and deterministic parameters to ensure consistent output, followed by validation and rendering of correct PlantUML statements into UML sequence diagrams. A proof-of-concept prototype was developed in Python to demonstrate the method's feasibility.

The remainder of this paper is organized as follows. Section 2.0 reviews the background and related work, highlighting prior studies in software model generation and identifying existing research gaps. Section 3.0 introduces the proposed method for automating the generation of UML sequence diagrams from functional requirements, detailing the integration with the GPT-4 model, prompt construction, deterministic parameter tuning, validation, and rendering processes. Section 4.0 describes the proof-of-concept prototype developed in Python, highlighting its architecture, key components, and interaction with the GPT-4 model. Section 5.0 presents the evaluation study, including the construction of oracle models, participant recruitment, similarity-based correctness assessment. Section 6.0 discusses the results and key findings. Finally, Section 7.0 concludes the paper by summarizing the main contributions, limitations, and potential directions for future research.

2.0 BACKGROUND

2.1. NLP4RE

Zhao et al. [15] conducted a systematic mapping study on natural language processing for requirements engineering (NLP4RE), analysing 404 studies published over 36 years. Their findings revealed that most research efforts concentrate on the elicitation and analysis phases, particularly on detecting quality defects, while few tools

have been validated for industrial use. Similarly, Alzayed and Al-Hunaiyyan [16] reviewed NLP applications across requirements elicitation, analysis, and validation, highlighting its benefits for information extraction and consistency checking but noting limitations in accuracy and completeness. Both studies emphasized the need for advanced NLP tools capable of supporting multiple phases and bridging NL requirements with structured models.

2.2. Automated Software Models Design

Research on automated software models design can broadly be divided into two categories:

- Structural models: which represent the static aspects of a system such as components, hierarchies, and relationships (e.g., UML class and component diagrams).
- Behavioural models: which describe the dynamic interactions and message flows between system entities (e.g., UML sequence and use case diagrams) [8].

Abdelnabi et al. [6] and Ahmed et al. [7] provided comprehensive reviews of techniques for transforming NL requirements into UML models. They classified existing approaches as rule-based, machine learning-based (ML-based), and hybrid. Rule-based approaches are simple but limited in accuracy, ML-based approaches achieve higher accuracy but are complex to implement, and hybrid approaches offer the best results at the cost of high complexity. Both reviews highlighted that most research focuses on generating UML class diagrams rather than other UML models, and that existing tools often produce incomplete diagrams requiring human intervention. No fully functional end-to-end tool currently exists for converting NL requirements into UML diagrams.

Several studies have proposed methods to automate the generation of UML class diagrams from NL requirements. Abdelnabi et al. [17] used NLP and heuristic rules to extract classes, attributes, and relationships, while Bashir et al. [18] combined NLP with ontology refinement in their READ system, achieving improved precision and recall. Although these approaches show promise, they remain constrained by specific input formats, limited datasets, and domain-specific ontologies that are laborious to build.

Research on behavioural model generation remains limited. Gulia and Choudhury [13] applied NLP to derive UML sequence and activity diagrams from textual specifications, though their evaluation was limited. Maatuk and Abdelnabi [19] extended this to generate UML class, use case, and activity diagrams, but focused more on capability demonstration than correctness assessment. Jahan et al. [14] automated UML sequence diagram creation from use case scenarios using NLP and rule-based mapping, achieving good completeness and correctness but lacking independent assessment.

Collectively, these studies reveal that most automated approaches concentrate on generating static UML models, while behavioural models, particularly UML sequence diagrams remain underexplored and less rigorously validated. This gap motivates this research.

2.2.1. Software Modelling Tools

Software modelling tools are essential in the software engineering domain for designing and visualizing system architectures, including UML sequence diagrams. These tools generally fall into two broad paradigms: traditional visual-based modelling tools and text-based modelling tools.

Traditional visual-based tools like Visual Paradigm [20] and Lucidchart [21] primarily rely on a Graphical User Interface (GUI) where users manually drag and drop components, draw interaction lines, and adjust the spatial layout of the models. While they provide highly customizable 'What You See Is What You Get' (WYSIWYG) environments, their reliance on manual visual manipulation makes them difficult to integrate into fully automated, machine-driven generation pipelines. Additionally, the underlying project files generated by these GUI tools are challenging to version-control or merge effectively in collaborative development environments.

To address these limitations, text-based modelling tools have gained increasing traction in recent years. Tools such as PlantUML [22], Mermaid [23], and UMLet [24] offer an effective way of creating software models using specialized code-like languages, which are then rendered into visual diagrams automatically. Unlike traditional visual tools, text-based alternatives abstract away the manual layout process; the rendering engine automatically computes the optimal spatial arrangement of elements based on standard UML conventions.

One key advantage is that such textual formats can be easily version-controlled, reviewed, and ensure effective team collaboration [25]. More critically for the context of this research, textual modelling syntaxes are fundamentally compatible with Large Language Models (LLMs). As noted by Cámara et al. [9], LLMs are capable of producing models in different intermediate modelling syntaxes, making text-based tools like PlantUML the ideal bridge. They enable the automated rendering of LLM-generated logic into standard UML sequence diagrams without the need for manual GUI adjustments or human intervention.

2.3. Pre-trained LLMs

There are several notable LLMs from key players in the artificial intelligence (AI) industry such as OpenAI's GPT models [26], Meta's LLaMA models [27], and Google's Gemini models [28]. With the increasing prominence of LLMs, numerous studies have demonstrated that GPT models achieve remarkable performance on various NLP tasks as evidenced by automatic evaluation metrics. For instance, in the domain of natural language understanding (NLU), Wei et al. [29] have shown that ChatGPT can attain robust performance, even surpassing some full-shot models. Similarly, in the domain of natural language generation (NLG), Wang et al. [30] concluded that ChatGPT sets a new state-of-the-art, maintaining parity with established datasets and outperforming known competitors.

2.3.1. Utilization of LLMs In Software Design

Recent advances in LLMs have spurred growing interest in their application across software engineering disciplines. Early findings indicate that LLMs perform well in code generation tasks, generally producing syntactically correct outputs [31]. However, research on their use in software design remains limited [32].

Ahmad et al. [10] explored the collaboration between architects and ChatGPT, where ChatGPT refined Architecturally Significant Requirements and generated UML class and component diagrams through iterative human-AI interactions. While promising, the approach was inefficient due to extensive back-and-forth prompting. Similarly, Cámara et al. [9] examined ChatGPT's ability to generate UML class diagrams across multiple domains, reporting generally accurate results but highlighting inconsistencies, syntactic and semantic errors, and domain dependency.

Kim [12] evaluated ChatGPT's performance across the software development life cycle using a tour reservation system case study. The model successfully derived FRs and non-functional requirements (NFRs) and produced textual UML class and sequence descriptions. However, inconsistencies between class and sequence diagrams revealed context and traceability limitations. Ferrari et al. [11] conducted a broader exploratory study using 28 NL requirement documents to generate UML sequence diagrams in PlantUML syntax. While the outputs adhered to UML standards and were generally understandable, they often lacked completeness and omitted key interactions, especially when inputs were ambiguous.

Collectively, these studies demonstrate that LLMs such as ChatGPT can generate UML models from NL, but their outputs are often incomplete, inconsistent, or reliant on human refinement. This underscores a critical research gap: the need for fully automated, reproducible, and validated approaches to UML model generation. Furthermore, a review of the evaluation methodologies in these existing works reveals a systemic challenge regarding quantitative benchmarking. Studies in this domain frequently evaluate their methods using qualitative human assessments based on completeness and understandability (e.g., Ferrari et al. [11]), scenario-based architecture analysis (e.g., Ahmad et al. [10]), or manual verification against researcher-created reference models (e.g., Cámara et al. [9]; Jahan et al. [14]). While some requirements engineering literature occasionally employs quantitative metrics such as Precision, Recall, and F-measure, these are predominantly applied to proprietary, in-house datasets. The lack of standardized, open-source benchmark datasets containing both NL requirements and their corresponding ground-truth UML models makes objective, numerical comparisons across different studies highly challenging.

2.4. Text Similarity Measurement

Text similarity measurement quantifies the degree of resemblance between two pieces of text in terms of content, structure, or meaning [33]. It is widely used in tasks such as information retrieval, translation evaluation, and text summarization [34]. In this research, similarity measurement is employed to evaluate the correctness of the UML sequence diagrams generated by the proposed method, by comparing their textual representations in the chosen intermediate syntax (PlantUML) against expert-validated oracle models.

Existing similarity metrics can be broadly divided into task-specific and general-purpose categories. Task-specific metrics, such as BLEU [35], ROUGE [36], and METEOR [37], were designed for evaluating machine translation or summarization systems. BLEU measures exact n-gram matches with brevity penalties, ROUGE emphasizes recall by capturing content overlap, and METEOR incorporates stemming and synonym matching to improve semantic evaluation. While effective for their original domains, these metrics are less suitable for this research, where structural rather than linguistic similarity is of primary concern and domain-specific synonym resources are unavailable.

General-purpose metrics, on the other hand, provide broader applicability. Gomaa and Fahmy [34] classify such methods into string-based, corpus-based, and knowledge-based approaches, corresponding respectively to lexical and semantic similarity dimensions. Among them, Cosine Similarity has proven effective for comparing documents of varying lengths, as it measures the cosine of the angle between two Term Frequency-Inverse

Document Frequency (TF-IDF) vectors, emphasizing direction rather than magnitude [38]. Lahitani et al. [39] applied this combination in an automated essay scoring system, comparing student essays against expert texts, demonstrating its practicality for textual similarity assessment.

