POLITECNICO
DI TORINO

POLITECNICO DI TORINO

Master Degree course in Engineering and Management

Master Degree Thesis

Adaptive Validation of Use Case Diagram Exercises: Integrating Large Language Models for Generating and Assessing Equivalent Solutions.

Supervisors

Prof. Riccardo Coppola

Doct. Giacomo Garaccione

Candidate

Ayoub El Fateh

Academic Year 2024-2025

Acknowledgements

Abstract

The increasing use of large language models (LLMs) in software engineering has opened up new avenues for automating complex modeling tasks, particularly in transforming natural language specifications into formal representations. This thesis studies the adaptive control of UML Use Case Diagram (UCD) exercises using LLMs, such as ChatGPT and DeepSeek, to generate analogous solutions and evaluate student tasks. A set of 23 manually selected exercises, each with human-written reference solutions in visual and JSON formats, is the basis for the empirical investigation. Using prompt engineering methodologies (role-based and iterative prompts), LLMs were prompted to generate UCDs, which were evaluated using a multifaceted framework concerning syntactic correctness, semantic equivalence, and pragmatic usefulness. The evaluation employs systematic measures to compare the results of LLM models with validated references and analyze their completeness and redundancy. The results show that, although LLMs possess adequate syntactic capability, their semantic flexibility is limited, especially in learning environments where variable but correct solutions are expected. This study contributes a reproducible workflow and a new testing framework that highlights the advantages and disadvantages of LLMs in supporting modeling education, offering pathways for integrating adaptive feedback systems and enhancing automated assessment tools.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The landscape of software engineering (SE) is constantly evolving, and the design phase is one of the most crucial in the development process because it is where the foundations of the software system are conceptualized, designed, and documented, laying the foundation for the subsequent development process. Hence, notations like the Unified Modeling Language (UML) are essential in translating ideas and requirements into formalized structures in software engineering. Use Case Diagrams are essential UML constructs for representing functional requirements from the perspective of system users. These diagrams facilitate communication between developers and stakeholders and are commonly employed during the early stages of requirements input and system design. This diagram is often the first to be introduced in academia and industry, as it offers a straightforward way to describe the system's functionality from the user's perspective. However, realizing accurate and comprehensive use case diagrams is challenging, especially during the requirement-gathering stages, due to the complexity of translating abstract requirements into detailed visual models, demanding a deep understanding of the system's domain and a good knowledge of UML's syntax and semantics. Students often struggle to translate textual requirements into correct visual models, and teachers are faced with a time-consuming task when evaluating proposals. The rapid spread of Large Language Models (LLMs) in various fields in recent years suggests a possible solution to this challenge. Due to their deep learning algorithms and large training data sets, LLMs have demonstrated competence in understanding and generating human-like text, opening up the possibility of automation in tasks that traditionally required human skills. In recent years, Large Language Models such as ChatGPT and DeepSeek have opened new opportunities in this area [1]. Within software engineering, the potential of LLMs to automate the generation of UML use case diagrams is particularly interesting, as it could improve efficiency and reduce errors in the design process, allowing human designers to prioritize more strategic

aspects of system development. The LLMs' performance has attracted increasing attention in research and education for software engineering tasks, such as code generation, documentation, and even UML diagramming [1, 2]. Indeed, these models can generate diagrams based on natural language input and even offer solution variants.

However, most of this research focuses on the structural aspect of modeling.

At the same time, little attention has been paid to use case diagrams, which require a different understanding, closer to how humans reason about interactions, actors, and goals. Thus, this thesis addresses that gap. The central question is whether it is possible to fill this gap by evaluating the feasibility and effectiveness of using LLM models to generate UML Use case diagrams. The goal is to assess whether LLM models can support or potentially improve traditional UML diagram generation practices.

The methodology adopted in this work is structured to address three fundamental research questions related to the generation and evaluation of UML use case diagrams using large language models. The first part is the creation of a curated repository of 23 exercises, each including a textual software requirement and a reference solution in visual and JSON (JavaScript Object Notation) format. UCDs are then generated for each exercise through two channels: one by a human modeler under controlled academic conditions and the other using LLMs, specifically ChatGPT and DeepSeek, guided by a prompt engineering strategy. The second part evaluates the results, introducing a framework combining quantitative and qualitative metrics. From a quantitative perspective, the diagrams are evaluated using completeness and redundancy rates, measuring the coverage of expected and unnecessary elements. From a qualitative perspective, the diagrams are analyzed in three dimensions: syntactic, semantic, and pragmatic.

By demonstrating the capabilities and limitations of large language models in this context, this study provides valuable insights into the potential integration of artificial intelligence into software engineering practices. The findings could pave the way for more efficient, accurate, and automated design processes, transforming human designers' roles, aiming to reduce software development's overall efficiency. Moreover, identifying areas where LLMs still lag, particularly in understanding and replicating domain-specific knowledge, highlights critical avenues for future advances in AI, with the target of reducing the gap between AI-generated and human-generated diagrams.

This thesis is structured into several key sections: a comprehensive **Background** chapter lays the theoretical foundations and places the study in the context of existing research. The chapter on **Methodology** illustrates the overall approach adopted to evaluate the diagrams. The chapter on **Results** presents the findings of this evaluation and numerical outcomes aimed at identifying the elements that most affect the quality of the diagrams generated.

The work culminates in a **Conclusion** that summarizes the knowledge gained from the research and proposes recommendations for the future.

Chapter 2

Background

2.1 Modeling and Requirements in Engineering

2.1.1 Introduction to the significance of modelling in Software Engineering

The term ”model” is derived from the Latin word modulus, which means measure, rule, pattern, example to be followed. Models can be found in all areas and applications of software engineering. In the software development field, modeling refers to the process of creating abstract representations of a system. It helps with understanding, visualizing, and communicating the system's structure and behavior. Modeling can be considered a framework used to guide developers and stakeholders during the software development life cycle, bridging the gap between user requirements and technical implementations. Models play several critical roles within software engineering:

1. They facilitate communication between technical and non-technical stakeholders, enhancing the collaborative development process.

2. Modeling provides an efficient method to represent, analyze, and validate different aspects of a system before the actual development begins. This will significantly reduce risks and ambiguities associated with system requirements and specifications.

Historically, software engineering has evolved from simply imitating physical and manual processes to creating innovative solutions that go beyond traditional paradigms. However, this shift to innovative development is often limited by the inherent limits of human imagination and creativity, underscoring the importance of leveraging structured and formal modeling techniques [3]. The latest progress has introduced sophisticated modeling

languages and tools, such as the Unified Modeling Language (UML), Systems Modeling Language (SysML), and Business Process Model and Notation (BPMN). These tools help to effectively capture and communicate software systems' structural and behavioral complexities. The UML, in particular, has become an industry standard because of its versatility and comprehensiveness, able to encompass both structural (e.g., class diagrams) and behavioral (e.g., use case diagrams) dimensions of software systems [4].

2.1.2 Design for Software Engineering

Effective software design involves making informed decisions about the system structure, data models, interfaces, and components. Design methodologies in software engineering typically involve iterative processes that enable the continuous refinement and enhancement of the system models. This iterative nature ensures ongoing alignment between stakeholder requirements and system capabilities. Techniques such as model-driven architecture (MDA) further enhance the design process by systematically using models at different abstraction levels, promoting automation, consistency, and traceability throughout the software development life cycle [3]. When software development follows the traditional waterfall approach, each generated document becomes prescriptive for the subsequent one, forming a logical chain from specification to architectural design, detailed design, source code, and program execution. Notably, the requirements specification document holds a dual role: it simultaneously describes user needs and prescribes the product to be developed. This duality positions the specification as the most crucial component in the software development process, as is shown in Figure 2.1 [3].

In addition, gamification and other innovative educational methodologies have been increasingly adopted to enhance understanding and skill development in software modeling among students and professionals. These approaches leverage interactive and engaging learning environments to significantly improve productivity and the quality of modeling practices significantly, demonstrating the practical benefits of gamified learning experiences in software engineering education. The figure 2.2 shows the main page of a UML tool, developed to display the list of available exercises and, on the left-side menu, information about the current user, such as their avatar, level, identifier, and experience points.

The icons below the avatar can be used to access the avatar customization page, the leaderboard section, and a Sandbox where users can freely create class diagrams without the restrictions of an exercise [3, 5].

Software design acts as a bridge between a system's initial requirements and its final

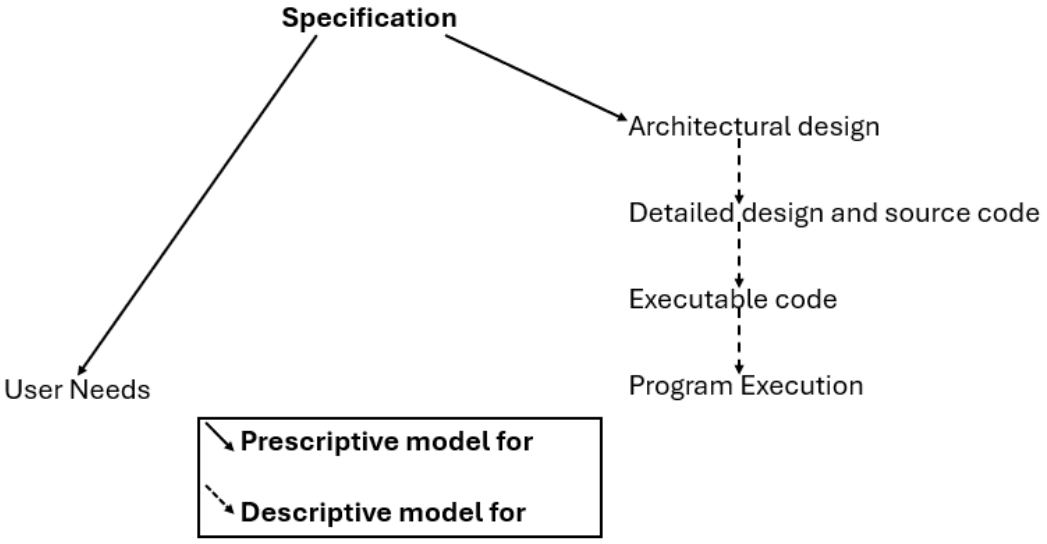

Figure 2.1. Models in Software Engineering: The (simplified) document chain [3]

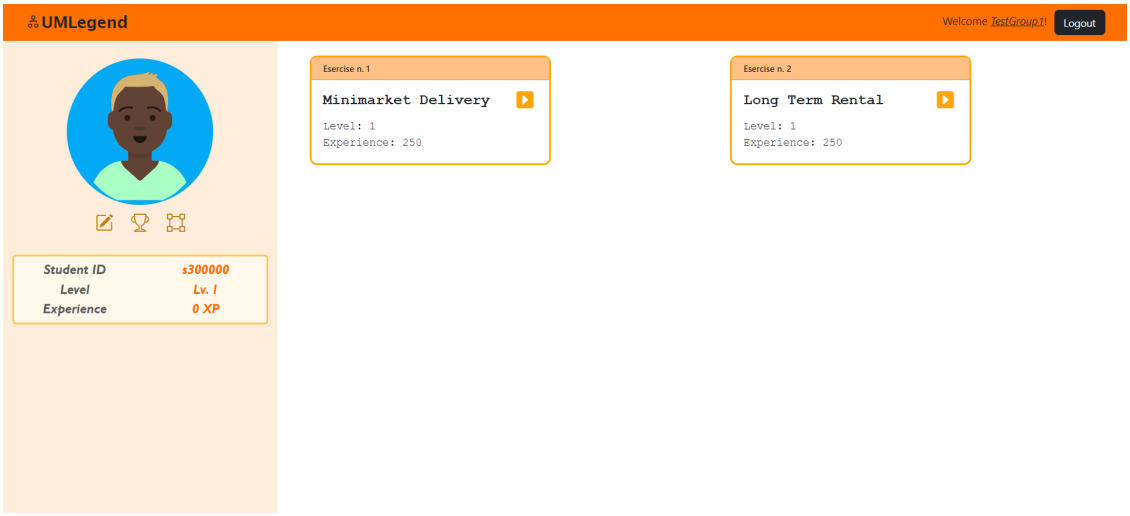

Figure 2.2. Homepage of the UMLegend tool, with the user information on the left and exercise list on the right [5]

implementation. Decisions made during this process significantly impact the maintainability, scalability, and quality of the final software product. Thus, a well-designed software system ensures efficient resource utilization, ease of integration, and adaptability to future changes.

2.1.3 Role of Requirements in Modeling

Requirements are fundamental to successful software modeling and serve as the critical foundation for accurate system design and implementation. They bridge the gap between user expectations and technical solutions, ensuring that the final product aligns with stakeholders' needs and business objectives. Practical requirements engineering involves a structured process of eliciting, analyzing, specifying, and validating both functional and nonfunctional requirements, which helps to minimize ambiguity and reduce costly rework during later development stages [6]. Requirements gathering, the first step in this process, involves identifying, collecting, and documenting stakeholder needs. It forms the basis for the entire software development process, ensuring alignment with business goals and user expectations. Accurate requirements gathering reduces the risk of costly design errors and supports effective risk mitigation throughout the software life cycle [7]. However, this phase can be particularly challenging due to the potential for vague or ambiguous inputs, conflicting stakeholder priorities, and evolving project scopes.

To address these challenges, a range of techniques is employed to capture and clarify requirements:

- Interviews: Direct conversations with stakeholders to understand their needs and expectations.
- Surveys and Questionnaires: Written forms distributed to gather structured feedback on specific topics.
- Prototyping: Developing preliminary software versions to gather user feedback and refine requirements.
- Observation: Studying users' natural environments to understand their tasks and challenges.
- Workshops: Collaborative sessions where stakeholders and development teams work together to define requirements.
- Use Cases: Detailed descriptions of user interactions with the system, capturing functional requirements through realistic scenarios.

Each of these techniques has advantages and limitations, and the choice often depends on the project's context, stakeholder availability, and the complexity of the developed system. For instance, interviews and focus groups can provide deep insights but may require significant time and coordination. At the same time, surveys offer broader input

but lack the depth of one-on-one interactions. Several common challenges can arise during requirements gathering, including:

- Vague or Ambiguous Requirements: Stakeholders may not articulate their needs clearly.
- Changing Requirements: Project scopes can evolve as new insights emerge, requiring flexibility in requirements management.
- Conflicting Requirements: Different stakeholders may have competing priorities or expectations.
- Unstated Requirements: Stakeholders may overlook critical needs that only become apparent as the project progresses.

To address these issues, effective requirements management practices include:

- Regular Communication: Maintaining continuous dialogue with stakeholders to clarify expectations.
- Prioritization: Focusing on the most critical requirements to ensure alignment with business goals.
- Comprehensive Documentation: Recording requirements and changes to ensure consistent understanding across the project team.

Effective requirements engineering is essential for successful software modeling, directly impacting the quality, reliability, and long-term sustainability of software systems [6, 7].

2.2 Use Case Modelling and UML

Use case diagrams are a fundamental component of the Unified Modeling Language (UML), which is widely used in software engineering to capture the functional requirements of a system. They visually represent the interactions between different actors (users or external systems) and the system itself, clearly defining what the system needs to do from the user's perspective. This approach aligns strongly with the principles of requirements engineering, which emphasizes the accurate capture of stakeholder needs to ensure that the final software product aligns with business objectives and user expectations [6]. The basic elements of a use case diagram include actors, use cases, and their relationships. Actors represent external entities interacting with the system, such as users, devices, or other systems. Use cases, typically represented as ovals, describe the system's actions or

services to these actors. The relationships, including associations, extensions, inclusions, and generalizations, define how these components interact [8]. Use case diagrams serve several critical purposes in software development:

- Defining system boundaries: They help define the scope of the system by explicitly identifying the functionality it provides and the external entities with which it interacts.
- Improving communication: Use case diagrams bridge technical teams and non-technical stakeholders, facilitating more transparent communication and understanding of system requirements.
- Support system design and documentation: They form the basis for more detailed design patterns and serve as a reference throughout the development lifecycle.
- Identification of functional requirements: They capture the system's intended functionality, providing a structured way to document what the system needs to do from the user's perspective.