3.0 THE PROPOSED METHOD

This research was structured into four systematic phases:

1. Problem Identification: Identifying the inefficiency and lack of automation in translating natural language requirements into behavioral UML models (as discussed in Section 1 and 2).
2. Artifact Design: Formulating the conceptual framework to leverage the GPT-4 model, define prompt structures, and establish deterministic parameters (detailed in Section 3).
3. Development: Constructing a proof-of-concept software prototype to operationalize the proposed method (detailed in Section 4).
4. Evaluation: Rigorously assessing the correctness of the generated artifacts using expert-validated oracle models and cosine similarity metrics (detailed in Sections 5 and 6).

Within the Artifact Design phase (Phase 2), the proposed method was designed to consist of four primary operational stages: (1) prompt construction, (2) generation of PlantUML statements, (3) validation, and (4) rendering. First, a structured prompt is constructed and sent to the GPT-4 model [26] to extract critical information from software FRs and generate corresponding PlantUML statements. Next, the generated PlantUML statements are validated to ensure syntactic and structural correctness. Finally, validated PlantUML statements are rendered into UML sequence diagrams. The overall workflow is illustrated conceptually in Fig. 1 below.

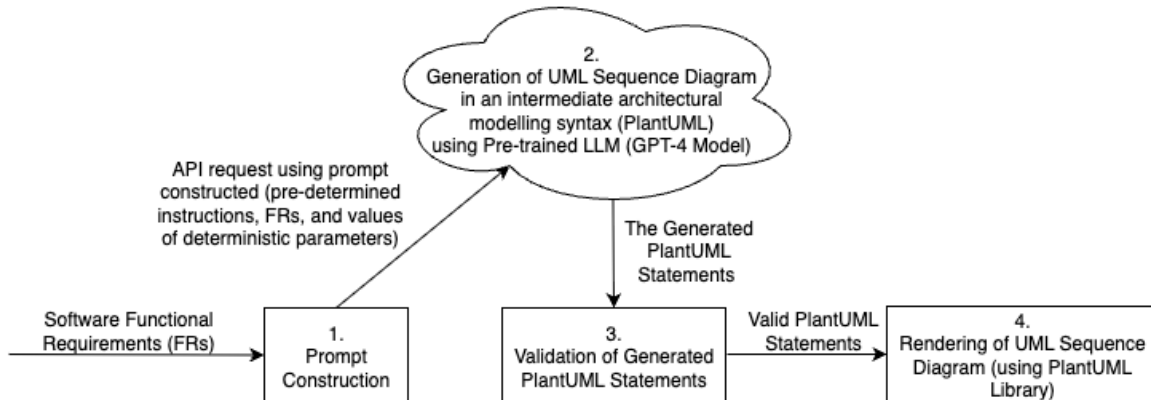


Fig. 1: Design of the proposed method

3.1. Leveraging GPT model

The proposed method leverages pre-trained LLM to extract information from NL software FRs. The decision to utilize pre-trained LLM in this research stemmed from the maturity of models that have undergone extensive training on large datasets which cannot be replicated easily and effectively, under limited time and individual resources. LLMs from various providers have been trained on a massive corpora of texts, encompassing a wide range of sources such as books, articles, and online content [40]. This comprehensive training exposes LLMs to diverse language patterns and contexts, thereby enhancing their ability to comprehend the input text accurately. According to Raiaan et al. [40], LLMs are bringing radical transformation to NLP applications by enabling remarkable performance improvements. Hence, the use of pre-trained LLMs eliminates the need for manual construction of linguistic rules and training NLP models from scratch.

As highlighted in Section 2.3, it is evident that the GPT model is more mature compared to other LLMs. Therefore, the GPT model has been selected as the LLM for this research. GPT-4, the latest version of GPT family by OpenAI, has been trained on nearly three times more data than GPT-3. It is engineered to comprehend longer and more complex contexts, with improved reasoning capability and creativity [26]. Consequently, GPT-4 has been selected as the LLM to be leveraged in this research to extract key information from FRs.

While GPT-4 provides the linguistic intelligence required to interpret NL inputs, additional modules are necessary to manage its output determinism, validate the generated syntax, and render the final UML sequence diagrams. Conceptual integration of these components is described in the following section.

3.2. Conceptual Framework for GPT-4 Integration

To achieve consistent and reproducible results, the proposed method integrates GPT-4 via API-based framework rather than relying on ChatGPT's conversational interface directly. Cámara et al. [9] identified several drawbacks when employing ChatGPT for the generation of software models, including syntactic and semantic flaws, inconsistency in responses, and scalability concerns. Such inconsistencies in responses, noted by Cámara et al. [9], are attributed to ChatGPT's default deterministic parameter.

In contrast, the API-based integration in the proposed method allows direct control of deterministic parameters such as temperature and seed, ensuring stability across repeated executions. The conceptual framework operates through three interconnected modules:

- A prompt generation module constructs a structured instruction combining a predefined system message with user-supplied FRs.
- The GPT-4 module processes the prompt and returns structured PlantUML statements that describe the actors, messages, and interactions derived from the requirements.
- The validation and rendering modules verify and visualize the statements as UML sequence diagrams.

Unlike manual prompting in ChatGPT, this integrated framework operates autonomously and deterministically, minimizing randomness in the responses while maintaining linguistic precision and scalability.

3.3. Generation of PlantUML statements Using GPT Model

During preliminary experimentation, it was observed that the GPT model may potentially generate ASCII-style sequence diagrams using plain characters to draw boxes and lines in text based as depicted in Fig. 2, making the parsing and transformation into actual graphical sequence diagrams very challenging. Using a structured and well-defined format facilitates easier transformation.

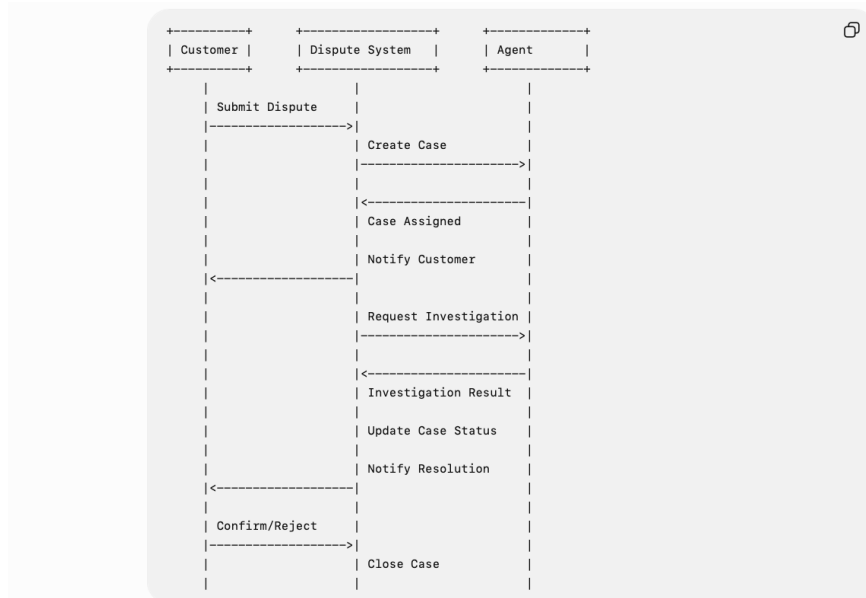


Fig. 2: ASCII-style UML sequence diagram in draw boxes and lines generated by the GPT model

According to Cámara et al. [9], GPT models are capable of producing models in different intermediate modelling syntaxes from various tools such as PlantUML [22], Mermaid [23], and UMLet [24]. Among these, PlantUML is selected in this research due to the following reasons:

- **Simple and Intuitive Syntax [22]:** PlantUML expresses UML semantics in short, readable statements (e.g., actor, participant, ->), which can be easily produced by both humans and language models.
- **Availability of an Official Rendering Library [41]:** PlantUML provides a library with a remote rendering interface, ensuring consistent UML sequence diagram generation across environments.

- **Reproducibility and Determinism** – Because diagrams are generated directly from PlantUML statements, identical PlantUML statements always yield identical diagrams, guaranteeing consistency and eliminating subjective interpretation during rendering.
- **Facilitation of Quantitative Evaluation** – PlantUML statements can be compared computationally through similarity measures.

Moreover, PlantUML syntax closely reflects the structure of its graphical output, providing a cohesive and predictable design experience. As demonstrated in Table 1, the PlantUML statements correspond directly to the graphical elements of UML sequence diagrams - for instance, the “->” symbol denotes a message exchanged between two participants.

Table 1: Sample PlantUML statements and corresponding UML sequence diagram

PlantUML Statements	UML Sequence Diagram
<pre>@startuml actor User actor Admin participant System User -> System: Select action System -> User: Prompt User -> System: Provide required info System -> Admin: Notify System -> User: Response @enduml</pre>	
<pre>@startuml participant Alice participant Bob Alice -> Bob: Authentication Request Bob -> Alice: Authentication Response Alice -> Bob: Another authentication Request Alice <- Bob: Another authentication Response @enduml</pre> [22]	[22]

3.4. Validation of Generated PlantUML Statements

Although GPT-4 generally produces syntactically correct PlantUML statements, instances of incomplete or invalid output have been observed. Such cases highlight the necessity of a validation stage to ensure the statements are suitable for rendering UML sequence diagrams. Validation guarantees that the generated PlantUML statements are correctly formatted and complies with PlantUML syntax and structural rules, thereby minimizing runtime errors during diagram generation.

The validation mechanism performs several checks:

- Ensuring that the PlantUML statements start with “@startuml” and end with “@enduml”, which are essential commands to mark the start and end of UML sequence diagram specifications.
- Verifying that all actors or components are correctly defined with the keywords “actor” or “participant”.
- Identifying formatting issues, such as ensuring that component names with spaces are enclosed in double quotes (“Target user”).
- Checking for syntax compliance, such as ensuring proper message flows using ->.

3.5. Rendering of PlantUML Statements into UML Sequence Diagram

The final stage of the proposed method involves translating validated PlantUML statements into graphical UML sequence diagrams.

As highlighted in Section 3.3, PlantUML provides a library with a remote rendering interface that enables the automatic generation of diagrams from PlantUML statements. Accordingly, the rendering process is handled by a UML sequence diagram renderer, which is part of the proposed method, utilizing the official PlantUML library.

The library interprets text-based commands such as participant, actor, and -> and converts them into corresponding graphical elements like lifelines, message arrows, and activation boxes.

The UML sequence diagram renderer ensures that the diagram layout is structured according to UML standards, automatically adjusting spacing and alignment to create a clean and professional representation. This automated process allows for the real-time transformation of the statements into diagrams, making it possible to update the visuals instantly whenever the PlantUML statements are modified. The result is a fully rendered UML sequence diagram that mirrors the underlying statements, ensuring consistency between the statements and its graphical representation throughout the lifecycle of the system design.

Together, these five sections describe the complete conceptual workflow of the proposed method from leveraging GPT-4 to interpret NL requirements through to the validation and rendering of UML sequence diagrams. The next section details the prototype implementation developed in Python to operationalize and evaluate this method.

4.0 PROOF-OF-CONCEPT PROTOTYPE

This section presents the prototype implementation developed to evaluate the technical feasibility of the proposed method described in Section 3.0. The idea of a fully automated method would remain a conceptual idea without an actual implementation; therefore, a proof-of-concept prototype was developed to demonstrate its feasibility. Through this implementation, new insights (such as the use of an intermediate modelling syntax, i.e. PlantUML) were uncovered, which influenced the final design of the proposed method.

The prototype, developed in Python programming language, serves to verify the technical feasibility of the proposed method. It accepts FRs in English as input and generates the UML sequence diagram without human intervention according to the provided software FRs. It leverages OpenAI's GPT-4 model [26] to enable NL comprehension and extraction of key information from the provided FRs and return the information in PlantUML statements. The PlantUML statements are subsequently validated and rendered into UML sequence diagrams.

The overall workflow has the following key components and stages:

- Prompt construction: Constructing the prompt to GPT-4 model via OpenAI API [42]
- PlantUML statements validation: Validating PlantUML statements before rendering
- Rendering: transforming the validated PlantUML statements into UML sequence diagrams using PlantUML library

4.1. Integration with GPT-4 model

In the prototype, the GPT-4 model serves as the core component responsible for interpreting FRs and generating corresponding PlantUML statements. Building upon the conceptual framework described in Section 3.2, the prototype integrates GPT-4 via the OpenAI API rather than through the ChatGPT conversational interface.

This integration allows deterministic control of the model's output and supports automated execution within the processing pipeline. During runtime, the prototype's Prompt Generation Module automatically constructs the input payload, which includes:

- A system message that defines the model's analytical role (e.g., "You are a software analyst that converts FRs into PlantUML sequence diagrams.").
- A user message containing user supplied FRs.
- Configuration parameters that govern determinism (e.g., temperature = 0, seed = 12345).

The API returns the generated PlantUML statements in JSON format, which are passed directly to the validation and rendering modules.

4.2. Deterministic Configuration Parameters

To ensure consistency and reproducibility of results, the prototype incorporates two deterministic control parameters - seed and temperature in each API request to GPT-4. These parameters reduce the inherent stochasticity of the model's token-sampling process.

4.2.1. Seed Parameter

According to the official cookbook by OpenAI, to ensure that the responses generated by the model are more consistent and reproducible, the seed parameter should be set in the request made to the GPT model [43].

The seed parameter is a deterministic control mechanism that ensures the consistency and reproducibility of responses from the GPT-4 model. The seed parameter can be any integer value and when the same seed value is

used across multiple requests, the model attempts to generate the same output given identical input conditions. Without a fixed seed, different requests with the same input may produce slightly different outputs due to randomness in token sampling. This approach helps maintain consistency in the responses and addresses the inconsistency issue highlighted in the study by Cámara et al. [9].

During the prototype implementation, various seed values were tested, but no noticeable differences were observed in the generated responses when a seed value was included. By setting the seed to 12345, the model's response remains stable and reproducible, ensuring that the generated PlantUML statements are consistent across multiple executions. The numerical value 12345 is arbitrary but serves as a predefined value to ensure deterministic behaviour. Any integer could be used, but 12345 was selected for simplicity and uniformity in experimental settings.

4.2.2. Temperature Parameter

Like the seed parameter, the temperature is optional and accepts a value between 0 and 1. This parameter controls the sampling behaviour, with higher values resulting in more random outputs and lower values producing more focused and deterministic outputs. The temperature parameter controls the degree of randomness in the GPT-4 model's output generation. A higher temperature (e.g., 0.8 or 1.0) makes the output more random, allowing for diverse and creative responses. A lower temperature (e.g., 0.1 or 0) makes the output more deterministic, ensuring that the most likely response is chosen.

During the prototype implementation, adjusting the temperature parameter showed significant effects: higher values, such as 0.8, resulted in inconsistent outputs even with identical requests, while lower values, such as 0.1, generated highly consistent outputs for the same request. The prototype is using a temperature value of 0 forces GPT-4 to produce the most probable response for a given input. This is critical for generating consistent and accurate UML sequence diagrams, as variations in output could introduce inconsistencies in modelling, ensuring a structured and reliable output format.

4.3. Prompt Engineering and Design

The limitations of previous work, such as Camara et al. [9] highlighted the importance of well-designed prompts. Their approach provided limited application domain context. Consequently, the quality of the generated responses heavily depended on the model's creativity, potentially leading to the unsatisfactory outcomes highlighted by Camara et al. [9]. Hence, to address this limitation, the proposed method utilizes a prompt with a clear and specific system message which has been fine-tuned along with the user-supplied FRs provided to the GPT model, in line with the prompt engineering recommendations suggested by Ekin [44].

In the GPT model, the system message directs the model's behaviour by providing specific instructions or context, while user messages convey requests or comments for the model to process. The proposed method uses a predefined system message and user-supplied FR set (as detailed in Table 2) to guide the model. The prompt fine-tuning process involved two iterations, focusing on refining the system message while keeping the user message consistent across iterations. The selected FR set is shown in Table 2.

First Iteration: In this research, the initial requests to the GPT model were designed with a predefined system message set as "You will be provided with a block of software requirement document; you should generate the corresponding sequence diagram.". The user message is the FR set. However, this setup resulted in the generation of sequence diagrams using plain characters, as depicted in Section 3.3.

Second Iteration: It became apparent that specifying the intermediate modelling syntax such as PlantUML [22], Mermaid [23] and UMLet [24] is necessary, as the system message was not specific enough, and the expected output was not clearly specified. Therefore, the predefined system message has been refined as follows: "*You will be provided with a block of software requirement document, and your task is to generate a correct and complete sequence diagram with valid PlantUML syntax*". The request made to the GPT model is as shown in Table 2. With this setup, the response from the GPT model is the PlantUML statements as illustrated in Table 3, which is ready for transformation into actual graphical sequence diagrams.

4.4. PlantUML Statements Validation Module

A validation module was developed and integrated into the software prototype to validate the PlantUML statements before rendering them into UML sequence diagrams.

The validation process consists of multiple rule-based checks as described in Section 3.4 and fully automated. Once the GPT model generates the PlantUML statements, they are immediately passed through the validation

module before rendering. If the validation module identifies the PlantUML statements as having syntax or structural errors, an error flag is returned to avoid the unnecessary rendering of the UML sequence diagram.

Currently, the validation module functions solely as a validator, returning errors if an invalid PlantUML statements block is encountered before attempting to render the UML sequence diagram. It lacks the capability to automatically correct invalid statements, which is a feature that could further enhance the user experience of the proposed method.

Table 2: Request parameters to GPT model

Model	gpt-4-1106-preview
System Message	“You will be provided with a block of software requirement documents, and your task is to generate a correct and detailed sequence diagram with valid PlantUML syntax.”
User Message	Dispute Management System • Users must be able to select disputable transactions within the system. • The system must prompt users to enter required information when initiating a chargeback request or ticket retrieval request. • Upon the creation of a dispute request, the system must send a notification to the respective merchant. • The system must provide a confirmation to the user upon the creation of a ticket retrieval request or chargeback request. • Users should be able to track the status of their dispute requests (e.g., pending, in progress, resolved). • Merchant should be able to respond to dispute requests by providing relevant information and evidence. • Users should receive notifications about updates on their dispute requests, such as when a merchant responds or when the dispute is resolved.
Seed	12345
Temperature	0

Table 3: Generated PlantUML statements

<pre> @startuml title Dispute Management System actor User participant System participant Merchant User -> System: Select disputable transaction User -> System: Submit dispute request (Chargeback / Ticket Retrieval) System -> Merchant: Notify merchant of new dispute System -> User: Confirm dispute creation Merchant -> System: Submit response (info, evidence) System -> User: Notify merchant responded User -> System: Check dispute status System -> User: Show current status (Pending / In Progress / Resolved) System -> User: Notify when resolved @enduml </pre>
--

Upon successfully passing the validation module, the syntactically correct PlantUML statements are processed and rendered into their final visual representations. To demonstrate the practical capabilities of the developed prototype, several UML sequence diagrams were generated.

Fig. 3 illustrates a sample sequence diagram rendered from the generated PlantUML statements in Table 3. As depicted, the prototype accurately extracts and maps the natural language requirements into the appropriate UML notations, successfully identifying the primary actors (e.g. User), system lifelines, and the correct chronological flow of synchronous and asynchronous messages (e.g., submitting requests and notifying status).