In addition, use case diagrams are particularly effective in the early stages of software development, when high-level requirements are being defined. They provide a direct way to capture the "big picture" of the system without diving into low-level technical details, making them accessible to technical and non-technical stakeholders [9]. Figure 2.3 shows an example of a use case diagram.

2.2.1 Actors, Use Cases, Include, and Extend

In the context of use case diagrams, several key components define the structure and interactions within the system.

These components include actors, use cases, and the relationships that connect them: inclusion, extension, and generalization links.

Together, these elements form the backbone of a well-defined functional model, which captures the primary functionality a system needs to support from the user's perspective.

1. **Actors**: are external entities interacting with the system. They can represent human users, external systems, or devices that provide input to the system or receive output from it. In use case diagrams, actors are typically represented as stick figures to emphasize their external relationship with the system. They are not part of the system but are crucial in defining its boundaries and intended interactions.

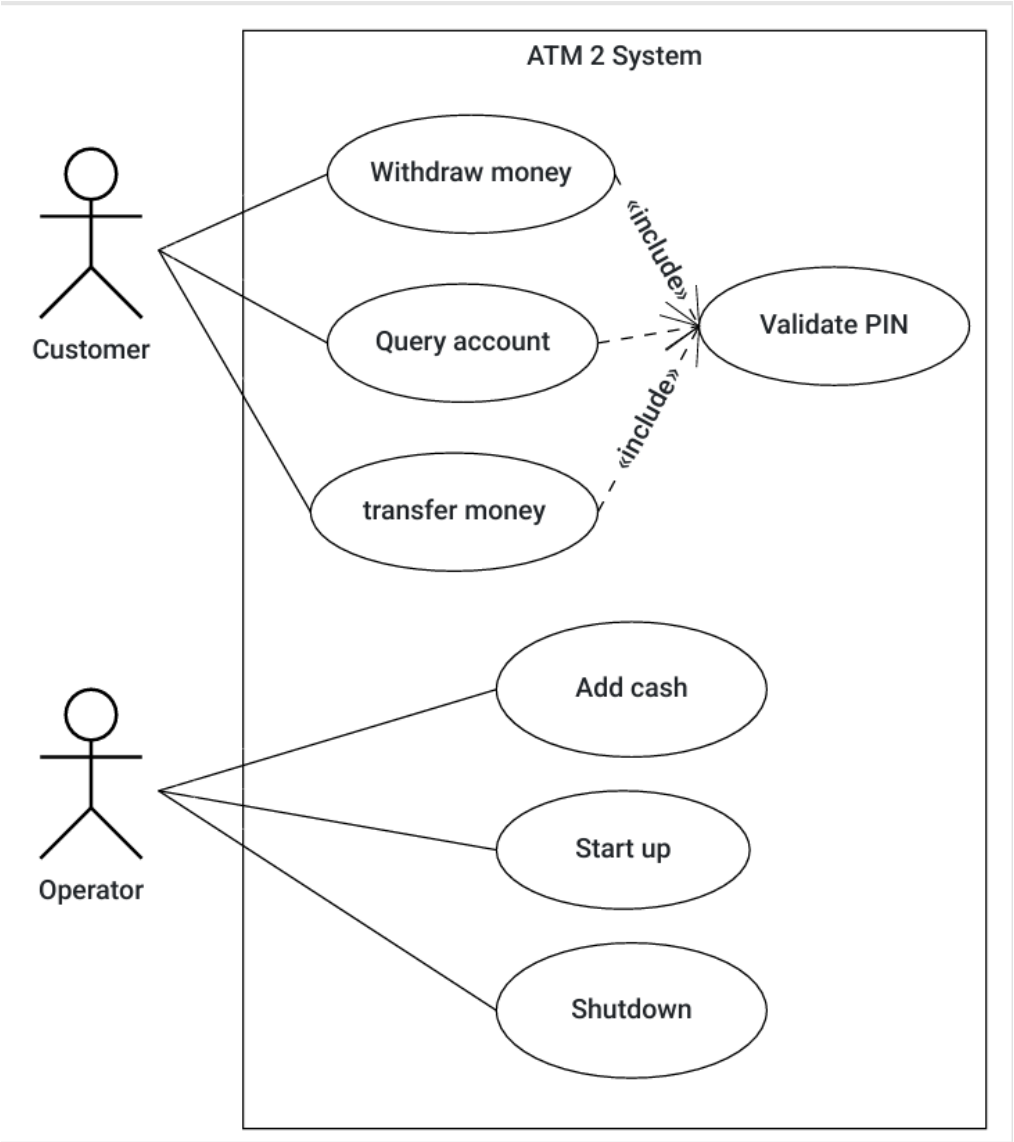

Figure 2.3. Example of a Use Case Diagram for an ATM system

2. **Use Cases**: describe the specific functionality provided by the system to achieve a particular goal for an actor. They are represented as ovals and capture the main interactions between the actors and the system, defining the tasks the system must perform to meet the user's needs. Each use case should focus on delivering a measurable outcome to the actor, reflecting the system's primary purpose. The include, extend, and generalization relationships are used to model more complex interactions [8]:

- **Include**: This relation captures a mandatory, reusable behavior shared by multiple use cases. For example, a use case "Login" could be included by both use cases "Buy Item" and "Update Profile", reflecting the common need for user authentication.
- **Extend**: Represents an optional or conditional behavior that adds functionality to a basic use case under certain conditions. For example, a "Checkout" use case might extend to include "Apply discount" if a promotional code is provided.
- **Generalization** relations capture hierarchical connections between actors or use cases, representing specialized forms of more general entities. For example, a "Registered User" might be a specialized actor within the broader "User" category, inheriting its general behaviors but adding unique capabilities.

These relationships are critical to accurately capture complex system behaviors, support scalability, and promote reuse within the model. They also play a key role in defining the boundaries and scope of the system, ensuring that all essential interactions are represented and understood by both developers and stakeholders [10].

2.2.2 Writing Good Requirements for Use Cases

Writing practical requirements for use cases is a crucial aspect of software modeling, as it directly affects the clarity and completeness of the resulting diagrams. Hence, well-written requirements serve as the basis for accurately representing the system, reducing ambiguity and minimizing the risk of costly redesigns later in the development process [11].

Effective requirements for use cases should be:

- Clear and concise: they should be simple and free of unnecessary complexity. They must clearly describe the intended behavior without ambiguity, ensuring all stakeholders share a common understanding.
- Complete and comprehensive: they should cover all necessary functionality without omitting critical aspects of the system. This includes both regular and alternate flows to capture all possible user interactions.
- Consistent and unambiguous: they must be free of contradictions and avoid using terms that different stakeholders could interpret differently.
- Testable and verifiable: each requirement must be measurable and testable, allowing adequate validation against the system's expected performance.

The User Story format is a widely adopted approach to writing effective requirements for use cases, commonly used in agile methodologies. User stories typically follow the format:

- "As a [type of user], I want [an action] so that [a benefit is obtained]."

For example:

1. As a customer, I want to be able to track the status of my order so that I can be informed of delivery updates.
2. As a system administrator, I want to manage user accounts to ensure security and access control to the system.

Figure 2.4 illustrates an example of a User Story format.

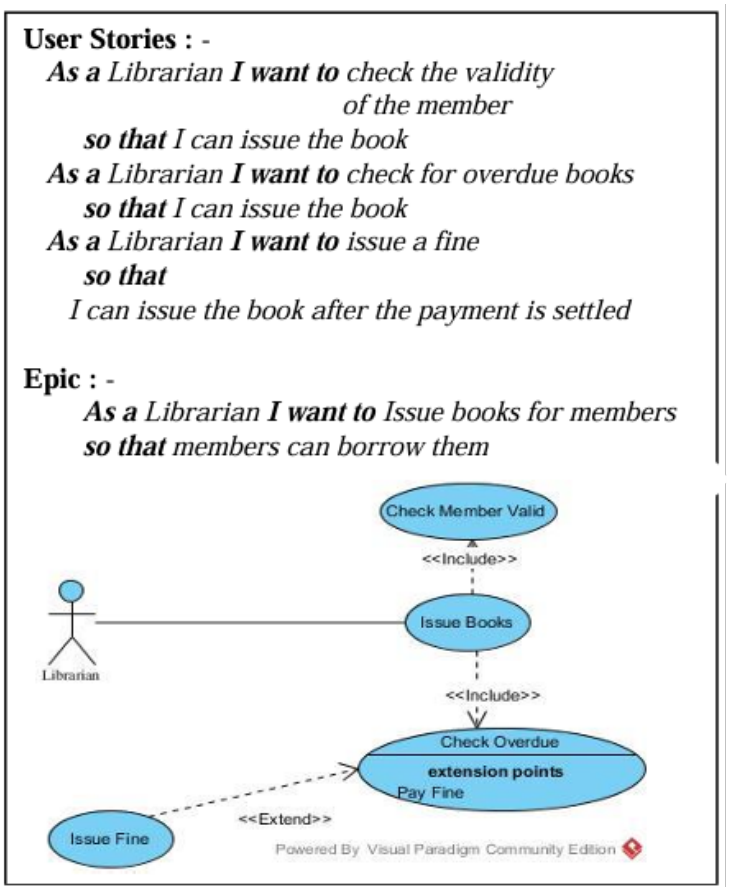

Figure 2.4. User story and Use case diagram for a library [8]

This format helps ensure that requirements capture the actor's goal and underlying motivations, providing a more user-centric approach to requirements specification.

In addition, several techniques can support the effective drafting of use case requirements, including [11]:

- Prototyping: Creating initial mockups or prototypes to clarify requirements through visual representation.
- Scenario analysis: Analysis of specific user interactions to capture different use case contexts.
- Stakeholder workshops: Directly involving end users and stakeholders to refine requirements.

2.2.3 Benefits for Developers and Stakeholders

Use case diagrams offer numerous advantages for developers and stakeholders, which are critical in ensuring successful software projects. These benefits are particularly evident in medium-sized projects, where effective communication and shared understanding are essential to manage complexity and align technical efforts with business goals [12]. Use case diagrams have several benefits for the developers and the stakeholders, the main ones are:

- Improved System Understanding: Use case diagrams provide a clear, high-level overview of system functionality, helping developers quickly understand the scope and structure of a project. This understanding supports better decision-making during the design and implementation phases.
- Enhanced Communication and Collaboration: Use case diagrams bridge the gap between technical and nontechnical stakeholders, providing a shared visual language that facilitates more effective collaboration and reduces the potential for misunderstandings.
- Better Requirements Validation: stakeholders can more easily validate and refine requirements when presented visually, ensuring that the final product aligns with business goals and user needs.
- Early Error Detection: Use case diagrams allow stakeholders to identify gaps and inconsistencies in requirements early in development, reducing the risk of costly changes at later stages.

2.2.4 Evolution of UML Tools and Use Case Diagrams

The evolution of UML tools and use case diagrams has been closely linked to the changing needs of software engineering practices over the past few decades. Initially introduced in the mid-1990s by the Object Management Group (OMG), UML was developed as a standardized way to model and visualize software systems. This approach unified various modeling methodologies into a single framework, addressing the need for a consistent and comprehensive way to describe complex software systems' structure and behavior. The first official version of UML, UML 1.0, was released in 1997, providing a foundational set of diagrams, including class, sequence, use case, activity, and state diagrams. This initial release focused on capturing the essential structural and behavioral elements required for effective software modeling. As software projects grew in size and complexity, the demand for more sophisticated UML tools increased. Today, modern UML tools like Enterprise Architect, Visual Paradigm, Lucidchart, and PlantUML offer robust features, including automatic code generation, reverse engineering, real-time collaboration, and integration with popular development environments.

These tools support traditional waterfall methodologies and agile practices, making UML a versatile choice for a wide range of projects [12].

In recent years, UML tools have evolved to support agile and DevOps methodologies, emphasizing rapid iteration and continuous integration. This shift has led to the development of more lightweight, collaborative modeling tools. These tools enable teams to keep their design documentation in sync with rapidly changing codebases, reducing the overhead of maintaining accurate models. Looking for future trends, integrating UML with advanced technologies like AI, machine learning, and natural language processing is expected to enhance the capabilities of these tools further. This includes the potential for automated model generation from textual requirements, real-time error detection, and intelligent model validation, promising even greater efficiency and accuracy in software design [12].

2.2.5 Challenges and Limitations of UML Use Case Diagrams

Despite their widespread use and advantages in software design, UML Use Case Diagrams also present several challenges and limitations [12]:

- Complexity: As the size and complexity of a system increase, use case diagrams can become cluttered and difficult to interpret, particularly for non-technical stakeholders. Overly complex diagrams can obscure the essential interactions between actors and the system, reducing their effectiveness as communication tools.

- Ambiguity: Although UML provides a standardized notation, ambiguity can still arise in how specific interactions and extensions are represented. This is particularly problematic when different teams interpret the same diagram differently, potentially leading to inconsistent understanding and implementation.
- Scalability Issues: Use case diagrams are generally effective for small to medium-sized systems, but can struggle to represent the full complexity of larger enterprise systems without becoming overwhelming and unwieldy.
- Integration Challenges: Despite the availability of numerous UML tools, integrating use case diagrams with other development tools, like issue trackers and continuous integration systems, remains challenging. This lack of seamless integration can hinder their practical use in modern DevOps pipelines.

2.3 Large Language Models (LLMs)

Large Language Models (LLMs) are a type of Artificial Intelligence (AI) that have emerged as powerful tools for a wide range of tasks, including Natural Language Processing (NLP), machine translation, vision applications, and question-answering. LLMs are advanced artificial intelligence (AI) systems explicitly developed to process and generate human-like text. Their prominence increased significantly following the public launch of OpenAI's ChatGPT in November 2022, demonstrating exceptional capabilities in answering questions, summarizing information, paraphrasing, and translating text with remarkable accuracy [13]. The interactive nature of LLMs such as ChatGPT has facilitated widespread adoption across various domains, notably in medicine and engineering. These models offer significant potential for democratizing knowledge and improving accessibility to information. However, several concerns accompany their usage. The risk of disseminating misinformation or unintentionally promoting scientific inaccuracies arises primarily due to the models' limited transparency and accountability. Moreover, since LLM outputs depend significantly on their training datasets, any inherent biases in these data can result in biased outputs (figure 2.5). The development trajectory of LLMs has been rapid and heavily influenced by advances in neural network models that imitate the human brain's interconnected structure. These neural networks process data through multiple layers, performing tasks such as pattern recognition, decision-making, and classification. Modern LLMs evolved from earlier NLP systems, such as the BERT model. In particular, after OpenAI introduced GPT-1 in 2018, other prominent models were released by tech giants like Google and Meta. The novel launch of ChatGPT, which uses Reinforcement Learning from Human Feedback (RLHF), represented a significant

advance, offering more accurate and human-like interactions than previous models [14].

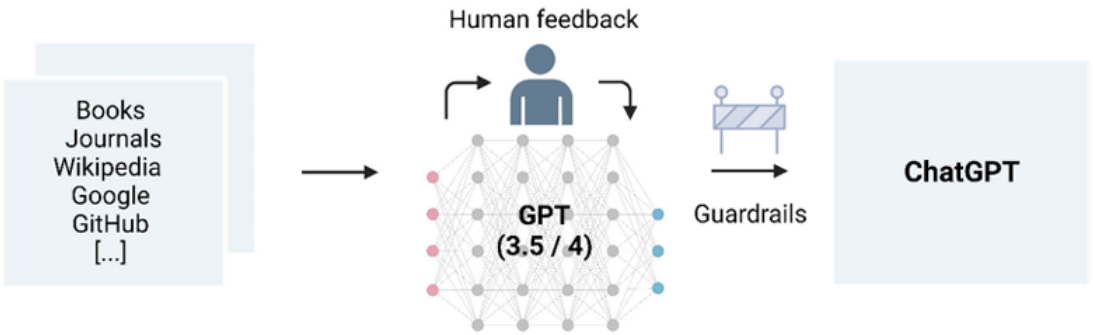

Figure 2.5. Design of the architecture behind ChatGPT, including training, iterations of reinforcement learning by human feedback, choice of available model, and implementation of guardrails to improve safety [14].