5.0 EVALUATION

It is critical to conduct a comprehensive evaluation of the correctness of the software models (specifically UML sequence diagrams) generated by the prototype and the underlying proposed method, given the capability of the proposed method to generate the UML sequence diagram automatically from software FRs. An evaluation was conducted involving 6 experts and assessed correctness by calculating the similarity between the oracle model (i.e., the sample answer validated by the experts) and the model generated by the prototype for the same set of FR. A higher similarity score indicates a higher correctness of the respective model generated by the prototype. The overall evaluation workflow, including participant involvement, oracle model construction, and similarity measurement, is illustrated in Fig. 4.

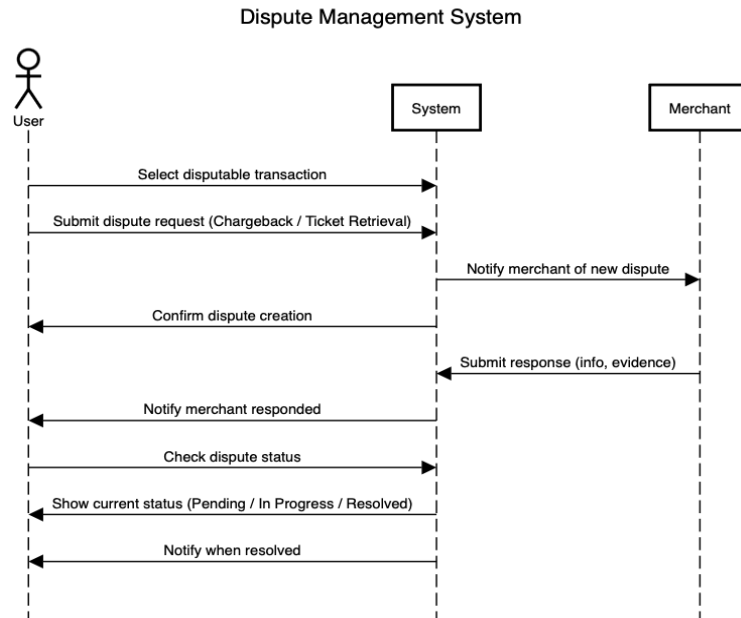


Fig. 3: Sequence diagram rendered from the generated PlantUML statements for a Dispute Management System.

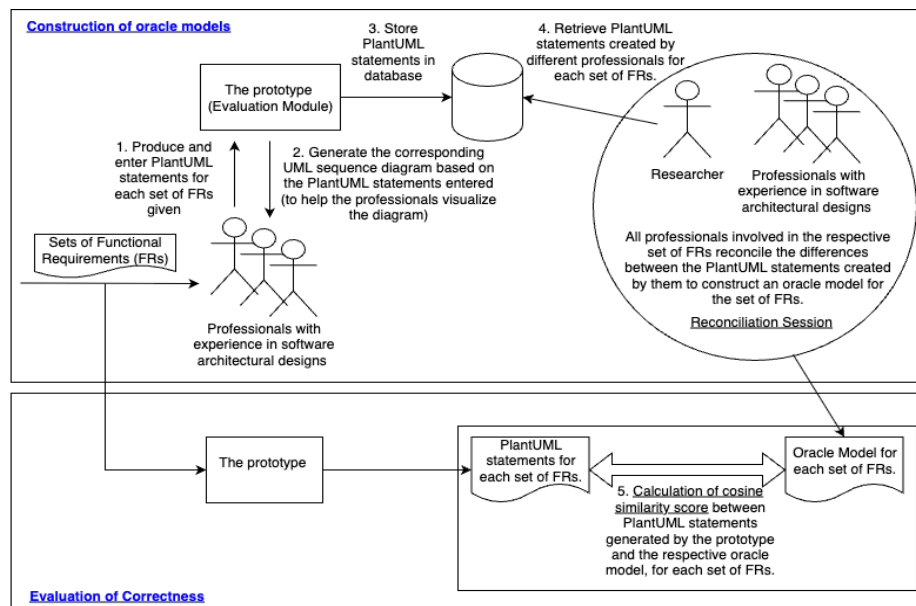


Fig. 4: Evaluation workflow of the proposed method

5.1. Evaluation By Comparing with Oracle Model

Evaluating sequence diagrams is often subjective and depends on an individual's expertise and understanding. As noted in the existing literature [13, 14], few approaches are fully automated in generating sequence diagrams from textual user requirements, as is the case with the proposed method. However, comparing results with other existing work is challenging due to variations in input structure and the degree of human intervention involved. Therefore, comparing results with other studies may not always be reliable.

To the best of the researcher's knowledge, there are no existing datasets comprising FRs and their corresponding "oracle" models, nor prior evaluations of a similar nature. Hence, it is necessary to construct an oracle model for each set of FRs, to serve as ground truth in the correctness evaluation.

To construct the oracle models, experts who are practicing professionals in the software engineering field were recruited to produce PlantUML statements representing UML sequence diagrams based on the provided FRs. The PlantUML statements produced by each different participant (the experts) for the same set of FRs were then compared, reconciled, and consolidated into a single oracle model, which was subsequently used for correctness evaluation.

The final stage of data collection involved generating system models using the prototype developed in this research. For each set of FRs, the researcher provided the FRs to the prototype, which automatically generated the corresponding PlantUML statements. These PlantUML statements are called the generated models in this research.

Inspired by Jahan et al. [14], a similarity measure was used to evaluate whether a sequence diagram generated by the prototype is correct and suitable for practical use. As explained in Section 3.3, its corresponding PlantUML statements directly determine each UML sequence diagram, hence correctness was assessed by comparing the PlantUML statements of the generated models against the oracle models.

Using an expert validated oracle model in the evaluation process is crucial to minimizing bias. In this research, the study participants (i.e. the experts) produced the PlantUML statements of the oracle models through a reconciliation process, during which the statements from individual participants were consolidated and validated. This third-party validation was essential to ensure that the evaluation result is less susceptible to bias.

In conclusion, this evaluation approach used in this research addresses the subjectivity challenges in evaluating the correctness of the generated models. Using third-party validation of oracle models further strengthens the reliability of the evaluation results.

5.2. Preparation for Evaluation Study

Prior to the commencement of the evaluation study, an evaluation module was developed and integrated into the software prototype to facilitate the assessment of the correctness of UML sequence diagrams generated using the proposed method.

The purposes of the evaluation module are outlined below:

- **Facilitate Participants Input** – The module is to provide a structured environment to the participants to manually create sequence diagrams based on their interpretation of the assigned FR set.
- **Automated Data Collection** – The submitted diagrams are stored in the software prototype's database for further analysis and comparison.
- **Visual Validation** – The module includes a built-in PlantUML rendering function that allows participants to immediately visualize their diagrams as they update the PlantUML statements.

The module is integrated into the software prototype to share infrastructure. It is designed exclusively for evaluation purposes and will not be accessible to regular users. The module allows participants to refer to the assigned FR set while simultaneously crafting a sequence diagram based on their interpretation of these requirements, using PlantUML syntax.

During the drafting process of the sequence diagram, participants were able to visualize the real-time of the sequence diagram as they update the PlantUML statements, as illustrated in Fig. 5. This feature ensured a smooth transition from the initial draft to the final diagram. Once a participant has completed their sequence diagram, they submit the corresponding PlantUML statements, which are stored in the prototype's database for further analysis. The collected submissions were later used in the reconciliation process to construct the oracle models.

5.3. Functional Requirements

For this evaluation study, ten (10) sets of FR were sourced from various channels, including books and open-source datasets. All sets are related to the web application domain and are expressed in English. The complexity is consistent across the sets, which cover well-known applications such as ride-sharing systems, restaurant ordering and reservation systems, and ticket booking systems. Each set contains seven (7) to twelve (12) lines of

requirements. The ten (10) FR sets were divided into two bundles: FR Sets 1-5, and FR Sets 6-10, which were later assigned to participants in the study.

Functional Requirements

Select a system's functional requirement*
Ticket Booking System

Functional Requirement Description

1. Users should be able to browse upcoming events by category, date, or location.
2. Users should be able to view detailed information about each event, including its description, date, time, venue, and available tickets.
3. Users should be able to select the number of tickets they want to purchase for an event.
4. Users should be able to view the contents of their shopping cart, including the selected tickets and the total price.
5. Users should be able to modify the quantity of tickets in their cart or remove tickets from the cart.
6. Users should be able to proceed to checkout and complete their ticket purchase securely.
7. Users should be able to choose from various payment methods and enter payment details.
8. Users should receive a confirmation email after successfully purchasing tickets.

Sequence Diagram

PlantUML Syntax

```
@startuml
actor User
participant "Event Browser" as EB
participant "Event Details" as ED
participant "Shopping Cart" as SC
participant "Checkout" as CO
participant "Payment" as PM
participant "Email Service" as ES

User->>EB: Browse events by category, date, or location
EB->>User: Display upcoming events
User->>ED: View event details
ED->>User: Display event information
User->>SC: Select number of tickets
SC->>User: Update shopping cart
User->>SC: View shopping cart
SC->>User: Display shopping cart contents
User->>SC: Modify or remove tickets
SC->>User: Update shopping cart
User->>CO: Proceed to checkout
CO->>User: Display checkout options
User->>PM: Choose payment method and enter details
PM->>User: Enter payment details
User->>PM: Complete purchase
PM->>User: Display purchase confirmation
User->>ES: Receive confirmation email
ES->>User: Send confirmation email
end
```

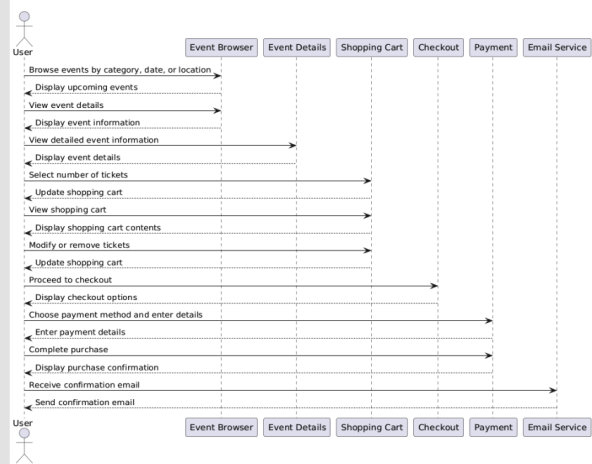


Fig. 5: Evaluation module integrated in the software prototype

5.4. Study Participants

Software design requires a comprehensive understanding of software design principles, system requirements, and architectural patterns. According to Bass, Clements, and Kazman [45], software architects must balance technical and business requirements while ensuring adherence to established design principles and architectural styles. Individuals with such expertise are well positioned to evaluate the correctness and practical relevance of UML sequence diagrams generated by the proposed method.