2.3.1 What Are LLMs

LLMs are advanced AI systems that understand, generate, and manipulate human language. They leverage deep learning techniques, primarily the Transformer architecture, which uses self-attention mechanisms to capture complex patterns and contexts within vast amounts of textual data. These models have revolutionized natural language processing (NLP), significantly improving tasks such as text generation, translation, summarization, and question answering. Historically, language models evolved from basic statistical models like n-grams and Hidden Markov Models (HMMs), which predicted text based on probability (figure 2.6).

However, these early models had limitations in understanding complex language contexts.

The introduction of neural networks, particularly recurrent neural networks (RNNs) and Long Short-Term Memory networks (LSTMs), marked a significant improvement. Yet, they still struggled with capturing long-range dependencies in language data [15]. A breakthrough came in 2017 with the Transformer model introduced by Vaswani et al., significantly enhancing the ability of LLMs to process and generate language by capturing extensive contextual relationships [16]. Models such as OpenAI's GPT series (Generative Pre-trained Transformers), Google's BERT, and Meta's LLaMA have since set new standards by effectively learning from massive datasets during pre-training and fine-tuning for specialized tasks.

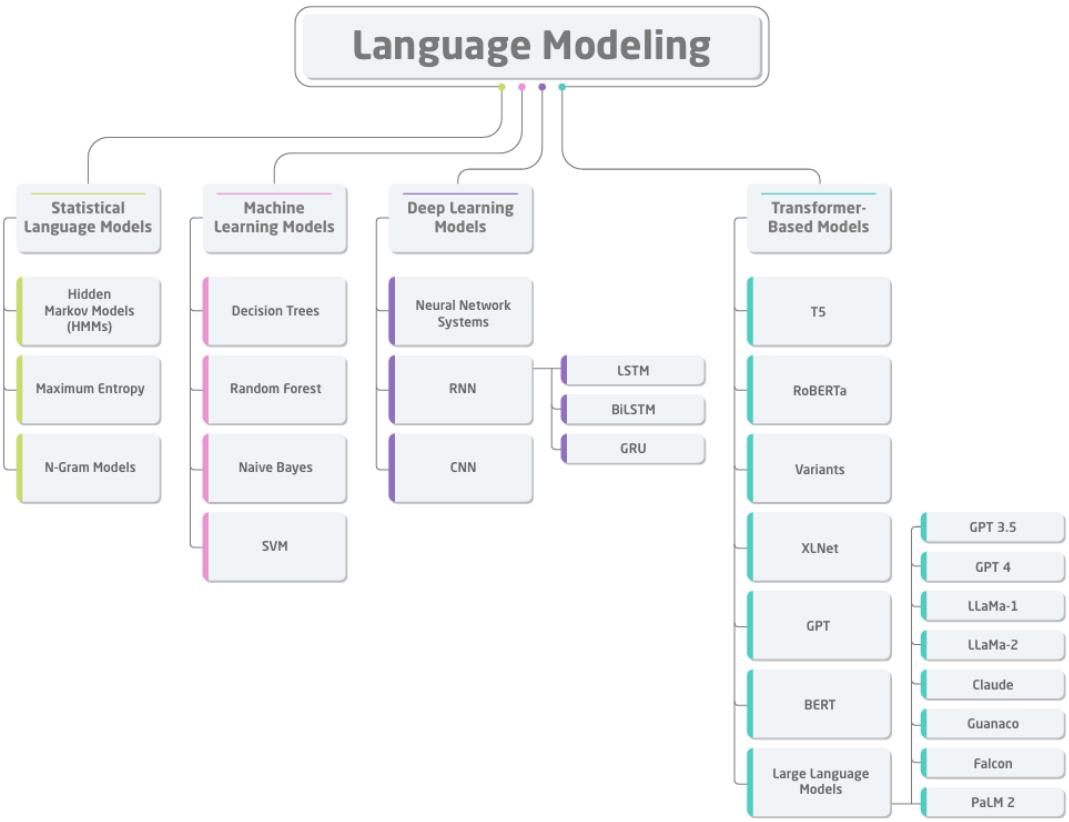

Figure 2.6. The division of LLMs is categorized into four major blocks: Statistical language models, Machine learning models, Deep learning models, and Transformer-based models [15]

2.3.2 Architecture and mechanism

Large Language Models (LLMs) are primarily based on the Transformer architecture, a neural network model that significantly enhanced the capability of natural language processing tasks through its innovative attention mechanisms.

Hence, transformers use self-attention to dynamically weigh the significance of each word relative to others, enabling the model to understand complex language contexts more effectively than previous architectures.

At the core of Transformer-based LLMs are the encoder and decoder modules cite veeramachaneni2025:

- The encoder processes the input text, generating context-aware representations, while the decoder generates output based on these representations. BERT is encoder-only and excels in bidirectional understanding tasks such as sentiment analysis and

question-answering, where context from past and future words is crucial.

- The decoder, GPT models like GPT-3 and GPT-4, are designed primarily for tasks like text generation and completion, where predicting subsequent words based on previous context is essential.

Instead, the key concepts underlying LLM functionality include [15]:

- Training Data: The effectiveness of an LLM is directly influenced by the quality and diversity of its training dataset. These datasets usually include sources such as books, articles, and websites, ensuring comprehensive linguistic coverage.
- Tokenization: This divides text into smaller, manageable units known as tokens, enabling models to handle extensive vocabulary and complex language structures efficiently.
- Embeddings: Tokens are represented numerically through embeddings, capturing their semantic meaning and context. Embeddings facilitate a deeper understanding of linguistic relationships, such as synonyms and antonyms.
- Attention Mechanism: This component allows models to dynamically focus on specific segments of the input text, significantly improving their ability to understand context and produce coherent and contextually accurate responses.

The training process of LLMs involves the following steps:

1. Pre-training: During this step, the model is exposed to vast amounts of text data, allowing it to learn language patterns broadly.
2. Fine-tuning: refines this general knowledge on specific tasks or domains using smaller, targeted datasets.
3. Instruction-tuning: enhances model accuracy by teaching it to follow specific instructions or prompts, thus improving its capability to perform effectively in real-world scenarios.

While transformers have significantly advanced NLP, they also require substantial computational resources due to their complexity and the vast datasets needed for training. These demands have raised concerns about efficiency, scalability, and the environmental impact of deploying large-scale language models. Addressing these challenges continues to be a critical focus in the ongoing development and application of LLMs [17].

2.3.3 Applications

Large Language Models (LLMs) have become essential tools across numerous industries due to their versatility in processing and generating human language. In healthcare, LLMs like BioBERT and ClinicalBERT have been adapted for disease diagnosis, patient monitoring, and medical record analysis, improving clinical decision-making and patient outcomes. In education, LLMs power personalized learning tools, automate grading, and support interactive tutoring systems, enhancing student engagement and academic performance. In business, they are used for customer service automation, market analysis, and financial forecasting, enabling companies to optimize operations and improve customer satisfaction. Additionally, LLMs support social media management, sentiment analysis, and content creation, providing valuable insights into consumer behavior and enhancing brand management. They are also employed in legal contexts for contract analysis and document review, streamlining legal processes and reducing manual workloads [18].

2.3.4 Progression of Large Language Models

The development of Large Language Models (LLMs) has seen rapid and transformative growth, evolving from early statistical approaches to advanced, multimodal systems. The progression of LLMs can be broadly categorized into four major stages (Figure 2.6):

1. Statistical models: the earliest language models, such as n-grams and Hidden Markov Models (HMMs), relied on simple probabilistic approaches, focusing primarily on the frequency of word sequences without deep contextual understanding. These early models were limited in capturing long-range dependencies in text and struggled with handling complex language structures.

2. Neural language models: The next significant leap came with the introduction of neural language models, including recurrent neural networks (RNNs) and long short-term memory (LSTM) networks. These models marked a substantial improvement over statistical methods by incorporating deep learning to capture more complex language patterns. However, due to vanishing gradient issues, RNNs and LSTMs faced challenges with long-term dependencies.

3. Transformer-based architectures models: this architecture revolutionized natural language processing by introducing self-attention mechanisms, which allowed models to consider the full context of a text input, significantly enhancing language understanding and generation. Transformers form the backbone of models like BERT, GPT, and T5, which have set new benchmarks in numerous NLP tasks.

4. Current large-scale foundation models: the current era of LLMs is characterized by foundation models like GPT-4, PaLM, LLaMA, and Gemini, which feature billions to trillions of parameters, extensive pre-training on diverse datasets, and sophisticated fine-tuning techniques. These models have demonstrated remarkable capabilities across various applications, including language translation, content creation, medical diagnostics, and complex reasoning [15].

2.3.5 Barriers and Ethical Reflections

The development and deployment of Large Language Models (LLMs) face several significant barriers, both technical and ethical. One of the primary challenges is bias in training data. LLMs learn language patterns from large amounts of publicly available text, which often contains societal biases related to race, gender, and culture. These biases can be inadvertently reinforced and amplified in the models' outputs, leading to unfair or discriminatory behavior in real-world applications. Another critical issue is the lack of interpretability and transparency. Despite their impressive capabilities, LLMs operate as complex black-box systems, making it difficult to trace how specific outputs are generated. This lack of transparency can undermine user trust and complicate efforts to ensure accountability, particularly in high-stakes domains like healthcare or legal services. LLMs also struggle with ethical consistency. They are trained on diverse datasets that reflect a wide range of moral perspectives, which can lead to conflicting outputs when faced with ethically ambiguous situations [19]. This inconsistency poses significant challenges for integrating LLMs into applications requiring consistent moral reasoning, such as autonomous decision-making or digital assistants. Furthermore, LLMs are inherently conservative in their learning, often replicating the biases and norms present in their training data without the ability to adapt to changing social values over time. This limitation reinforces outdated stereotypes and standards, making it difficult for these systems to remain ethically relevant in evolving cultural contexts. Finally, there are broader social implications to consider. The widespread deployment of LLMs could potentially centralize power among the organizations that develop and control these models, raising concerns about information monopolies and the erosion of human autonomy in decision-making [20].

2.4 Prompt Engineering Strategies

Prompt engineering is critical to effectively leveraging Large Language Models (LLMs) for software engineering tasks. Unlike traditional machine learning approaches that require extensive fine-tuning, prompt engineering focuses on carefully crafting the input prompts provided to LLMs, allowing them to generate contextually accurate and relevant outputs without retraining. This approach is particularly valuable for use case diagram generation, where precise and context-aware responses are essential. LLMs can interpret prompts differently depending on their structure, wording, and the amount of contextual information included [21].

Effective prompt design can significantly impact the quality of the generated outputs, influencing the resulting diagrams' syntactic and semantic accuracy. The main strategies for prompt engineering include Direct Prompting, Iterative Prompting, and Role-Based Prompting, each offering distinct advantages depending on the complexity and requirements of the task. Together, these strategies form a flexible toolkit for optimizing LLM outputs, allowing developers and researchers to tailor model responses to the specific needs of their projects. The choice of strategy depends on the complexity of the task, the desired level of detail, and the intended application of the generated outputs [22].

2.4.1 Direct Prompting

Direct prompting involves providing the LLM with a straightforward, unambiguous instruction. This approach relies on the model's pre-trained language understanding and is particularly effective for well-defined tasks where the desired output is clear. For example, a direct prompt for generating a Use Case Diagram might be:

- ***Prompt**: Generate a Use Case Diagram for an online shopping system, including actors like customer, administrator, and payment gateway.*

Direct prompting is often the simplest and fastest way to generate initial outputs. Still, it can struggle with complex requirements, as the model may lack the necessary context to understand the task fully [22].

2.4.2 Iterative Prompting

Iterative prompting is a more dynamic approach, involving a sequence of prompts designed to refine and improve the model's output over multiple interactions. This strategy is beneficial when the initial response is incomplete or lacks detail. For example, the process might start with a general prompt to identify key use cases, followed by targeted prompts to expand or correct the model's understanding:

1. ***Initial Prompt**: List the main use cases for a library management system.*
2. ***Follow-Up Prompt**: Expand on the 'Borrow Book' use case, including preconditions, main flows, and alternate flows.*

Iterative prompting allows for greater flexibility and precision, making it a powerful technique for complex, multi-step tasks where initial responses often require refinement [22].

2.4.3 Role-Based Prompting

Role-based prompting frames the LLM's responses by assigning it a specific professional role, such as a software engineer, project manager, or domain expert. This approach helps align the model's outputs with the language, context, and priorities typical of the assigned role, improving the relevance and accuracy of the responses. For instance:

- ***Prompt**: You are a senior software architect. Generate a Use Case Diagram for a hospital management system, including actors like doctors, nurses, patients, and administrators.*

This technique can produce more contextually accurate outputs by framing the task from a specific perspective, making it particularly useful for domain-specific requirements engineering [22].

2.5 Exploring the Role of Large Language Models in Requirement: A Literature Review

2.5.1 Introduction

Large-scale language models have emerged as powerful tools in natural language processing, enabling significant advances in various fields, including software engineering. These models can help automate the traditionally manual process of identifying and transforming user needs into structured system specifications. Hence, this step has enabled them to perform well in tasks such as text generation, translation, summarization, and question answering, often with impressive accuracy. Traditionally, requirements gathering has been a manual, time-consuming process that relies heavily on clear communication between stakeholders and developers. Poorly defined requirements commonly cause project delays, budget overruns, and even outright failures. For instance, recent studies [23] have demonstrated the potential of LLMs to automate the generation of UML diagrams from raw text, effectively bridging the gap between human language and machine-readable

models. However, despite their promise, several challenges are associated with applying LLMs to requirements engineering. The accuracy and quality of the generated models depend heavily on the quality of the input text and the ability of the model to understand domain-specific language. Moreover, these systems can struggle with natural language's inherent ambiguity and variability, potentially leading to incorrect or incomplete specifications if not carefully managed [23]. This review will explore the current state of LLM applications in requirements gathering in the following sections, examining both their potential benefits and the challenges they present. Case studies and practical examples will be included to illustrate the effectiveness of these models in real-world scenarios.