Participants were selected based on predefined eligibility criteria to ensure both relevant expertise and diversity of professional experience. This selection process aimed to include individuals who could offer valuable insights into the proposed method. The participants chosen are software architects and software engineers with backgrounds in software engineering, software architecture, and requirements engineering. The eligibility criteria are as follows:

- Are professionals with expertise in software engineering, software architecture, and requirements engineering.
- Have experience in working with software architectural designs.
- Have experience in working with PlantUML syntax.

The eligibility criteria welcomed professionals with various titles - such as software engineers or requirements engineers. However, the recruitment process yielded participants who currently hold the title of Software Architect. This is advantageous for the study, as the architect role is typically a senior position that inherently encompasses extensive experience in software design, engineering, and requirements analysis. As reflected in Table 4, five of the six participants also possess significant prior experience operating specifically as software engineers, effectively fulfilling the mixed-expertise criteria.

These criteria ensured that all participants possessed the technical competence to contribute to the evaluation of the correctness of the generated UML sequence diagrams by the proposed method. The participants were recruited through various channels to ensure a diverse pool representing relevant expertise in the field, which includes:

- Invitations sent through professional network platform (e.g. LinkedIn)
- Contacting individuals directly based on their known involvement in similar studies or projects.

The data collection from the participants was carried out in a two (2) step approach. The first step focused on collecting participants' details for demographic analysis purposes. Participants were first asked to complete a Google Form designed to capture demographic and background information. This form collected details such as professional role, years of experience, and familiarity with UML and PlantUML. The responses were automatically recorded and securely stored within Google Forms during the data collection period.

In the second step, participants were provided access to the software prototype developed by the researcher. Each participant received five (5) sets of FR, each describing different system scenarios. For each assigned FR set, participants were required to construct a sequence diagram by producing the corresponding PlantUML statements within the evaluation module. Upon completion, participants submitted their PlantUML statements directly through the module interface. All submitted data were recorded and persisted in the prototype's database.

5.5. Participants Demography

A total of six (6) participants have been recruited, comprising software architects and software engineers with experience in software architectural design. While the sample size consists of six (6) participants, this number is strictly justified by the qualitative, expert-driven nature of the study, as the evaluation required a high cognitive load and time commitment - specifically, analyzing complex FR sets and manually writing corresponding PlantUML statements. Additionally, this setup generated a robust dataset of 30 individually evaluated scenarios (6 participants $\times$ 5 FR sets each), providing ample data to thoroughly assess the correctness of the generated UML sequence diagrams.

Table 4 presents the demographic profile of the six (6) participants who took part in the evaluation study. All participants have prior experience working with system architectural designs, and two (2) of them have previously used PlantUML to create UML diagrams. While all participants are software architects, their years of experience vary, with two (2) having more than five (5) years of experience, and four (4) having less than five (5) years. Moreover, five (5) participants also have at least three (3) years of experience as software engineers.

Participants were split into two groups reflecting different levels of experience. As can be seen in Table 4, Participant Group 1 consisted of more experienced software architects, with each individual having at least three (3) years of experience. Participant Group 2 comprised less experienced software architects, each with less than two (2) years of experience. To avoid overburdening any individual participant, Participant Group 1 was assigned to work on FR Sets 1 – 5, and Participant Group 2 was assigned to work on FR Sets 6 – 10.

Table 4: Demography of participants

Participant Group	Participant	Do you currently work as a:	Years of exp. as software architect	Years of exp. as software engineer	Have you worked with system architectural designs in your professional experience?	Have you used PlantUML to create UML diagrams before?
1 (FR Set 1 – 5)	1	Software Architect	> 5 years	> 5 years	Yes	Yes
	2	Software Architect	> 5 years	> 5 years	Yes	No
	3	Software Architect	3 - 4 years	3 - 4 years	Yes	Yes
2 (FR Set 6 – 10)	4	Software Architect	1 - 2 years	> 5 years	Yes	No
	5	Software Architect	1 - 2 years	Never	Yes	No
	6	Software Architect	1 - 2 years	3 - 4 years	Yes	No

5.6. Identify Differences, Reconcile and Construction of Oracle Models

The construction process began by identifying differences among each participant's PlantUML statement submissions for each set of FR, focusing on elements such as the sender, receiver, and sequence flows. For each difference identified, the three (3) participants assigned to that specific FR set were invited to a discussion session to review the variations and reach a consensus on the most accurate representation. The agreed-upon version was then consolidated into the final oracle model for that FR set. For instance, Table 5 shows parts of the PlantUML statements submitted by the participants involved in FR Set 1 (Dispute Management) alongside the corresponding resulting oracle model. At the end of this process, participants produced and validated an oracle model for each of the ten (10) FR sets used in this evaluation study.

Table 5: Reconciliation of the 3 PlantUML statements submissions and resulting Oracle model for FR Set 1

Participant	PlantUML Statements Submitted	The resulting Oracle model
Participant 1	@startuml actor user as user participant system as system actor merchant as merchant user -> system: retrieve disputable transactions menu ...	@startuml actor user participant system actor merchant user -> system: get disputable transactions ...
Participant 2	@startuml actor user participant system actor merchant user -> system: get disputable transactions ...	
Participant 3	@startuml actor user participant system actor merchant user -> system: select transactions (transaction) ...	

5.7. Evaluation Metrics

For each set of the FR, the similarity between the oracle model and the generated model was measured using cosine similarity to assess the correctness of the generated model. The cosine similarity, applied on TF-IDF vectors derived from the PlantUML statements, quantifies how closely the generated model aligns with the oracle model. A higher similarity score indicates stronger alignment and thus higher correctness in representing the FR.

According to *Rahutomo et al.* [46], cosine similarity is widely adopted in NLP tasks as it focuses on the alignment of vector representations and being robust to variations in text length and word frequency, making it well-suited for comparing structured textual data like PlantUML statements.

In this research, the similarity score serves as the primary evaluation metric, providing a quantitative measure of how closely the generated model matches its corresponding oracle model at both the structural and interaction levels. Structural similarity measures how closely the two models resemble each other in terms of their organization and components. It looks at factors like the types of components used such as classes or modules, how these components are represented (as actors or objects and so on). On the other hand, interaction level similarity focuses on how the components communicate to each other (synchronous or asynchronous), analysing the messages exchanged between them, and how they respond to events. Together, these two types of similarity help determine how well a generated model matches an oracle model in both structure and functionality.

TF-IDF with cosine similarity was selected after reviewing alternative metrics. It was preferred for its semantic robustness, independence from text length, and low computational cost, as it does not require additional linguistic resources like METEOR or BLEU. By focusing on vector alignment, cosine similarity provides a flexible and comprehensive means of assessing the correctness of GPT generated models.

5.8. Calculation of Similarity Scores

The similarity measure begins by building a vocabulary (a list of unique terms) from the PlantUML statements of both the oracle model and the generated model. Then, the term frequency-inverse document frequency (TF-IDF) algorithm was used to transform the PlantUML statements of the oracle model and the generated model into vector representations of words, enabling efficient computation and comparison for various subsequent NLP tasks.

According to Bafna, Pramod, & Vaidya [47], TF-IDF is a widely used weighting scheme in information retrieval and text mining that structures a document by calculating a numerical representation for each word based on its frequency in the document and its rarity across the entire vocabulary. The overall TF-IDF value increases with the frequency of a word in the document, however the IDF component serves to diminish the importance of terms that appear too commonly across many documents. Subsequently, the similarity between the two vectors is calculated using cosine similarity, which measures the cosine of the angle between two vectors in a multi-dimensional space [48]. The formula for cosine similarity is:

$$\text{Cosine Similarity} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (1)$$

where

1. A and B are the TF-IDF vector representation of the oracle and generated model
2. A_i and B_i are the i^{th} components of the vectors A and B

The following describes an example on how the similarity score was calculated for the PlantUML statements of the oracle model and generated model shown in Table 6.

Table 6: The Oracle Model and Generated Model for a specific FR set

Oracle model	Generated model
@startuml	@startuml
actor user	actor user
participant system	participant system
user -> system: get disputable transactions	user -> system: get transactions
system -> user: display disputable transactions	system -> user: show transactions
@enduml	@enduml

The TfidfVectorizer (from the scikit-learn library [49]) was used to represent the textual content of both the oracle and generated models numerically for comparison. TfidfVectorizer first built a shared vocabulary, which is a list of all unique terms across both documents: {"@startuml", "actor", "user", "participant", "system", "get", "disputable", "transactions", "display", "show", "@enduml"}.

Each model is then represented as an eleven-dimensional TF-IDF vector, with one value corresponding to each term in the shared vocabulary. Terms that do not appear in a given model are assigned a TF-IDF value of zero. This uniform vector length ensures that the two models can be directly compared using similarity measures such as cosine similarity.