2.5.2 Leveraging Open Source LLMs for Software Engineering Education and Training

The paper "Leveraging Open Source LLMs for Software Engineering Education and Training" explores how open-source Large Language Models like ChatGPT, Mistral, and LLaMA (Large Language Model Meta AI) can support software engineering education. The authors emphasize that LLMs can automate critical tasks such as requirements elicitation, code generation, and project management, reducing the cognitive load on students and improving learning outcomes. A key advantage of open-source LLMs is their transparency and customizability, which allows educators to adapt these models to specific educational goals. For example, LLMs can help students draft initial functional requirements or simulate API (Application Programming Interface) interactions, supporting project-based learning where practical experience is essential (Figure 2.7). An example from the paper demonstrates this capability:

- ***Prompt**: Generate a list of non-functional requirements for an online learning platform focused on software engineering.*
- ***Expected Output:***
    1. *The platform should ensure secure user authentication and data privacy.*
    2. *It should support high availability with minimal downtime.*
    3. *The system should be scalable to handle many simultaneous users.*
    4. *The platform should provide a responsive user interface optimized for different devices.*

However, the paper also highlights several challenges, including the need for effective prompt design, the potential overreliance on AI, and concerns about data privacy. These limitations underscore the importance of using LLMs as complementary tools in

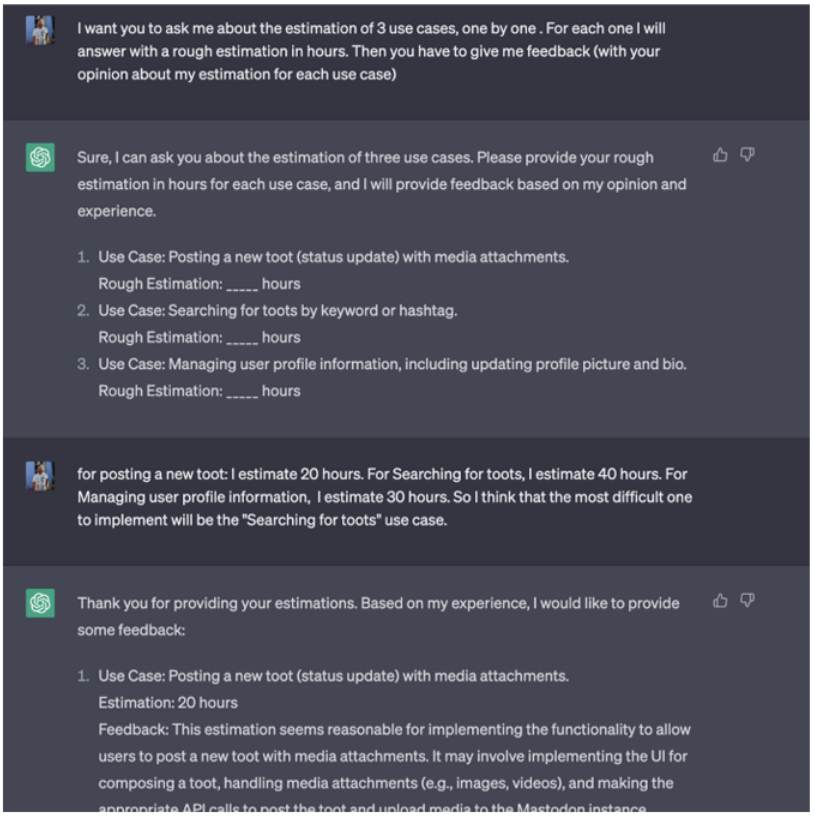

Figure 2.7. GenAI can play the role of an external evaluator of students' goals, estimates, and plans [24]

education, supporting but not replacing traditional teaching methods to ensure students develop critical problem-solving skills [24]

2.5.3 From User Stories to UML Diagrams Driven by Ontological and Production Model (2021)

UML diagrams include class, use case, and package diagrams. User stories, commonly used in agile methodologies, are typically written in natural language, which can introduce ambiguity and inconsistency into the design process.

To address this, the authors use a combination of Stanford CoreNLP for natural language processing and Prolog for rule-based relationship extraction. The approach also integrates an ontology to capture domain-specific knowledge, allowing the system to identify equivalent relationships and refine use cases. This ontology helps overcome the limitations of purely rule-based methods by providing a structured understanding of the components within user stories, such as actors, actions, and objects.

The approach is shown in the following Figure 2.8.

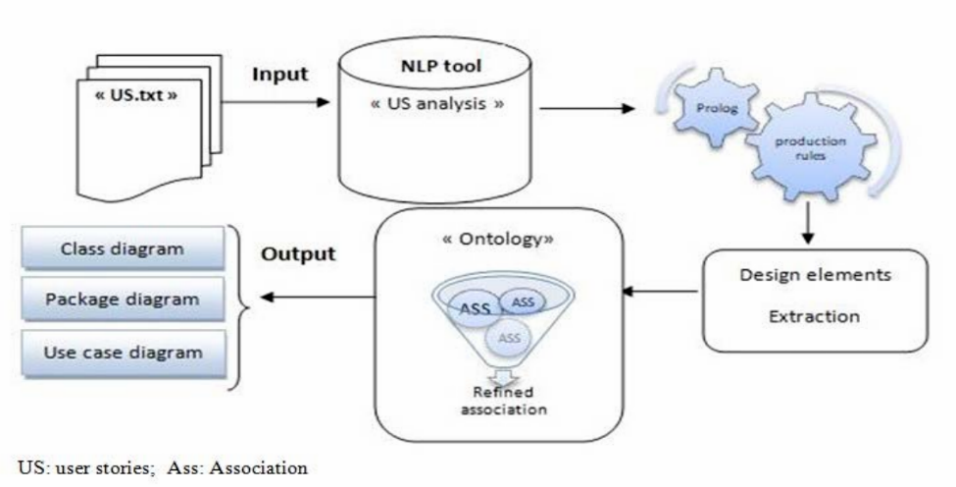

Figure 2.8. Architecture of the Proposed Approach [25]

The developed tool is implemented with Python. It generates class diagrams, use case diagrams, and package diagrams from user stories. It has been validated through multiple case studies, demonstrating its effectiveness in producing accurate UML diagrams.

Unlike previous methods, this approach significantly reduces redundancy and error rates by refining associations and eliminating unnecessary elements in the generated diagrams. The text analysis, relationship extraction, and ontology integration process is designed to streamline the transition from unstructured user stories to structured UML models, providing a more automated and reliable approach to requirements modeling [25].

2.5.4 Embracing Large Language Models for Medical Applications: Opportunities and Challenges (2023)

This paper illustrates how Large Language Models (LLMs) can significantly improve clinical decision-making and patient outcomes. These models, including BioBERT, ClinicalBERT, and BlueBERT, have been adapted for medical contexts, enabling them to perform in tasks like: named entity recognition, relation extraction, and question-answering (some of the applications are reported in Figure 2.9).

These models capture the specialized vocabulary and complex relationships typical of medical texts by leveraging domain-specific training data from sources like PubMed abstracts and clinical notes.

The paper also emphasizes the importance of continuous dynamic training to keep these models updated with rapidly evolving medical knowledge.

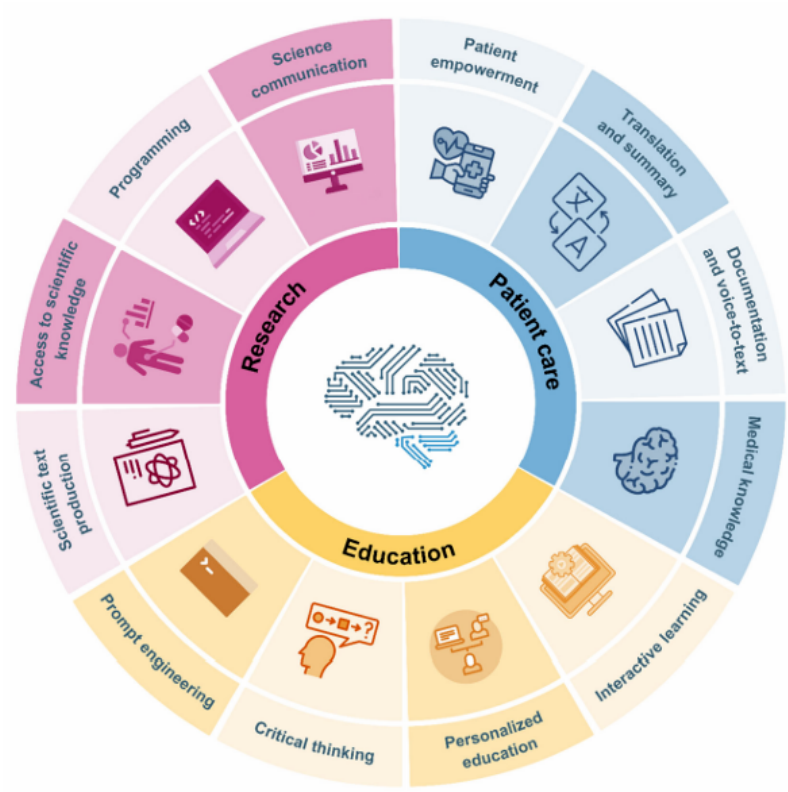

Figure 2.9. Overview of potential applications for LLMs in medicine [14]

Regular updates ensure that LLMs reflect the latest research and clinical guidelines, a critical factor in high-stakes fields like oncology and infectious diseases. Moreover, the authors highlight the need for interdisciplinary collaboration to address challenges like data privacy, model transparency, and the ethical implications of AI in healthcare. Effective integration of LLMs into clinical workflows requires collaboration between medical professionals, data scientists, ethicists, and policymakers to ensure that these technologies are both practical and ethically sound. Finally, the paper underscores the importance of education and training for healthcare professionals. Thus, this includes technical skills in AI and machine learning and a deep understanding of the ethical and practical challenges associated with using LLMs in medical practice [14].

Chapter 3

Methodology

3.1 Research Questions and Hypotheses

In the field of software engineering, Large Language Models have demonstrated outstanding potential in the transformation of Natural Language Processing tasks. An area of interest is the generation and evaluation of Use Case Diagrams. These are essential for defining the functional requirements of software systems. Hence, the diagrams capture the interactions between users (actors) and the system by providing a high-level view of the intended functionality. Because of these diagrams' critical role in the early stages, an accurate capture of user requirements is a focus of this thesis. The main objective of this work is to evaluate the effectiveness of LLMs in generating and validating use case diagrams from natural language inputs. Thus, this involves assessing the quality of the diagrams produced and exploring the impact of the prompting strategy on the accuracy and completeness of these outputs. To guide this investigation, we have defined the following research questions:

- RQ1: What prompt engineering strategy successfully improves the quality of Use Case Diagrams generated by different LLMs?
- RQ2: How accurately can different Large Language Models generate Use Case Diagrams from natural language requirements compared to human-designed diagrams?
- RQ3: How can the quality of Use Case Diagrams be effectively assessed using metrics for syntactic, semantic, and pragmatic correctness?

These research questions aim to address the main challenges of this thesis: focusing on the potential of LLMs to automate a traditionally manual and error-prone part of software design.

The three questions also aim to identify the most effective suggestion strategies for guiding the results of LLMs and to develop a robust framework for evaluating generated diagrams against human-created reference models. The following sections of this chapter will describe the methodologies employed to answer these questions. We started by creating a diverse repository of exercises, applying an LLM prompting strategy, and defining metrics for evaluating diagram quality. This study aims to provide a comprehensive assessment of LLM capabilities in the context of software requirements engineering, aligning directly with the primary focus of this thesis.

3.2 Repository Creation (collection of exercises)

To carry out the studies for this thesis, an existing archive of 17 exercises covered in the study "*Evaluating large language models in exercises of UML use case diagrams modeling*" (2025) [4] was used. However, during the work, they added six more exercises. They modified two existing exercises with new ones, thus increasing the critical role of these diagrams in the number of experiments and complexity. These exercises were sourced from various academic materials or online repositories and were selected to reflect various application domains. The goal was to present multiple challenges commonly faced in modeling UML Use Case Diagrams. The repository includes 23 exercises, each carefully reviewed to ensure relevance and balance across multiple contexts. These exercises are structured to highlight different levels of diagram complexity and to simulate real-world requirements modeling tasks. Each exercise is composed of three key components:

- A comprehensive textual description outlining a given system's functional and non-functional requirements.
- A reference to the source, where applicable, to ensure transparency and allow for deeper understanding or further research.
- A validated reference solution designed using the UML tool "Apollon", provided by the Technical University of Munich and serving as a reference for evaluating diagrams generated by LLM.

The digital appendix containing the complete collection of exercises and their structured solutions is available in this thesis's appendix A. Instead, a summary of the exercises' sizes, in the form of actors, use cases, and include and extend relationships, is presented in Table 3.1.

Exercise	Actors	Use Cases	Includes	Extends
1	4	7	0	0
2	3	11	0	0
3	4	8	0	0
4	4	4	0	0
5	2	7	3	0
6	3	6	1	0
7	1	5	0	0
8	1	7	0	0
9	6	7	0	0
10	3	3	0	0
11	4	7	1	3
12	1	9	2	2
13	5	11	0	0
14	2	11	3	0
15	3	7	0	0
16	3	5	0	0
17	2	12	0	0
18	3	8	1	1
19	3	15	4	2
20	3	6	4	0
21	4	9	2	3
22	4	9	4	0
23	6	4	2	0

Table 3.1. Details about the selected exercises

3.2.1 Exercise Selection and Complexity

The selection process aimed to gather exercises of varied difficulty to thoroughly evaluate the capabilities of both human participants and LLMs in producing accurate and effective UML Use Case Diagrams. Exercises were chosen based on clarity, completeness of requirements, domain variety, and their potential to reflect real-world software systems. To ensure a systematic evaluation, each exercise was assigned a difficulty score on a scale from 1 to 5, defined as follows:

1. Very easy tasks involving a minimal number of actors and clearly defined use cases.
2. Slightly more complex exercises introducing a few additional interactions or actors.
3. Moderate tasks with some complexity that may involve include/extend relationships and several actors.
4. Hard exercises that require deep understanding of UML concepts and involve several

use cases.

5. Realistic system simulations that involve extensive actors, complex conditional flows, and abstract generalizations.

To reduce bias and ensure objectivity, three reviewers independently rated each exercise before any diagram was generated. These reviewers, all familiar with UML modeling, assigned scores based only on the textual requirement description.

This structured evaluation process offers a balanced and consistent foundation for comparing the diagram generation performance of humans and different LLMs across varying levels of complexity.

3.2.2 Designated Solutions

For each exercise in the repository, a designated solution was developed to serve as a reference point for comparison. These reference diagrams were created by individuals with experience in UML modeling, using standard modeling tools to ensure consistency and compliance with UML notation guidelines. Each solution includes a visual Use Case Diagram, accompanied by metadata and annotations that clarify specific modeling choices, such as include/extend relationships, generalizations, and actor interactions. In addition to the diagram, a structured data representation (e.g., JSON format) is provided to facilitate automated evaluation by LLMs and validation tools. These designated solutions represent the ground truth in the evaluation process. They were reviewed for correctness and completeness, and serve as the benchmark for assessing the syntactic, semantic, and pragmatic accuracy of the diagrams generated by different LLMs. By maintaining a transparent and standardized reference for each exercise, the study ensures that comparisons of the diagrams are objective, consistent, and meaningful.

3.2.3 Example of Exercise

This section shows one of the twenty-three exercises in the archive. The exercise concerns a real application for requesting and creating quizzes and has an estimated difficulty level of ”4” (hard). The text is shown in below and the solution provided is in Figure 3.1.

```
Exercise 18
Description: A user can request a quiz for the system.
The system picks a set of questions from its database
and compose them together to make a quiz. It rates
the user's answers and gives hints if the user requests it.
```

```
In addition to users, we also have tutors
who provide questions and hints. And also, examiners must
certify questions to make sure they are not too trivial,
and that they are sensible. Make a use case diagram
To model this system. Work out some of your use cases.
```

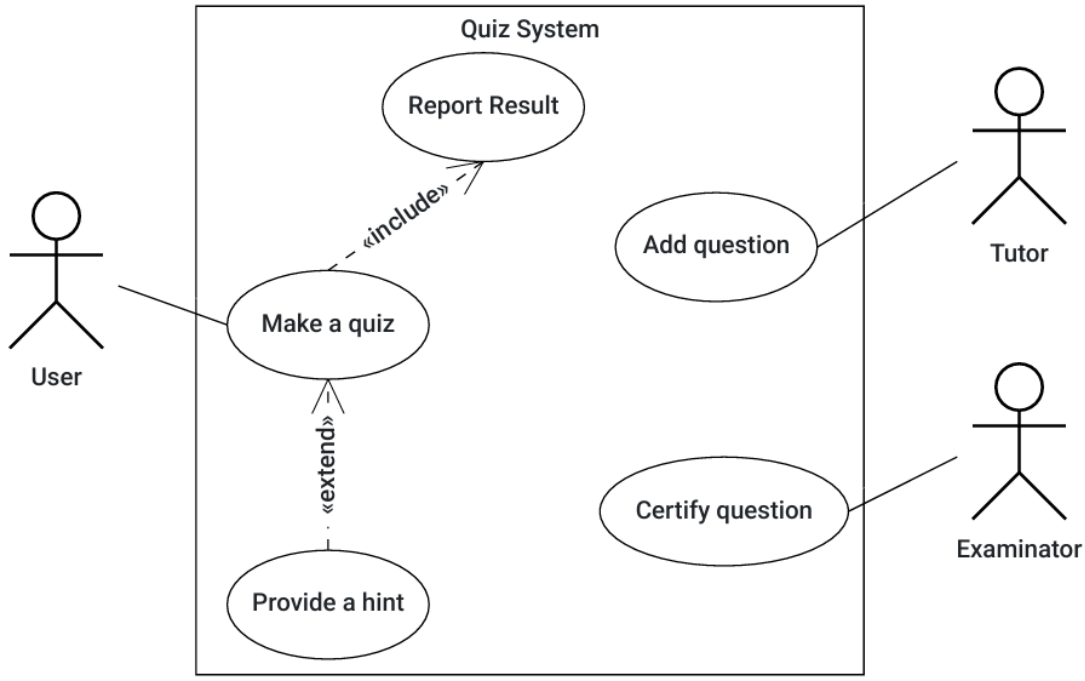

Figure 3.1. Exercise 18 UCD Solution

3.3 Use Case Diagram made by Human

3.3.1 Tool employed

This study used the Apollon tool to create human-generated use case diagrams that were employed as reference solutions. In addition, the LLM code generated from the text of an exercise was uploaded to the same tool and used to create the corresponding diagram. Apollon is a browser-based modeling environment developed by the Technical University of Munich, specifically designed for educational use in software engineering courses. It supports the creation of UML Use Case and BPMN diagrams, focusing on ease of use, accessibility, and integration with automated assessment tools (Figure 3.2).

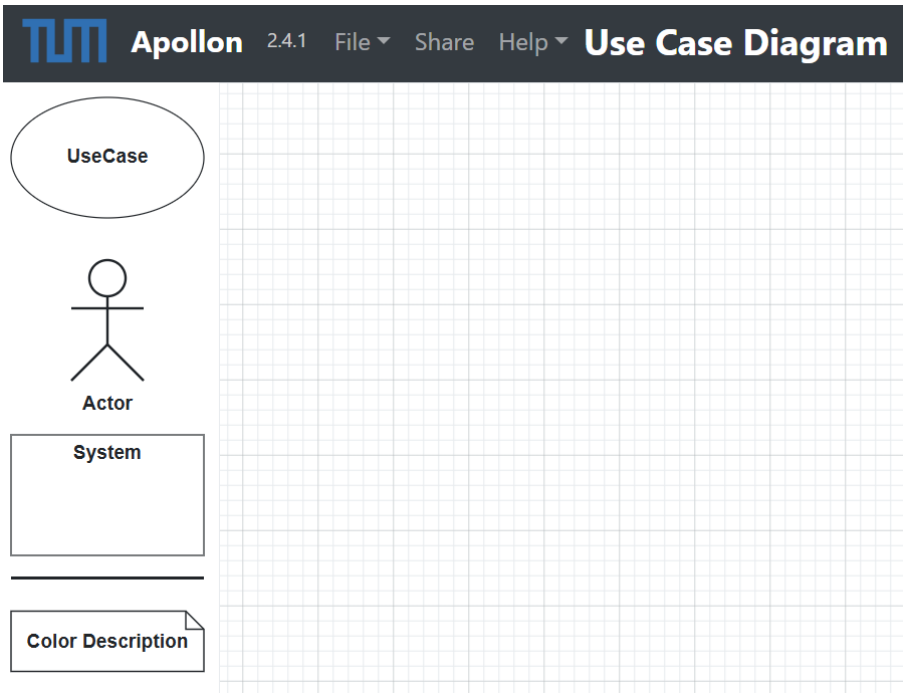

Figure 3.2. Apollon Tool User Interface

The decision to use Apollon was based on several factors. First, its web-based nature eliminates installation barriers, allowing consistent use across different operating systems. Second, it offers an intuitive user interface that is well-suited for novice and experienced users, making it ideal for academic environments. Another important reason for selecting Apollon is the ability to export diagrams in JSON format. This feature was essential for the experimental setup, as it allowed the Use Case Diagrams created by humans and LLMs to be downloaded and stored in a machine-readable structure.

These JSON files were then used in the comparison and evaluation processes alongside LLM-generated outputs in the same format. Instead, the limitations of the Apollon tool are as follows.

- May abstract or simplify some UML elements.
- Performance may be reduced with very large or complex diagrams.
- There are fewer customization and export options than professional UML tools.

3.3.2 Rules adopted

To set up a structured and realistic modeling environment, a specific series of rules was established to create Use Case Diagrams manually. These rules were intended not only to ensure uniformity and methodological consistency, but also to simulate typical constraints of professional settings, such as limited time, isolated work, and the absence of reference materials. Adopting these constraints ensures a fair basis for comparing the outputs produced by LLMs. These constraints were carefully selected to balance difficulty and fairness, enabling a reliable and consistent environment in which the modeling capabilities of human designers could be tested against those of AI models. The following rules were applied:

1. Time-constrained modeling: Each diagram was to be completed within a 30-minute timeframe. This rule replicates real-world time pressures and discourages over-refinement, promoting efficient decision-making.

2. Independent work without aid: Participant was required to complete the exercises without consulting external resources such as books, notes, or online platforms. This constraint ensured that the models reflected only the participants' understanding and reasoning abilities.

3. No Collaboration Permitted: All diagrams were created individually, without discussion or teamwork. This allowed for an objective assessment of individual skill and provided a direct comparison to the autonomous output of LLMs.

3.4 Use Case Diagram Generation with LLMs

3.4.1 Selected LLMs

During the research, two large language models (LLMs) were selected: OpenAI's ChatGPT and DeepSeek. The latter were used to evaluate their ability to generate use case diagrams from natural language requirements. These models were chosen based on their popularity, accessibility, and proven track record in natural language understanding and software engineering tasks.

1. ChatGPT is a popular conversational agent built on the GPT-4 architecture, which is known for its awesome performance in understanding and generating coherent and relevant text. It has been widely studied in academic and industrial fields for its versatility and usability in prompt-based tasks.

2. DeepSeek-V3, on the other hand, is a new entrant in the LLM field. It is a powerful AI model that specializes in long-context understanding and document analysis. Known for its strong coding and math skills, it offers free access with web search capabilities, making it a competitive alternative to GPT-4 for research and real-time tasks.

Therefore, the models were tested under consistent conditions using the same set of exercises from the repository.

For instance, using the quiz creation and request exercise again, the solution developed by ChatGPT-4 and DeepSeek is illustrated in the Figure 3.3 and the Figure 3.4.

In the Appendix C, a first part of the JSON code generated using the Appollon tool during the resolution of Exercise 18 is reported.

3.4.2 Prompt adopted

A role-based prompt strategy generated use case diagrams with large language models (LLMs). This approach involves assigning the model a defined role within a specific scenario. At the same time, a context is provided that helps to adapt its output to the expectations of the task. The LLMs used, ChatGPT and DeepSeek, were prompted using the following structured instruction:

- "***You are a student enrolled in a master's degree program in engineering and are attending the Information Systems course. Generate a UML use case diagram (with include, extend, and generalize connections) from the text of the uploaded exercises. The audience consists of the course professors and assistants. An example of the solution format is also included, which must be in JSON format. The JSON format must be supported by the tool https://apollon.ase.in.tum.de/. The JSON format must be downloadable.***"

This prompt is designed to serve several purposes:

- Contextualisation: the student profile encourages academically appropriate output in both form and tone.
- Format specificity: by requiring valid Apollon-compatible JSON, the prompt ensures a computer-readable structure suitable for assessment.
- Content structure: The inclusion of specific UML elements (e.g., includes, extends, generalizes) guides the model towards more complete and semantically rich diagrams.

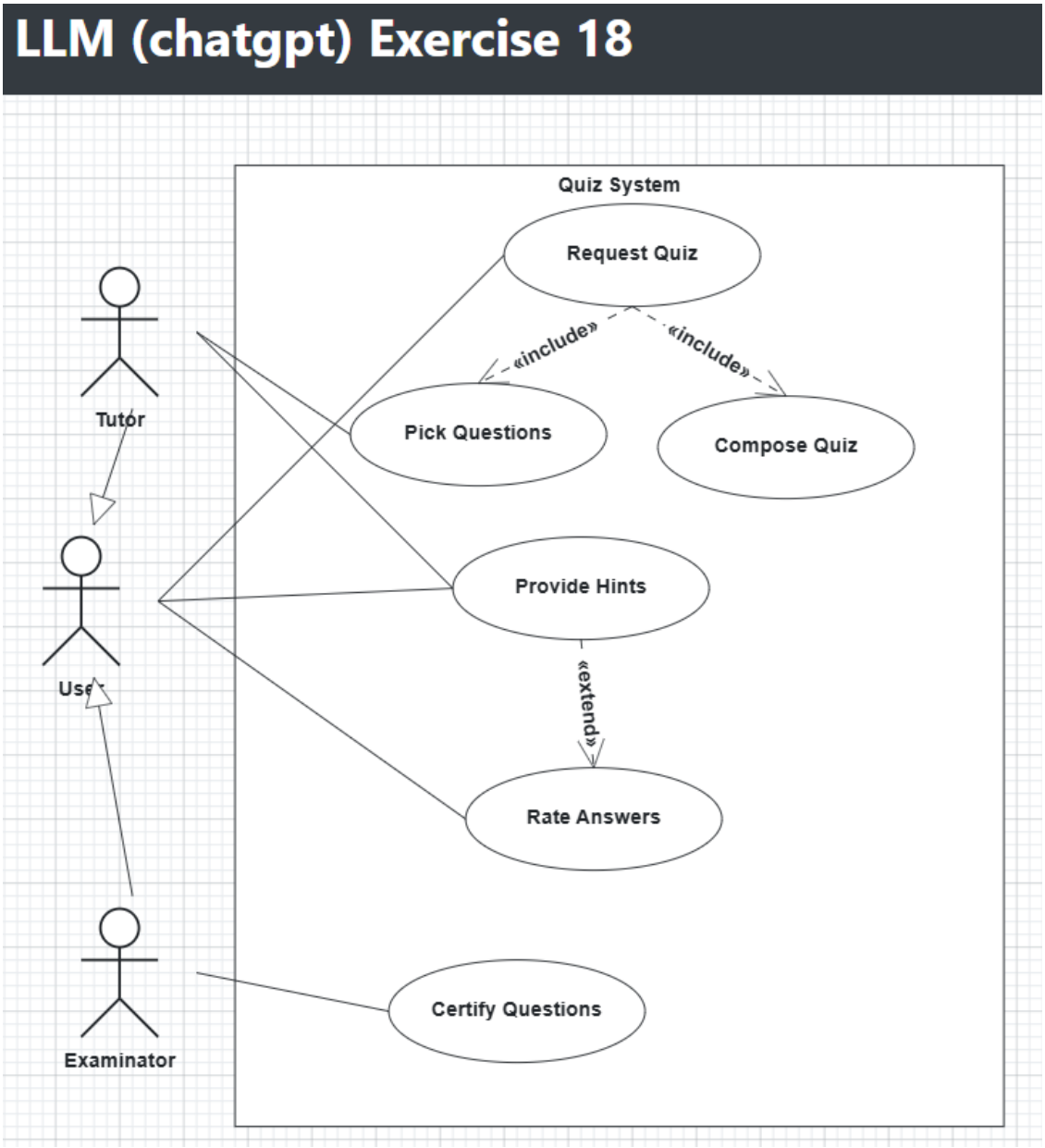

Figure 3.3. Solution made by ChatGpt-4

Using role-based and task-specific prompts builds on insights and the importance of prompt engineering to maximize LLM performance on structured tasks such as code analysis and model generation. The findings highlight that well-defined prompts significantly improve accuracy, consistency, and domain alignment compared to generic or insufficiently specified instructions [26].

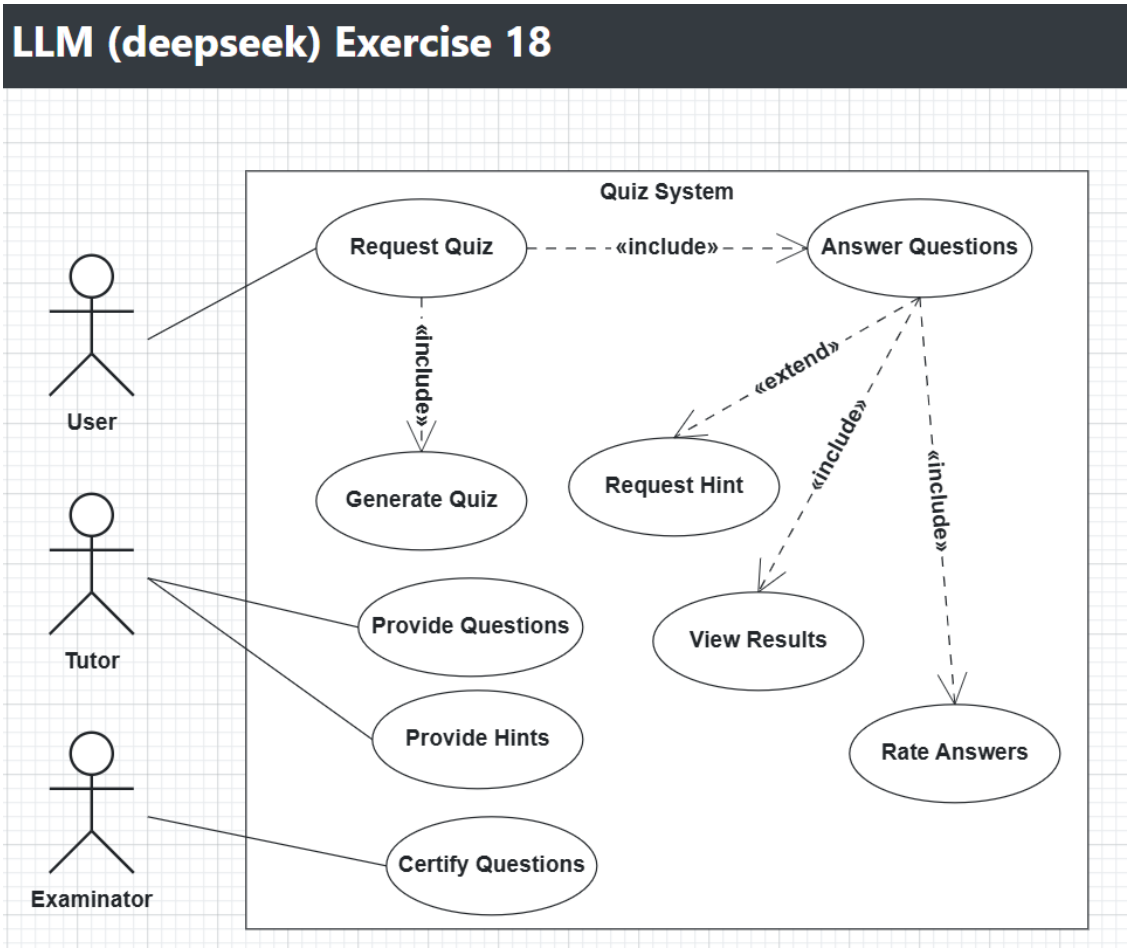

Figure 3.4. Solution made by DeepSeek-V3

3.5 Evaluation Framework

A collection of 23 UML Use Case Diagram exercises was used to compare the quality of diagrams produced by various agents, including a human modeler, ChatGPT (GPT-4), and DeepSeek. Each exercise includes a textual description and a reference solution that serves as the ground truth for testing correctness and completeness. The individual who did this thesis, pursuing a master's in Management Engineering, separately developed a solution to every exercise and then created pertinent outputs from ChatGPT and DeepSeek through a standard prompt. Both human and LLM-generated solutions were manually inspected and checked along the three main dimensions: completeness, redundancy, and quality measures, including syntactic, semantic, and pragmatic correctness. Therefore, the evaluation focused on these three key dimensions, in line with previous

approaches to UML Use Case Diagram and Class Diagram evaluation [4, 27].