TfidfVectorizer then calculated the TF-IDF values for each term in each model. The resulting TF-IDF values for each model as below:

- TF-IDF Vector for oracle model: [0.1, 0.5, 0.3, 0.5, 0.4, 0.2, 0.7, 0.5, 0.2, 0.0, 0.1]
- TF-IDF Vector for generated model: [0.1, 0.5, 0.3, 0.5, 0.4, 0.2, 0.0, 0.5, 0.0, 0.3, 0.1]

Then, the cosine similarity score was calculated as below:

$$\begin{aligned} & [(0.1 \times 0.1) + (0.5 \times 0.5) + (0.3 \times 0.3) + (0.5 \times 0.5) + (0.4 \times 0.4) + (0.2 \times 0.2) + \\ & \quad (0.7 \times 0.0) + (0.5 \times 0.5) + (0.2 \times 0.0) + (0.0 \times 0.3) + (0.1 \times 0.1)] \\ & \frac{\sqrt{(0.1^2 + 0.5^2 + 0.3^2 + 0.5^2 + 0.4^2 + 0.2^2 + 0.7^2 + 0.5^2 + 0.2^2 + 0.0^2 + 0.1^2)}}{\sqrt{(0.1^2 + 0.5^2 + 0.3^2 + 0.5^2 + 0.4^2 + 0.2^2 + 0.0^2 + 0.5^2 + 0.0^2 + 0.3^2 + 0.1^2)}} \times \\ & = \frac{[0.01 + 0.25 + 0.09 + 0.25 + 0.16 + 0.04 + 0.00 + 0.25 + 0.00 + 0.00 + 0.01]}{\sqrt{1.59} \times \sqrt{1.15}} \end{aligned}$$

$$= \frac{1.06}{1.352}$$

$$= 0.784$$

Since TF-IDF vectors contain only non-negative values, the value of the cosine similarity score calculated based on these vectors ranges from 0 (completely dissimilar) to 1 (completely similar) [50]. The cosine similarity score was converted to percentage by multiplying it by 100 and is called similarity score in Section 6.0.

6.0 RESULTS AND DISCUSSION

Table 7 shows the similarity scores calculated for each of the 10 FR sets. The similarity scores range from 74.72% to 93.47%. The highest similarity score is observed in Set 9 (93.47%), followed by Set 10 (92.28%) and Set 2 (91.30%). The lowest similarity score is in Set 3 (74.72%) followed by Set 7 (75.29%). The average similarity score across the 10 FR sets is 86.14%.

Table 7: Similarity scores

Participant Group	FR Set	Description	Similarity Score
Participant Group 1	Set 1	Dispute Management System	88.85%
	Set 2	Timeoff Request Management System	91.30%
	Set 3	Ticket Booking System	74.72%
	Set 4	Guest Checkout	86.41%
	Set 5	Restaurant Ordering System	85.23%
Participant Group 2	Set 6	Book Store System	88.72%
	Set 7	Student Course Registration System	75.29%
	Set 8	Restaurant Reservation System	85.16%
	Set 9	Ride Sharing System	93.47%
	Set 10	E-Learning Management System	92.28%

6.1. Analysis of Low Similarity Score Set

To identify possible reasons for the lower similarity scores (around 75%) for FR Set 3 (Ticket Booking System) and FR Set 7 (Student Course Registration System), their generated model and the oracle model were carefully compared. As shown in Table 8, for FR Set 3 (Ticket Booking System), the generated model included additional components, such as shopping cart and checkout service, as separate components, reflecting a modular architecture. In contrast, the oracle model followed a monolithic design, encapsulating these functionalities within the larger ticket booking system, treating them as internal processes rather than distinct components. This difference does not imply that either model is wrong; rather, both represent valid software models with different modularity.

Similarly, for FR Set 7 (Student Course Registration System) as shown in Table 9, the generated model abstracted the course-related operations into a standalone “Course Catalog” component, which introduces a decoupled design pattern. Meanwhile, the oracle model maintained a tightly coupled architecture where the system directly managed database interactions. Again, this reflects a difference in architectural interpretations rather than correctness. The GPT-4 model used in the prototype appears to favour a modular design approach, possibly due to patterns learned from training data in modern web application development.

In this study, these differences are not treated as errors but as valid alternative interpretations of the same FR set. They reflect variation in the level of architectural abstraction rather than lapses in correctness. For example, the oracle models produced by the participants favoured a monolithic approach, emphasizing direct interaction for simplicity and centralized control. In contrast, the generated model leans towards a modular or service-oriented approach, aligning more closely with micro-services design principles. Such divergences inevitably arise when FRs are interpreted without accompanying NFRs (e.g. modularity, scalability, reliability) or architectural constraints such as technology stack or architectural style (e.g., layered or microservices architecture), forcing the GPT models to rely on unverified assumptions and its own creativity. Ultimately, neither approach can be considered universally better. The suitability of each depends entirely on the requirements, which were not explicitly defined in the FR.

Overall, most of the FR sets demonstrate high similarity scores, suggesting that the prototype and the underlying proposed method are capable of generating UML sequence diagrams (in the form of PlantUML statements) that resemble those constructed manually by practicing professionals in the software engineering field.

The relatively low similarity scores for FR Set 3 and FR Set 7 do not diminish the overall performance of the prototype. Instead, these outliers reflect differences in architectural assumptions - implicit design interpretations made when FRs lack explicit guidance. For instance, GPT-4 may assume a microservices structure while experts model a monolithic one, or infer synchronous rather than asynchronous communication. Future work could focus on refining the system's robustness by explicitly incorporating NFR or architectural constraints into the FRs.

6.2. Analysis of Participant Group Results

Table 4 shows that Participant Group 1 has more years of experience (at least 3 years) as software architects and two (2) of the participants in this group have prior experience using PlantUML. Participant Group 2 has generally less experience (1 – 2 years), and none have used PlantUML before. Interestingly, despite these differences, both groups achieved comparable results as shown in Table 7, where the similarity scores (between the generated models and the oracle models) for the different sets of FRs handled by both groups of participants are comparable.

Table 8: FR Set 3, Oracle Model and Generated Model

FR Set 3	• Users should be able to browse upcoming events by category, date, or location. • Users should be able to view detailed information about each event, including its description, date, time, venue, and available tickets. • Users should be able to select the number of tickets they want to purchase for an event. • Users should be able to view the contents of their shopping cart, including the selected tickets and the total price. • Users should be able to modify the quantity of tickets in their cart or remove tickets from the cart. • Users should be able to proceed to checkout and complete their ticket purchase securely. • Users should be able to choose from various payment methods and enter payment details. • Users should receive a confirmation email after successfully purchasing tickets.
Oracle Model	Generated Model
<pre>@startuml actor user as user participant system as system participant "payment gateway" as payment participant "email service" as email user -> system: browse upcoming events (category, date, location) system -> user: upcoming events user -> system: view event detail system -> user: display event detail user -> system: add ticket to shopping cart (quantity) user -> system: get cart system -> user: cart details user -> system: update cart items (item, quantity) system -> user: return success system -> user: payment method user -> system: payment details system -> system: process payment system -> user: display payment success page system -> email: send confirmation email email -> user: send confirmation email @enduml</pre>	<pre>@startuml actor user participant "ticket booking system" as system participant "shopping cart" as cart participant "checkout" as checkout participant "payment gateway" as payment participant "email service" as email user -> system: browse events user -> system: search events system -> user: display event details user -> system: select tickets system -> user: display available tickets user -> cart: view shopping cart cart -> user: display cart contents user -> cart: modify cart user -> cart: remove tickets user -> checkout: proceed to checkout checkout -> payment: choose payment method payment -> checkout: confirm payment checkout -> email: send confirmation email email -> user: send confirmation email @enduml</pre>

Participant Group 1 achieved similarity scores ranging from 74.72% to 91.30%, with an average of 85.30%. Participant Group 2's scores ranged from 75.29% to 93.47%, with a slightly higher average of 86.98%. Notably, Participant Group 2 achieved the highest similarity score (93.47%), which is for FR Set 9.

The sets of FRs assigned to both groups of participants are similar in nature and complexity, focusing on straightforward and well-defined interactions within the web application domain. This comparability likely

contributed to the similar performance observed across both groups, despite the differences in their experience levels and prior familiarity with PlantUML. The consistent result suggests that the variation in similarity scores may be more influenced by the complexity or clarity of the FRs rather than the participants' experience.

This finding suggests that the prototype (that implements the proposed method) is well-suited for handling simple FR sets to produce correct outputs. However, for a more complex FR set, additional studies would be necessary to assess whether the proposed method can maintain at an acceptable correctness.

6.3. Comparison with Related Work

Table 10 presents a comparative summary of the proposed method against five key related studies, which are Cámara et al. [9] Ahmad et al. [10], Ferrari et al. [11], Kim [12], Gulia and Choudhury [13] and Jahan et al. [14]. This comparison highlights the advancements and contributions of the proposed method in addressing existing gaps (including lack of third-party validation, dependence on human intervention, and inconsistent outputs) in LLM utilization for automated software design.