3.5.1 Completeness and Redundancy Rates

Two primary quantitative measures, completeness and redundancy, were compared to assess the quality and accuracy of the produced UML use case diagrams. These measures allow for a systematic and measurable comparison of diagrams created by human participants and diagrams produced by large language models, such as ChatGPT and DeepSeek.

Completeness Rate

Completeness rate defines how effectively an output diagram addresses all the significant elements specified in the reference solution. It is quantified by dividing the number of elements identified correctly by the total number of elements anticipated in the diagram. To determine this, we considered the following [4]:

- Correct Actors: Actors shown in the developed diagram that correspond to their equivalents in the reference solution, either by name, synonymous terms, or obvious contextual similarity.
- Correct Use Cases: Use cases that match the reference solution in role and naming. It is also important that the corresponding actor(s) likewise comply; if a use case is mistakenly assigned to an actor, it is considered incorrect.
- Correct Include and Extend Relationships: These relationships link pairs of use cases in accordance with the reference solution in a way that the direction and type (whether extend or include) are correct.

The completeness rate was then calculated using the following formula 3.1:

$$\text{Completeness Rate} = \frac{\text{Correct Actors} + \text{Correct Use Cases} + \text{Correct Relationships}}{\text{Total Elements in the Reference Solution}} \tag{3.1}$$

Redundancy Rate

The redundancy ratio calculates the percentage of the unnecessary or non-essential elements contained within the resulting diagram elements not contained in the provided solution [4]. This encompasses:

- Superfluous actors or use cases outside the problem domain.

- Improper generalizations or evidence-free correlations from the exercise description were inserted.

The redundancy rate was delineated as follows:

$$\text{Redundancy Rate} = \frac{\text{Unmatched Actors} + \text{Unmatched Use Cases}}{\text{Total Elements in the Generated Diagram}} \tag{3.2}$$

This metric also ranges from 0 to 1, where 0 denotes no redundancy; all the elements present are meaningful and necessary.

These two indicators were consistently applied across all diagrams generated by human, ChatGPT, and DeepSeek actors. They formed the basis for assessing each solution's syntactic and semantic precision in alignment with the research methodology described above.

3.5.2 Metrics: Syntactic, Semantic, Pragmatic

In order to make a comprehensive analysis of UML Use Case Diagrams' (UCDs) quality, this study employs a systematic assessment framework founded on three interrelated dimensions: syntactic, semantic, and pragmatic quality. This kind of classification follows the example of recognized scientific literature concerning conceptual modeling and software design quality [27].

Syntactic Quality

Syntactic correctness is a measure that assesses whether a diagram complies with the formal syntax and construction rules specified in the UML standard. To evaluate syntactic quality, the following rules were taken into account:

- Use only valid UML elements (e.g., actors as stylised figures)
- Place actors outside the system boundaries and use cases inside.
- Use "include" and "extend" only between use cases, never from actor to use case or vice versa.
- Apply the correct directionality for relationships:"include" points from the base use case to the included use case; "extend" points from the extension use case to the base use case.
- Use generalization only between elements of the same type (actor-actor or use case-use case).

Semantic Quality

Semantic quality reflects the degree to which the diagram correctly depicts the meaning and structure of the system in the textual specifications. This dimension is more concerned with the model content accuracy than with its visual presentation or formatting issues. The rules for semantic quality are:

- Include all required actors and use cases described in the textual scenario.
- Assign each use case to the correct actors, based on their roles and objectives.
- Avoid omitting the main objectives of the users; every essential function of the system must be modeled as a use case.
- Apply ”include” only when one use case always calls another as part of its logic.
- Use ”extend” only when a use case optionally adds behavior to another under specific conditions.
- Avoid vague or too many generic use cases that do not reflect the user's concrete intentions.

The semantic quality evaluation was partly based on the completeness rate as described under Section 3.5.1, but also included qualitative misrepresentations, i.e., where use cases exist but are functionally incorrect.

Pragmatic Quality

Pragmatic quality estimates the degree to which the diagram can effectively communicate its structure and intention to the human viewer. This entails readability, understandability, and clarity of abstraction. Pragmatic correctness evaluation criteria are:

- Arrange the elements so that they are easy to read: align the actors on the sides, center the use cases, minimize the crossing of lines.
- Use consistent naming: actors as roles, use cases as verbal phrases.
- Avoid confusion by omitting trivial or obvious actions that distract from the main functionality.
- Eliminate unused or redundant elements.
- Keep the diagram concise: group related use cases in a meaningful way instead of scattering them.

- Note or comment only when necessary to clarify complex behaviors or conditions.
- Ensure that the diagram clearly communicates the main functional vision to both technical and non-technical stakeholders.

To quantify this, the diagrams were evaluated based on their redundancy rate (see Section 3.5.1), and also annotated for subjective issues such as overcomplication, lack of abstraction, or misalignment of the layout. These annotations were derived from the judgment of the evaluator (the thesis author), based on their knowledge of UML modeling and stakeholder usability.

Together, these three dimensions formed the foundation for the quality assessment framework of the diagram used in this thesis to support the evaluation and comparison of human- and LLM-generated solutions for Research Question 3.

3.5.3 Comparison Techniques

A systematic comparative framework was used to evaluate the quality of Use Case Diagrams (UCDs) generated by different agents, namely Humans, ChatGPT, and DeepSeek, across a data set of 23 exercises. The framework was designed to ensure consistency and transparency in calculating correctness and quality measurements.

Element-Level Matching

For each exercise, the diagrams created by all three subjects were compared to a model solution. The procedure involved:

- Manual browsing and pulling out important items from each diagram, including:
  - Actors
  - Use Cases
  - Relationships (include, extend, associations)
- Verifying whether every entity in the generated diagram was the same as the respective entity in the reference solution. Equality was accepted when name and function were identical or semantically equivalent (e.g., synonym naming or conceptually correct assignment).

This procedure was used with all 23 exercises and with each of the three categories of actors, making a total of 69 comparisons.

Tabular Comparison in Spreadsheet

The testing procedure was properly recorded in a designed Excel spreadsheet, in which:

- Each row corresponds to one diagram (i.e., one of the 69 evaluated cases).
- The columns note whether each expected element was:
  - Matched correctly (match)
  - Incorrectly represented (error)
  - Missing
  - Redundant

Separate sections of the table were assigned to differentiate between the syntactic, semantic, and pragmatic levels, based on the criteria laid down in Sections 3.5.1 and 3.5.2.

Error Classification and Aggregation

Based on the recorded comparisons, the following actions were taken:

- It was classified by type (syntactic, semantic, or pragmatic).
- Completeness was defined as the ratio of correct elements to total reference elements.
- Redundancy was calculated as the ratio of unnecessary/excess items to the total items present in the created diagram.
- All metrics were aggregated at the exercise and actor level.

This framework allowed for a standardized quantitative comparison and supported the numerical evaluation for the results and discussion chapter.

Chapter 4

Results and Discussion

4.1 Introduction

This chapter presents a detailed evaluation and results regarding the effectiveness of large language models (LLMs) in generating UML use case diagrams (UCDs). Following the methodology outlined in Chapter 3, the assessment focuses on comparing the diagrams produced by LLMs with human-generated reference diagrams in terms of syntactic correctness, semantic accuracy, and pragmatic usefulness. A set of 23 distinct use case modelling exercises, each with detailed textual descriptions and reference solutions, was used to measure the quality of the solutions generated by LLMs rigorously. The evaluation aims to determine whether diagrams produced by state-of-the-art LLMs such as ChatGPT-4 meet the quality standards expected in educational and practical software modeling contexts. The following paragraphs show the results in a quantitative way using completeness and redundancy metrics, along with metric rules to check how significant the differences are. In addition, a detailed example of a representative exercise is included to illustrate how the evaluation criteria were applied clearly. An analysis of errors also identifies common patterns in both successful and unsuccessful cases, providing insights into the practical implications of integrating LLMs into UML modeling teaching tools and processes.

4.2 Quantitative and Qualitative Evaluation

The following section summarizes the results of the comparison between the UML use case diagrams generated by a human solver, ChatGPT-4, and DeepSeek-V3. The evaluation includes quantitative metrics, completeness and redundancy rates, and qualitative

criteria, including syntactic, semantic, and pragmatic quality rules. Completeness and redundancy rates, which measure the accuracy and conciseness of the diagrams, are shown in the previous sections. These metrics combined provide a comprehensive view of the diagrams' structural correctness and overall communicative effectiveness. A boxplot graph was chosen to visualize these measures, as it effectively shows the distribution, central tendency, variability, and potential outliers within the dataset. Those were used to enable an intuitive comparison between the different solvers. Table 4.1 summarizes the minimum, maximum, and average completeness and redundancy rates for diagrams generated by human beings and those produced by ChatGPT-4 and DeepSeek-V3.

	Human			ChatGpt-4			DeepSeek-V3		
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
Completeness Rate	0.29	1	0.85	0.36	1	0.76	0.36	1	0.76
Redundancy Rate	0	0.42	0.14	0	0.56	0.16	0	0.45	0.17

Table 4.1. Rates for Human, ChatGpt-4 and DeepSeek-V3

4.2.1 Results Completeness Rate

The completeness rate evaluates the percentage of actors, use cases, and relationships that are correctly identified within the diagrams. From the numerical data in the Table 4.1, we can see that human-generated diagrams outperformed those generated by the LLM in terms of average completeness (0.85 vs. 0.76). Furthermore, human solutions started from a lower minimum (0.29), suggesting that while some diagrams had significant gaps, others were really complete. On the other hand, both ChatGPT-4 and DeepSeek-V3 had a higher minimum (0.36), showing more consistency at the lower end but lacking higher averages.

The completeness boxplot graph, in the figure 4.1, reveals significant differences among the solutions and reinforces the insight of the table. The box depicts the interquartile range (IQR), which shows the distribution of the central 50 % of the data. The human-generated diagrams present the highest median completeness rate, reflecting consistent correctness in most of the exercises, but with some considerably lower outliers. Both DeepSeek-V3 and ChatGPT-4 had similarly equivalent median values, though slightly lower than the human median.

The two LLM solutions also exhibit larger interquartile ranges, reflecting larger variation in completeness rates between different exercises compared to human-generated solutions. The whiskers extend the highest and lowest values within 1.5 times the IQR, and points outside of this range are plotted as outliers. Nevertheless, both ChatGPT-4 and DeepSeek-V3 have maximum completeness of 1, indicating that they can generate accurate diagrams in some scenarios.

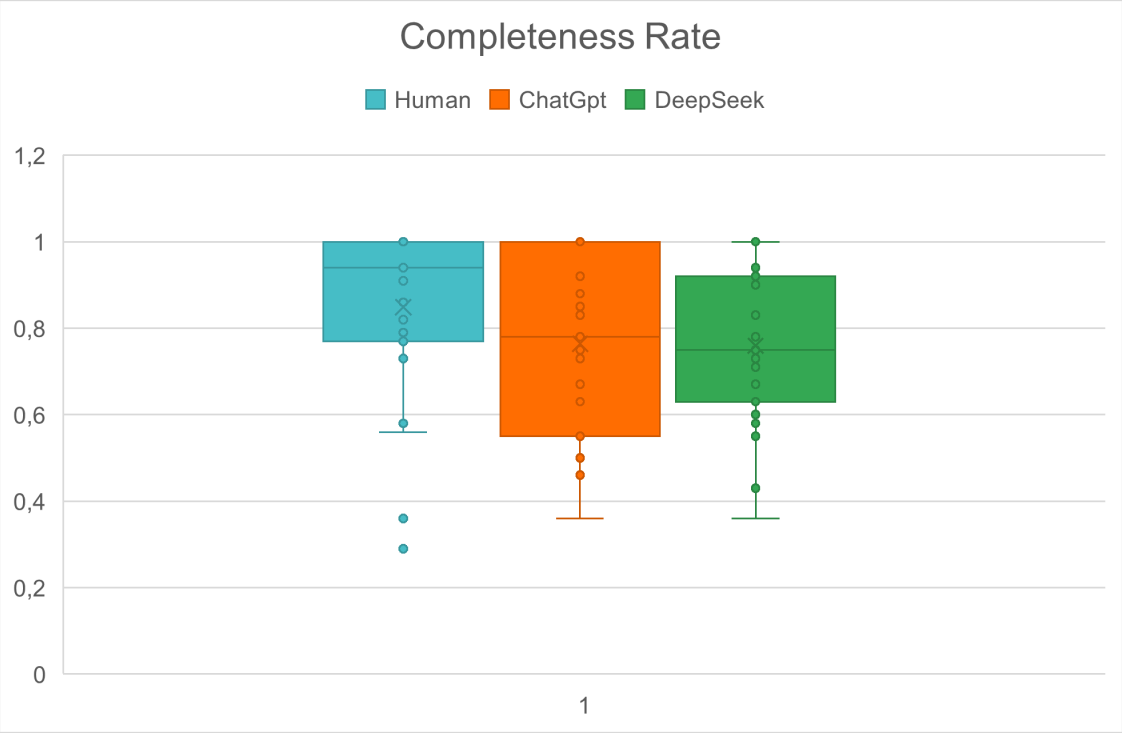

Figure 4.1. Completeness Rate Boxplot

4.2.2 Results Redundancy Rate

The redundancy rate evaluates the percentage of elements (actors and use cases) that are included in the generated diagrams but not present in the reference solutions, thus indicating redundant complexity. Table 4.1 summarizes these redundancy metrics.

The figures in the Table 4.1 show that humans generated the most concise diagrams on average (average redundancy = 0.14), while ChatGPT-4 (0.16) and DeepSeek-V3 (0.17) introduced slightly more redundant elements. The maximum redundancy observed was also lower for humans (0.42) than for ChatGPT-4 (0.56), showing higher peaks of inefficiency in the LLM results.

In Figure 4.2, these data are visually supported by the boxplot graph for redundancy rate, where human solutions show a lower and tighter IQR, pointing to more consistent performance in avoiding redundancy. ChatGPT-4 has the widest spread and highest maximum, indicating inconsistency in its ability to eliminate irrelevant elements. Instead, DeepSeek-V3 seems to be more stable than ChatGPT-4 in this metric, but still lags behind human performance.

Figure 4.2. Redundancy Rate Boxplot

In short, both the numerical tables and visual boxplots confirm that human modelers continue to outperform LLMs in terms of completeness and redundancy, even if only slightly. Boxplots provide valuable confirmation of statistical summaries, revealing average performance and variability and reliability of each approach. Despite these shortcomings, diagrams generated by LLMs show strong potential, sometimes matching human completeness, making them valid candidates as teaching and support tools in UML modeling activities.

4.2.3 Syntactic, Semantic and Pragmatic Assessment

In order to provide a more qualitative outlook on the performance of the generated diagrams, this subsection analyzes the solutions from a syntactic, semantic, and pragmatic

perspective. Thus, these dimensions allow for a deeper understanding not only of whether elements are included (completeness) or overrepresented (redundancy), but also of the extent to which the diagrams comply with modeling standards, represent the intended meaning, and satisfy the target audience. The table below summarizes the distribution of errors observed in the diagrams according to syntactic, semantic, and pragmatic criteria. These figures offer a measurable snapshot of the specific types of qualitative deviations each solver commits. All the human-generated and LLM-generated diagrams were confronted with the given solutions to evaluate their quality. The results are reported in the Figure 4.3.

Exercise	**Syntactic errors**	**Semantic errors**	**Pragmatic errors**
Exercise 1	1	7	2
Exercise 2	0	9	1
Exercise 3	0	6	2
Exercise 4	0	0	0
Exercise 5	0	0	0
Exercise 6	0	4	1
Exercise 7	1	0	1
Exercise 8	0	1	0
Exercise 9	0	0	0
Exercise 10	0	0	0
Exercise 11	1	4	2
Exercise 12	0	3	1
Exercise 13	1	4	2
Exercise 14	0	4	2
Exercise 15	0	3	1
Exercise 16	1	2	1
Exercise 17	0	4	1
Exercise 18	0	3	1
Exercise 19	0	0	0
Exercise 20	0	0	0
Exercise 21	0	2	0
Exercise 22	0	5	1
Exercise 23	0	1	1

Table 4.2. Errors made by humans across exercises.

The aggregated data from 23 exercises, which are summarized in Table 4.5 and in the Tables 4.2, 4.3, 4.4, offers the comparison of human and LLMs capabilities in UML use case diagram creation during each exercise.

Exercise	Syntactic errors	Semantic errors	Pragmatic errors
Exercise 1	1	2	0
Exercise 2	1	8	3
Exercise 3	0	8	3
Exercise 4	0	4	2
Exercise 5	0	8	2
Exercise 6	1	2	1
Exercise 7	0	3	1
Exercise 8	1	2	1
Exercise 9	0	2	1
Exercise 10	1	3	2
Exercise 11	2	3	2
Exercise 12	1	8	2
Exercise 13	2	10	1
Exercise 14	0	2	1
Exercise 15	2	4	1
Exercise 16	3	3	1
Exercise 17	0	8	1
Exercise 18	2	4	1
Exercise 19	2	10	2
Exercise 20	2	3	1
Exercise 21	0	8	2
Exercise 22	0	4	3
Exercise 23	2	2	1

Table 4.3. Error analysis for ChatGPT across different exercises

Syntactic Assessment

Throughout 23 exercises, the human-generated diagrams made a total of 6 syntactic errors, while the LLM-generated diagram exhibited a slightly higher number, totaling 20 errors for ChatGpt-4 and 17 for DeepSeek-V3. This may indicate better adherence to UML standards by the human designer. The box plot graph of syntactic errors represented in the Figure 4.4 provides a better visualization of how consistently each solver complies with UML syntax. Hence, in a more detailed way, we can see that:

1. The human solvers demonstrated a minimal error range, with almost all values clustered close to zero. This is evident in the compact interquartile range (IQR) and small tail extensions reaching a maximum of 1 error, confirming strong control over modeling language conventions.

2. ChatGPT-4 shows a visibly wider box with a higher upper tail reaching a maximum

Exercise	Syntactic errors	Semantic errors	Pragmatic errors
Exercise 1	0	2	1
Exercise 2	1	8	3
Exercise 3	0	10	2
Exercise 4	0	3	1
Exercise 5	0	7	2
Exercise 6	0	5	2
Exercise 7	1	3	1
Exercise 8	0	3	1
Exercise 9	0	1	1
Exercise 10	1	3	2
Exercise 11	0	3	2
Exercise 12	0	3	1
Exercise 13	0	7	1
Exercise 14	0	4	2
Exercise 15	0	4	1
Exercise 16	1	6	2
Exercise 17	1	3	2
Exercise 18	2	3	1
Exercise 19	0	8	2
Exercise 20	0	1	1
Exercise 21	0	2	1
Exercise 22	1	3	2
Exercise 23	1	3	2

Table 4.4. Summary of Errors in DeepSeek Exercises

	Syntactic Quality	Semantic Quality	Pragmatic Quality
Human	6	56	20
ChatGpt-4	20	105	35
DeepSeek-V3	19	101	33

Table 4.5. Errors Comparison of Quality Metrics

of 3 syntactic errors. This highlights more variable behavior, where some solutions stick to the syntax well while others diverge significantly.

3. DeepSeek-V3 performs similarly, with a narrower box than ChatGPT-4 but still larger than the human solver, indicating occasional syntactic errors.

Some of the most common issues for the LLMs were the incorrect use of "includes" and "extends" relationships or incorrect links between actors, all of which can compromise

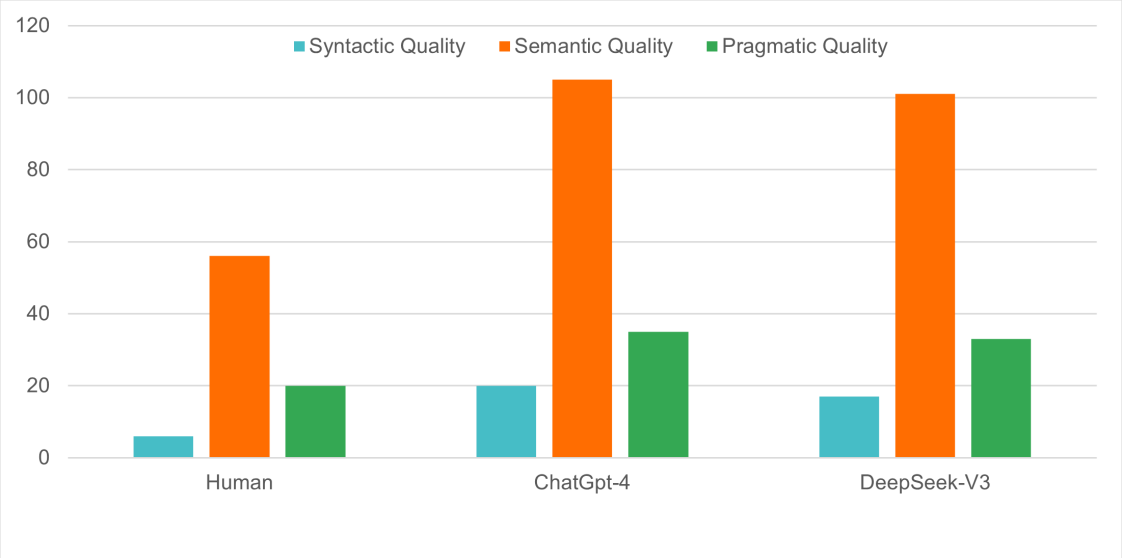

Figure 4.3. Number of errors divided in the three Qualities

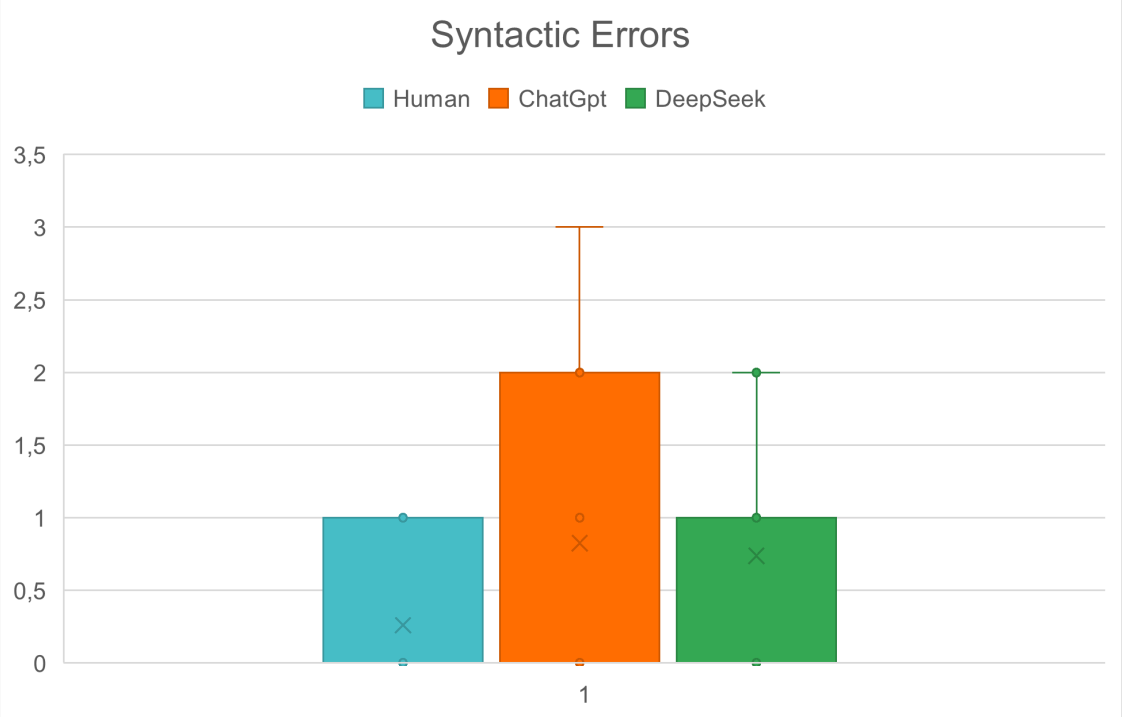

Figure 4.4. Syntactic Boxplot Graph

the formal correctness of the diagrams.

The boxplot graph, which shows a very compact distribution for humans with a firm control over UML syntax, is supported by the figures in the Table 4.6.

	Human			ChatGpt-4			DeepSeek-V3		
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
Sem. Errors	0	9	2.43	1	10	4.57	1	10	4.39
Syn. Errors	0	1	0.26	0	3	0.83	0	2	0.74
Prag. Errors	0	2	0.87	0	3	1.52	1	3	1.43

Table 4.6. Comparison of Errors Between Human, ChatGpt-4 and DeepSeek-V3

For instance, human-generated diagrams demonstrated exceptional consistency, with an average of only 0.26 errors, a maximum of just 1, and virtually no variability in the dataset. This is visually supported by the. Instead, ChatGPT-4 and DeepSeek-V3 showed a higher average number of syntactic errors, 0.83 and 0.74, respectively. These results demonstrate that, although LLMs benefit from structured outputs such as Apollon, they still need refinement to achieve the same level of syntactic rigor as human authors.

Semantic Assessment

The evaluation of semantic quality more evidently shows the difference between human actors and LLMs: human-generated diagrams accumulated 56 semantic errors across all exercises, whereas LLMs accounted for 105 for ChatGpt-4 and 101 for DeepSeek. Thus, this gap underscores the challenges faced by the LLM in accurately representing domain-specific knowledge and relationships within the diagrams. Starting from the numerical data reported in Table 4.6, the human diagramsLLMs produce showed an average of 2.43 semantic errors, with values ranging from 0 to 9. In contrast, ChatGPT-4 and DeepSeek-V3 recorded much higher averages, 4.57 and 4.39, respectively, both with a peak of 10 semantic errors. These results highlight a clear semantic gap between human-created diagrams and those produced by LLMs. The corresponding boxplot graph (Figure 4.5) for semantic errors visually reinforces these numerical insights, since:

1. Human solutions form a compact cluster with a narrow interquartile range (IQR) with most errors below five and demonstrating limited dispersion. These results indicate low variability and strong semantic alignment with the source requirements.

2. In contrast, ChatGPT-4's boxplot reveals a wide IQR and long whiskers, extending

Figure 4.5. Semantic Boxplot Graph

up to the maximum value of 10. This reflects a higher degree of inconsistency in ChatGPT-4's ability to interpret natural language requirements accurately.

3. DeepSeek-V3, although slightly more stable than ChatGPT-4, still shows significant semantic variation, as demonstrated by a moderately wide box and similar whisker length.

Both LLM solvers reached a maximum of 10 semantic errors, with ChatGPT-4 showing more extreme variance. These errors typically included misunderstandings of actor responsibilities, overgeneralization of use cases, or omissions of essential interactions. In summary, both the statistical data and the visual distribution show that LLMs, while capable of capturing general intent, often struggle with semantic precision in modeling. Human solvers clearly outperform LLMs in this dimension, thanks to their domain understanding and contextual reasoning abilities.

Pragmatic Evaluation

In terms of pragmatic quality, which assesses the diagrams' understandability, the human-generated diagrams contained 20 errors in total, compared to 35 for ChatGPT-4 and 33 for DeepSeek-V3. According to Table 4.6, the average number of pragmatic errors for human participants was 0.87, with a range between 0 and 2.

In contrast, ChatGPT-4 and DeepSeek-V3 recorded higher average numbers of errors, 1.52 and 1.43, respectively, with both reaching a maximum of 3. These results suggest that, while human solutions are more consistent pragmatically, diagrams generated by LLMs are subject to clarity and usability issues that could hinder effective communication or learning.

The boxplot graph of pragmatic errors further illustrates this:

1. Human-produced diagrams show a narrow interquartile range and relatively low dispersion, indicating consistent pragmatic quality. Most values cluster below the median of 1, with bars extending only up to 2, in line with the maximum value reported in the table.

2. The boxplots for ChatGPT-4 and DeepSeek-V3 reveal wider boxes and greater variability. Both models show a median around 1.5, with interquartile ranges ranging from approximately 1 to 2 and whiskers extending up to a maximum of 3. These distributions indicate that some diagrams generated by LLM were clear and useful, but many others lacked cohesion or had ambiguous labels, overly complex structures, or poorly organized relationships.

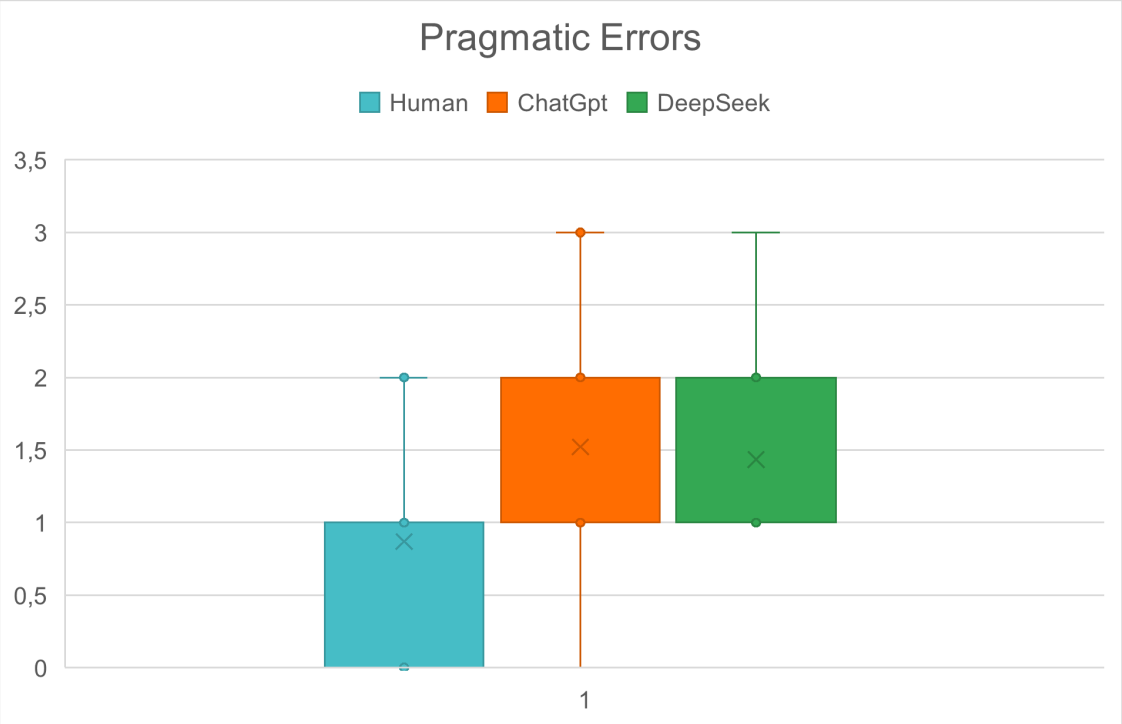

Figure 4.6. Pragmatic Boxplot Graph

In practical terms, these pragmatic defects can significantly affect how users interpret or apply diagrams, especially in educational contexts. Misplaced or redundant elements, inconsistent naming conventions, and poorly grouped features reduce the educational and communicative effectiveness of diagrams. Basically, while the average pragmatic performance of LLMs is not drastically worse than that of human participants, the greater variability and higher number of errors reveal important limitations. Ensuring the practical quality of automatically generated UML diagrams will require refined prompt strategies, clearer modeling conventions, or post-processing techniques to ensure readability and educational value.