Table 9: FR Set 7, Oracle Model and Generated Model

FR Set 7	● Students should be able to browse available courses by category, instructor, or keyword. ● Students should be able to search for specific courses by keyword. ● Students should be able to view detailed information about each course, including its description, instructor, schedule, and availability. ● Students should be able to enrol in a course if it's available and they meet any prerequisites. ● Students should be able to view a list of courses they are currently enrolled in. ● Students should be able to drop a course they are enrolled in. ● Students should receive a confirmation email after successfully enrolling in or dropping a course.
Oracle Model	Generated Model
<pre> @startuml actor student participant "enrollment system" as system participant "email service" as email participant "course database" as database student -> system: browse available courses (category, instructor, keyword) student -> system: search course (keyword) system -> database: retrieve course list database -> system: available courses system -> student: display available courses student -> catalog: view course details system -> database: get course details database -> system: course details (description, instructor, schedule, availability) system -> student: display course details student -> system: enroll course system -> database: check availability and eligibility database -> system: confirm enrollment system -> student: enrollment successful student -> system: view enrolled courses system -> database: fetch enrolled courses database -> system: enrolled courses list system -> student: display enrolled courses student -> system: drop course system -> database: update enrollment status system -> email: send confirmation email email -> student: send confirmation email @enduml </pre>	<pre> @startuml actor student participant "course catalog" as catalog participant "enrollment system" as system participant "email service" as email student -> catalog: browse courses by category, instructor, or keyword student -> catalog: search for specific courses by keyword catalog -> student: display available courses student -> catalog: view course details catalog -> student: display course details student -> system: enroll in a course system -> student: enrollment confirmation student -> system: view enrolled courses system -> student: display enrolled courses student -> system: drop a course system -> student: drop confirmation system -> email: send confirmation email @enduml </pre>

Unlike prior studies [9, 10, 12] that have relied on LLM powered chatbots such as ChatGPT which required iterative human interaction, the proposed method offers full automation by leveraging the GPT-4 model via API, and applies deterministic parameters to ensure consistent output. Furthermore, the correctness of the generated sequence diagrams was checked against oracle models (sample answers) constructed by third-party SE professionals by calculating cosine similarity scores, enhancing the objectivity and reliability of the results.

Among these, several studies explored the automatic generation of UML sequence diagrams. Ferrari et al. [11] evaluated ChatGPT using a one-shot prompting method to assess its capability in producing UML sequence diagrams. Although the generated diagrams adhered to UML syntax and were understandable, the study reported frequent omissions of key elements and a lack of completeness, especially for ambiguous inputs. Ahmad et al. [10] proposed a human-bot collaborative framework with ChatGPT to generate UML sequence diagrams through iterative dialogue. While the method produced relevant diagrams, it required multiple rounds of user interaction, resulting in inefficiency and limited automation.

Other studies focused on generating different types of UML diagrams. Cámara et al. [9] investigated ChatGPT's potential for UML class diagram generation, identifying issues such as inconsistent responses, dependency on domain familiarity, and syntactic errors. Kim [12] evaluated ChatGPT's capability across multiple software development phases and found that while the model could produce structured textual representations of class and sequence diagrams, these required manual visualization and were often incomplete or inconsistent with prior artifacts.

In contrast, Gulia and Choudhury [13] explored an automated approach for deriving UML activity diagrams from textual requirements without leveraging LLMs or chatbots. Their rule-based framework demonstrated potential however evaluation details were not reported, and the results claimed by the authors were based on a small-scale dataset, limiting the generalizability of their findings. Finally, Jahan et al. [14] employed a rule-based approach for generating UML sequence diagrams; however, their evaluation measured similarity between the generated outputs and use case scenarios that the researchers themselves had constructed by extracting nouns, verbs, and interactions from the FR. However, this approach lacked independent validation, potentially introducing bias into their evaluation results.

Collectively, these studies underscore the limitations of prior LLM-based approaches and provide the motivation for the proposed method, which aims to achieve more consistent and automated model generation. While quantitative benchmarking of evaluation metrics (such as accuracy, F-measure, precision, and recall) against these existing methods would be ideal, a direct numerical comparison is practically infeasible. As established in Section 2.3.1, prior works rely heavily on qualitative frameworks, proprietary datasets, or unavailable source code. Because the original samples and ground-truth Sequence Diagrams from these studies are inaccessible, simulating their exact evaluation settings or running their baseline tools in this research cannot be reliably performed. Consequently, this research utilizes independently validated oracle models and presents the qualitative benchmark in Table 10 as the most rigorous comparative analysis possible under current domain constraints.

6.4. Threats to Validity of Results

The primary threat to the validity of the evaluation results, particularly regarding the correctness of the models generated by the proposed method, stems from the reliance on 10 sets of FRs that were (1) grammatically well-formed, (2) relatively simple, (3) drawn from one application domain (web applications) and (4) explicit constraints were included (preconditions, sequencing, actor responsibilities, conditional logic) such as:

- Sequence of interactions: "Upon placing an order, the system must send an email notification to the user containing the order details and a tracking link." This specifies the sequence of actions.
- Actor responsibilities: "The user must provide all required fields (e.g., name, email) during registration before the system allows account creation." This specifies the preconditions for actions.
- Component behaviour: "The notification service must trigger a delivery confirmation to the user after the payment gateway confirms a successful transaction." This defines the role of specific components.
- Conditional logic: "If the user enters incorrect login credentials, the system must display an error message and prompt for re-entry." This ensures error-handling scenarios are incorporated in the sequence diagram.

These choices help the pipeline produce stable PlantUML statements but also limit generalizability of the findings. The results may not be directly applicable to situations where FRs are ambiguous, substantially more complex, or originate from other domains (e.g., embedded, real-time, safety-critical, or distributed systems).

Additionally, for each FR set, three (3) third-party professionals (from a pool of six) independently authored PlantUML statements and then participated in a reconciliation session to reach consensus on a single oracle model. While this process adds independence and reduces single-author bias, the small number of contributors per FR set remains a threat. Involving more professionals (or multiple independent panels) could yield a different oracle model, potentially affecting correctness scores. Several mitigation measures were applied to address these risks:

- recruit experienced practitioners in software engineering or design,
- conducted a rigorous reconciliation process with structured discussions to refine and consolidate the participants' submissions into the final oracle model.

In summary, the findings of this evaluation should be interpreted within the context of these limitations. The results primarily demonstrate the feasibility of the proposed method under controlled experimental conditions.

7.0 CONCLUSION AND FUTURE WORKS

This research proposed a fully automated method (without human intervention) for generating UML sequence diagrams (which is a type of behavioural model) by analysing FR sets written in grammatically correct English. The proposed method leverages OpenAI's GPT-4 model via the API, using fine-tuned prompts to transform requirements into PlantUML statements, applies deterministic parameters to ensure consistent output, and validates and renders the PlantUML statements into UML sequence diagrams.

A proof-of-concept software prototype was developed using Python. The prototype integrates with GPT-4 via the OpenAI API, performs validation of the generated PlantUML statements, and renders the corresponding UML sequence diagrams. This demonstrates the technical feasibility and automation capability of the proposed method.

Table 10: Comparison Between the Proposed Method and Existing Studies

Study	Approach	Input	Type of model generated	Level of automation	Evaluation Rigor
Proposed Method	Used LLM, specifically GPT-4 model via API, which allows tuning of deterministic parameters to ensure consistent output.	FR sets	UML sequence diagram	Fully automated without human intervention.	Correctness of the generated sequence diagrams were checked against oracle models (sample answers) constructed by third-party SE professionals by calculating cosine similarity scores.
<i>Ferrari et al.</i> [11]	One-shot prompt with ChatGPT (GPT-3.5).	NL requirements document in different formats, including “shall”-style, user stories and use case specifications.	UML sequence diagram	Fully automated without human intervention.	Qualitative analysis by researchers based on completeness, correctness, UML compliance, and understandability
<i>Ahmad et al.</i> [10]	LLM-powered chatbot, specifically ChatGPT.	Architecture Story which is a textual narration of the functionality, desired quality and constraints of the software.	UML class and component diagrams	Iterative human interaction with ChatGPT.	Evaluated via Software Architecture Analysis Method (SAAM), a scenario-based method, to evaluate if the architecture met quality goals, but no quantitative metrics were reported.
<i>Cámara et al.</i> [9]	LLM-powered chatbot, specifically ChatGPT.	Prompt with context on the system, along with technical constraints.	UML class diagrams	Iterative human interaction with ChatGPT.	Evaluated against reference models created by the researchers themselves and not by independent third-party experts.

Table 10: Continued.

Study	Approach	Input	Type of model generated	Level of automation	Evaluation Rigor
<i>Kim</i> [12]	LLM-powered chatbot, specifically ChatGPT.	Use cases and requirements for a tour reservation system (TORS)	UML class and sequence diagram	Semi-automated (ChatGPT generate the NL description of the diagram; researchers manual converted into UML class and sequence diagram)	Evaluated based on alignment with use cases and traceability across generated UML class and sequence diagrams.
<i>Gulia and Choudhury</i> [13]	Rule-based NLP approach.	FR sets	UML sequence and activity diagrams	Fully automated without human intervention.	Evaluation details not reported; results based on a small-scale dataset, limiting generalizability
<i>Jahan et al.</i> [14]	Rule-based NLP approach.	FR sets	UML sequence diagrams	Fully automated without human intervention.	Evaluated against reference diagrams created by the researchers themselves and not by independent third-party experts.

An evaluation study involving six (6) domain experts demonstrated the effectiveness of the proposed method. The correctness of the UML sequence diagrams generated by the prototype, measured using TF-IDF and cosine similarity against expert-validated oracle models, achieved an average similarity score of 86.14%, indicating strong alignment between the generated and reference models.

The proposed method, while demonstrating the feasibility of fully automating the generation of UML sequence diagrams from FR, has several inherent limitations. First, it relies on FRs written in grammatically correct English, as ambiguity or poor sentence structure may cause the GPT-4 model to misinterpret entities or interactions, leading to inaccurate PlantUML statements. Second, the method focuses exclusively on FRs and does not incorporate NFRs or architectural constraints. As a result, the generated diagrams capture behavioural aspects of the system but not higher-level architectural qualities or design trade-offs.

In conclusion, the result of this research demonstrates the potential of LLM (in particular GPT-4) in automatically generating UML sequence diagrams from FR sets written in English. This could streamline early-stage design in the software development lifecycle.

Future research could focus on addressing these limitations to enhance the proposed method. First, the method's capability to generate more context-aware UML sequence diagrams could be strengthened by explicitly incorporating NFRs and architectural constraints into the FRs. This would enable the model to better capture design concerns thereby improving both correctness and realism of the generated models. Second, future studies could involve a larger and more diverse collection of FRs including more complex and cross-domain scenarios beyond web applications. This would allow a broader evaluation of the method's robustness and adaptability across different system types and levels of complexity.

REFERENCES

- [1] N. B. Ruparelia, "Software development lifecycle models," *ACM SIGSOFT Software Engineering Notes*, Vol. 35, No. 3, pp. 8–13, 2010.
- [2] IEEE Computer Society, *Guide to the Software Engineering Body of Knowledge (SWEBOK): Version 4.0*, 2025.
- [3] I. Sommerville, *Software Engineering*, Global ed., Harlow, Pearson Education, 2016.
- [4] O. S. Dawood and A.-E.-K. Sahraoui, "From Requirements Engineering to UML using Natural Language Processing – Survey Study," *European Journal of Engineering Research and Science*, Vol. 2, No. 1, p. 44, 2017.

- [5] M. Kassab, C. Neill, and P. Laplante, "State of practice in requirements engineering: contemporary data," *Innovations in Systems and Software Engineering*, Vol. 10, No. 4, pp. 235–241, 2014.
- [6] E. A. Abdelnabi, A. M. Maatuk, and M. Hagal, "Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques," *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering*, IEEE, May 2021, pp. 288–293.
- [7] S. Ahmed, A. Ahmed, and N. U. Eisty, "Automatic Transformation of Natural to Unified Modeling Language: A Systematic Review," *2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA)*, IEEE, May 2022, pp. 112–119.
- [8] Object Management Group (OMG), *Unified Modeling Language Specification Version 2.5.1*, 2017.
- [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.
- [10] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M. S. Aktar, and T. Mikkonen, "Towards Human-Bot Collaborative Software Architecting with ChatGPT," *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, 2023, pp. 279–285.
- [11] A. Ferrari, S. Abualhaija, and C. Arora, "Model Generation with LLMs: From Requirements to UML Sequence Diagrams," *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*, IEEE, Jun. 2024, pp. 291–300.
- [12] D.-K. Kim, "Using ChatGPT to Develop Software Systems: Alert to Software Engineers?" *Journal of Software Engineering Research and Development*, pp. 1–16, 2023.
- [13] S. Gulia and T. Choudhury, "An efficient automated design to generate UML diagram from Natural Language Specifications," in *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, IEEE, Jan. 2016, pp. 641–648.
- [14] M. Jahan, Z. S. H. Abad, and B. Far, "Generating Sequence Diagram from Natural Language Requirements," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, IEEE, Sep. 2021, pp. 39–48.
- [15] L. Zhao et al., "Natural Language Processing for Requirements Engineering," *ACM Computing Surveys*, Vol. 54, No. 3, pp. 1–41, 2021.
- [16] A. Alzayed and A. Al-Hunaiyyan, "A Bird's Eye View of Natural Language Processing and Requirements Engineering," *International Journal of Advanced Computer Science and Applications*, Vol. 12, No. 5, 2021.
- [17] E. A. Abdelnabi, A. M. Maatuk, T. M. Abdelaziz, and S. M. Elakeili, "Generating UML Class Diagram using NLP Techniques and Heuristic Rules," in *2020 20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, IEEE, Dec. 2020, pp. 277–282.
- [18] N. Bashir, M. Bilal, M. Liaqat, M. Marjani, N. Malik, and M. Ali, "Modeling Class Diagram using NLP in Object-Oriented Designing," in *2021 National Computing Colleges Conference (NCCC)*, IEEE, Mar. 2021, pp. 1–6.
- [19] A. M. Maatuk and E. A. Abdelnabi, "Generating UML Use Case and Activity Diagrams Using NLP Techniques and Heuristics Rules," in *International Conference on Data Science, E-learning and Information Systems 2021*, New York, NY, USA: ACM, Apr. 2021, pp. 271–277.
- [20] Visual Paradigm, "Visual Paradigm," *Visual Paradigm*. Accessed: Sep. 23, 2026. [Online]. Available: <https://www.visual-paradigm.com>
- [21] Lucid Software Inc., "Lucidchart," *Lucidchart*. Accessed: Sep. 23, 2026. [Online]. Available: <https://www.lucidchart.com>
- [22] PlantUML, "Sequence diagram syntax and features," *PlantUML*. Accessed: Sep. 23, 2026. [Online]. Available: <https://plantuml.com/sequence-diagram>
- [23] Mermaid, "Mermaid," *Diagramming and charting tool*. Accessed: Sep. 23, 2026. [Online]. Available: <https://mermaid.js.org>
- [24] UMLet, "UMLet," *Free UML tools for fast UML diagrams*. Accessed: Sep. 23, 2026. [Online]. Available: <https://www.umlet.com>
- [25] A. Kurotych, L. Bulatetska, and O. Onyshchuk, "Reconstructing Entity Relationships in Database Schemas with Plantuml and Llms," *Cybersecurity: Education, Science, Technique*, Vol. 1, No. 29, pp. 152–160, Sep. 2025.
- [26] OpenAI, "GPT-4 Technical Report," *arXiv*, 2023.
- [27] H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," *arXiv.org*, Feb. 2023.
- [28] Google AI, "Google Gemini," *Gemini*. Accessed: Sep. 23, 2026. [Online]. Available: <https://gemini.google.com/app>
- [29] X. Wei et al., "ChatIE: Zero-Shot Information Extraction via Chatting with ChatGPT," *arXiv.org*, Feb. 2023.
- [30] J. Wang et al., "Is ChatGPT a Good NLG Evaluator? A Preliminary Study," *arXiv.org*, Mar. 2023.

- [31] 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*, New York, NY, USA: ACM, May 2022, pp. 1–5.
- [32] I. Ozkaya, "The Next Frontier in Software Development: AI-Augmented Software Development Processes," *IEEE Software*, Vol. 40, No. 4, pp. 4–9, Jul. 2023.
- [33] D. D. Prasetya, A. Prasetya Wibawa, and T. Hirashima, "The performance of text similarity algorithms," *International Journal of Advances in Intelligent Informatics*, Vol. 4, No. 1, p. 63, Mar. 2018.
- [34] W. H. Gomaa and A. A. Fahmy, "A Survey of Text Similarity Approaches," *International Journal of Computer Applications*, Vol. 68, No. 13, pp. 13–18, Apr. 2013.
- [35] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, Morristown, NJ, USA: Association for Computational Linguistics, 2001, p. 311.
- [36] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," *Text Summarization Branches Out*, Association for Computational Linguistics, pp. 74–81, 2004.
- [37] S. Banerjee and A. Lavie, "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments," *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pp. 65–72, 2005.
- [38] A. Huang, "Similarity measures for text document clustering," *Proceedings of the Sixth New Zealand Computer Science Research Student Conference*, pp. 49–56, 2008.
- [39] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan, "Cosine similarity to determine similarity measure: Study case in online essay assessment," in *2016 4th International Conference on Cyber and IT Service Management*, IEEE, Apr. 2016, pp. 1–6.
- [40] M. A. K. Raiaan et al., "A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges," *IEEE Access*, Vol. 12, pp. 26839–26874, 2024.
- [41] D. Napoleone, S. Marks, and E. Frederich, "PlantUML 0.3.0," *PyPI*. Accessed: Sep. 23, 2026. [Online]. Available: <https://pypi.org/project/plantuml>
- [42] OpenAI, "Chat API Reference" *OpenAI API Reference*. Accessed: Sep. 23, 2026. [Online]. Available: <https://platform.openai.com/docs/api-reference/chat>
- [43] S. Anadkat, "How to make your completions outputs consistent with the new seed parameter," *OpenAI Developers*. Accessed: Sep. 23, 2026. [Online]. Available: https://cookbook.openai.com/examples/reproducible_outputs_with_the_seed_parameter
- [44] S. Ekin, "Prompt Engineering For ChatGPT: A Quick Guide To Techniques, Tips, And Best Practices," Institute of Electrical and Electronics Engineers (IEEE), Apr. 2023.
- [45] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Boston, Addison-Wesley, 2012.
- [46] F. Rahutomo, T. Kitasuka, and M. Aritsugi, "Semantic Cosine Similarity," *The 7th International Student Conference on Advanced Science and Technology*, 2012.
- [47] P. Bafna, D. Pramod, and A. Vaidya, "Document clustering: TF-IDF approach," in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, IEEE, Mar. 2016, pp. 61–66.
- [48] J. Han, J. Pei, and H. Tong, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2022, pp. 39–82.
- [49] Scikit-learn Developers, "TfidfVectorizer," *scikit-learn*. Accessed: Sep. 23, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- [50] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

BIOGRAPHY

Ji Yong Loo is a solution architect specializing in the design and implementation of software systems across diverse technology domains. He received his B.Comp.Sc. (Hons.) degree from Universiti Sains Malaysia. His current research interests include cloud computing, artificial intelligence, and software engineering.

Moon Ting Su is a Senior Lecturer in the Department of Software Engineering, Universiti Malaya. She received her Ph.D. (Computer Science) from the University of Auckland, New Zealand. She is a member of the ACM and IEEE Computer Society, and a senior member of the IEEE. Her expertise lies in software architecture and design, design patterns, object-oriented programming, end-user programming, and recommender systems. She was the treasurer for the iiWAS 2005, MoMM 2005, and Informatics 2007 international conferences. She served on the PyCon Malaysia 2015 organising committee. Before joining the education sector, she was an analyst programmer.