4.2.4 Interpretation Across Syntactic, Semantic and Pragmatic Dimensions

To better understand the performance of LLM- and human-generated diagrams, it is helpful to analyze the trends emerging from quantitative metrics (completeness and redundancy rates) with those derived from quality assessments (syntactic, semantic, and pragmatic errors). The analysis is based on both the numerical summaries and the visual distribution models presented in the previous sections.

Completeness and Semantic Quality

Completeness is a direct reflection of semantic quality. When a diagram reaches high completeness, it means that the model accurately captures the actors, use cases, and their interactions, as described in the requirements text. This alignment is semantic in nature: it demonstrates that the modeler has understood the domain concepts and has successfully translated them into UML elements. The box plot and the summary table of semantic errors reinforce this interpretation. The human diagrams scored highest in completeness (mean = 0.85) and had the fewest semantic errors (mean = 2.43). In contrast, ChatGPT-4 and DeepSeek-V3, both with lower completeness (0.76), had significantly more semantic errors (mean of 4.43 and 4.26). This clearly suggests that completeness is a reliable superficial indicator of semantic fidelity. Hence, supports the conclusion that diagrams with lower completeness tend to omit key use cases or actors, resulting in a greater number of semantic misunderstandings.

Redundancy, Syntactic Validity and Pragmatic Clarity

Redundancy in UML diagrams represents structural excess, that is, elements that are unnecessary or not supported by the problem definition. High redundancy often results from syntactic errors or the inability to distinguish relevant from irrelevant content.

The table and boxplots show that humans exhibited the lowest redundancy (mean = 0.14), which is consistent with both the lowest syntactic error rate (mean = 0.26) and the lowest pragmatic error rate (mean = 0.87). On the contrary, LLMs tended to introduce more redundant elements (ChatGPT-4: 0.16, DeepSeek-V3: 0.17), which correlates with more syntactic (0.87, 0.83) and pragmatic errors (1.52, 1.48). The graphs confirm it: diagrams with more structural noise also present more layout, labeling, or complexity problems. Thus, this comparison highlights that redundancy often translates into cluttered or harder-to-read diagrams, reducing their practical and didactic usefulness.

Interdependent Relations

From the analyses of the quantitative and quality metrics, we could see that:

- Syntactic problems (e.g., improper use of "include," "extend," or actor associations) often cause structural irregularities, contributing to redundancy and increasing cognitive load.

- Semantic errors arise when use cases or actors are missing, irrelevant, or incorrectly related. These directly affect completeness.

- Pragmatic problems are most evident in messy, ambiguous, or overly complicated diagrams, often due to syntactic and semantic mismatches, as demonstrated by high redundancy and inconsistency in the quality of diagrams.

In conclusion, completeness primarily measures the semantic success of a diagram, while redundancy encompasses both syntactic discipline and pragmatic clarity. High completeness and low redundancy indicate coherent, correct, and communicatively effective diagrams, traits that distinguish the results obtained by individuals from those generated by an LLM in this study.

4.3 Example of an Evaluated Exercise

To better understand the methodology of the previous chapter, an example will be made using exercise number 12 of the collection, which describes the hospital's reception information system (the whole exercise with the diagrams can be found in the appendix B. The following results are discussed upon confronting the diagram made by the human and ChatGpt-4 and DeepSeek-V3 with the given solution.

4.3.1 Exercise Description and Evaluation

Therefore, the following is the description of the Exercise 12:

- *Hospital Management System is a large system including several subsystems or modules providing a variety of functions. Purpose: Describe major services (functionality) provided by a hospital's reception. The Hospital Reception subsystem or module supports some of the many job duties of hospital receptionists. The receptionist schedules patient's appointments and admission to the hospital, collects information from patient upon patient's arrival and/or by phone. For the patient that will stay in the hospital ("inpatient") she or he should have a bed allotted in a ward. Receptionists might also receive patient's payments, record them in a database and provide receipts, file insurance claims and medical reports.*

Reference Elements for Exercise 12

The reference solution will serve as the ground truth for scoring completeness, redundancy, and syntactic, semantic, and pragmatic quality in the evaluation part.

Actors

- The exercise presents just one actor, which is the "Receptionist"

Use Cases

- The exercise given solution has a total of nine use cases:

    1. Patient Registration
    2. Schedule Patient Appointment
    3. Schedule Patient Hospital Admission
    4. Patient Hospital Admission
    5. Outpatient Hospital Admission
    6. Inpatient Hospital Admission
    7. Bed Allotment
    8. File Insurance Forms/Claims
    9. File Medical Reports

Relationships

The type of relationships present in the given solution are as follows:

- Associations:
  - Receptionist –> All nine use cases.
- Include:
  - Patient Hospital Admission **includes** Patient Registration
  - Inpatient Hospital Admission **includes** Bed Allotment
- Exclude:
  - Patient Registration **extended by** Schedule Patient Appointment
  - Patient Registration **extended by** Schedule Patient Hospital Admission
- Generalization:
  - Outpatient Hospital Admission and Inpatient Hospital Admission are **generalizations** of Patient Hospital Admission.

The table 4.7 shows a summary of all the elements used for the evaluation.

Category	Elements Counted
Actors	1
Use Cases	9
Associations	6
Includes	2
Extends	2
Generalizations	2

Table 4.7. Elements Counted by Category

Human Created Diagram Errors

- *Syntactic errors*: No syntax errors were found during the evaluation breakdown.
- *Semantic Errors.* During the evaluation, four semantic errors were found:
  1. Wrong Mapping "*Collect patient information*" is not exactly the same as "*Patient registration*".

2. Missing Correct Extensions "*Schedule patient appointment*" and "*Schedule patient admission*" should extend "*Patient Registration*".
3. Unnecessary Use Case "*Record payments in DB*" use case is not part of the reference solution.
4. Missing Correct Inclusions: " Patient Admission " should include " Patient Admission ".

- *Pragmatic errors.* One pragmatic error was found during the assessment:

1. Unused Elements: Introduction of an action that is redundant with respect to the domain, which is " Record payments in the DB ".

Quality Dimension	**Error Count**	**Notes**
Syntactic	0	None syntactic errors were found
Semantic	4	Incorrect and extra use case, incorrect behavior links
Pragmatic	1	Extra domain actions

Table 4.8. Error Count Across Quality Dimensions

Table 4.8 shows a final summary of the number of errors obtained from the human-generated diagram based on the rules mentioned in Chapter 3.

ChatGpt-4 Created Diagram Errors

- *Syntactic errors*: During the evaluation one errors were found:

1. Actor not placed inside the system boundaries

- *Semantic Errors.* During the evaluation, eight semantic errors were found:

1. Extra actor "*Patient*" not required from the textual scenario.
2. Extra actor "*Insurance Company*" not required from the textual scenario.
3. Extra actor "*Management System*" not required from the textual scenario.
4. Missing Essential Use Case "*Inpatient Admission*" that is described in the textual scenario.
5. Missing Essential Use Case "*Patient Admission*" that is described in the textual scenario.
6. Unnecessary Use Case "*Receive Payment*" use case is not part of the reference solution.

7. Unnecessary Use Case "*Provide Receipt*" use case is not part of the reference solution.
8. Unnecessary Use Case "*Record Payment*" use case is not part of the reference solution.

- *Pragmatic errors.* Two pragmatic errors were found during the evaluation:

1. Unused Elements: Introduction of unnecessary complexity with respect to the domain, which includes three extra actors and three use cases.
2. Impact on the understandability: Too many lines crossed and actors not aligned

Table 4.9 shows a final summary of the number of errors obtained from the ChatGPT-4 generated diagram based on the rules for the quality metrics.

Quality Dimension	**Error Count**	**Notes**
Syntactic	1	Incorrect placement of actors
Semantic	8	Incorrect and extra use cases and actors
Pragmatic	2	Extra domain actions and not easy to read diagram

Table 4.9. Error Count Across Quality Dimensions

DeepSeek-V3 Created Diagram Errors

- *Syntactic errors*: During the evaluation no errors were found.
- *Semantic Errors.* During the evaluation, nine semantic errors were found:

1. Extra actor "*Patient*" not required from the textual scenario.
2. Extra actor "*Insurance Company*" not required from the textual scenario.
3. Extra actor "*Management System*" not required from the textual scenario.
4. Missing Essential Use Case "*Inpatient Admission*" that is described in the textual scenario.
5. Missing Essential Use Case "*Patient Admission*" that is described in the textual scenario.
6. Unnecessary Use Case "*Receive Payment*" use case is not part of the reference solution.
7. Unnecessary Use Case "*Provide Receipt*" use case is not part of the reference solution.

8. Unnecessary Use Case "*Record Payment*" use case is not part of the reference solution.
9. Incorrect relationship direction"*Generate Medical Report* should not extend "*Admit Patient*"

- *Pragmatic errors.* During the evaluation, one pragmatic error was found:

1. Unused Elements: Introduction of unnecessary complexity with respect to the domain, which includes three extra actors and three use cases.

Table 4.10 shows a final summary of the number of errors obtained from the ChatGPT-4 generated diagram based on the rules for the quality metrics.

Quality Dimension	**Error Count**	**Notes**
Syntactic	0	None
Semantic	9	Incorrect use of extend and extra use cases and actors
Pragmatic	1	Extra domain actions

Table 4.10. Error Count Across Quality Dimensions

Completeness and Redundancy Rates for the Different Solutions

To evaluate the quantitative value of UML use case diagrams, we used two key measures: completeness, which checks how many expected elements (Formula 4.1) are properly included, and redundancy (Formula 4.2), which captures the proportion of unnecessary elements added. These measures provide a clear, quantitative basis for comparing human-generated and LLM-generated diagrams with reference solutions.

$$\text{Completeness Rate} = \frac{\text{Correct Actors} + \text{Correct Use Cases} + \text{Correct Relationships}}{\text{Total Elements in the Reference Solution}} \tag{4.1}$$

$$\text{Redundancy Rate} = \frac{\text{Unmatched Actors} + \text{Unmatched Use Cases}}{\text{Total Elements in the Generated Diagram}} \tag{4.2}$$

Given the complete data of the exercise 12, the results obtained for the human and LLMs solutions are as follows:

- Human Diagram:

$$\text{Completeness Rate} = \frac{1+9+1}{14} = 0.79(79\%) \tag{4.3}$$

$$\text{Redundancy Rate} = \frac{0+1}{12} = 0.08(8\%) \tag{4.4}$$

- ChatGpt-4:

$$\text{Completeness Rate} = \frac{1+6+0}{14} = 0.5(50\%) \tag{4.5}$$

$$\text{Redundancy Rate} = \frac{3+5}{15} = 0.53(53\%) \tag{4.6}$$

- DeepSeek-V3

$$\text{Completeness Rate} = \frac{1+5+0}{14} = 0.43(43\%) \tag{4.7}$$

$$\text{Redundancy Rate} = \frac{3+2}{11} = 0.45(45\%) \tag{4.8}$$

Therefore, this section presents the application of these measures to Exercise 12, comparing the results from three different sources: a human player, ChatGPT-4, and DeepSeek-V3. For instance, in Exercise 12, the human diagram achieved the highest completeness (0.79) and lowest redundancy (0.08). Instead, both LLM models demonstrated lower completeness and higher redundancy, with ChatGPT-4 achieving a completeness of 0.50 and a redundancy of 0.53, and DeepSeek-V3 scoring 0.43 and 0.45, respectively.

4.3.2 Numbers Analysis of the Exercise

The numerical results of exercise 12 reflect a direct relationship between the types of modeling errors (syntactic, semantic, and pragmatic) and the resulting completeness and redundancy rates. Diagrams with higher semantic and pragmatic errors consistently revealed lower completeness and higher redundancy, while syntactic accuracy, even if important, seemed less influential from a numerical point of view due to its relatively low incidence. By analyzing the numbers, we obtained that:

- The human-generated diagram showed no syntactic errors, a moderate level of semantic mistakes (4), mostly due to incorrect mappings and missing extension relationships, and one pragmatic issue, tied to the addition of an unnecessary action. These issues slightly impacted its completeness, 79%, and introduced a small degree of redundancy, 8%. Despite some omissions, the human solution maintained strong adherence to the domain intent and showed a high level of abstraction and clarity.

- In contrast, the ChatGPT-4 solution had one syntactic error (actor placement), eight semantic errors, and two pragmatic errors. These errors directly affected its completeness rate, which dropped to 50% due to the lack of essential use cases and the misuse of domain concepts. Furthermore, the presence of 3 additional actors and three redundant use cases led to a high redundancy of 53%, revealing excessive and unnecessary modeling.

- The DeepSeek-V3 diagram did not contain any syntactic violations, but it accumulated nine semantic faults and one pragmatic error. Similar to ChatGPT-4, it failed to model fundamental concepts and introduced several irrelevant components, resulting in the lowest completeness score of 43% and a high redundancy of 45%. Specifically, the diagram misrepresented relationships and added extraneous actors and use cases that deviated from the functional core of the scenario.

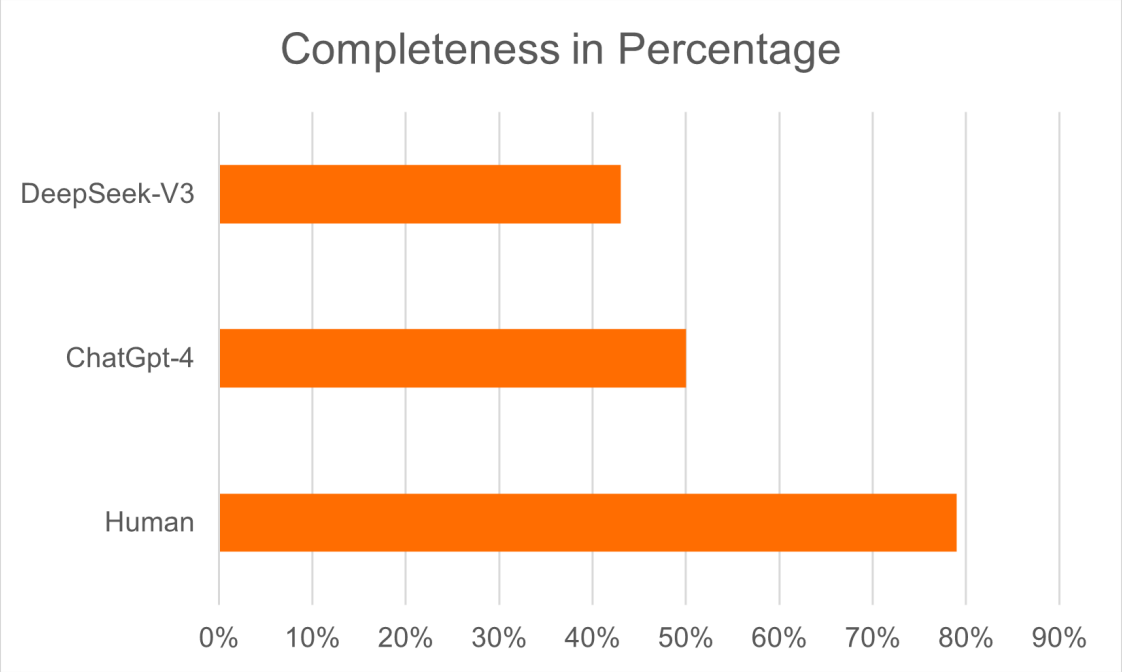

Figure 4.7. Completeness Percentage

These results obviously illustrate the correlation between qualitative errors in diagrams and metric performance:

- Semantic errors reduce completeness, since such errors typically involve missing or incorrectly represented use cases, actors, and relationships.

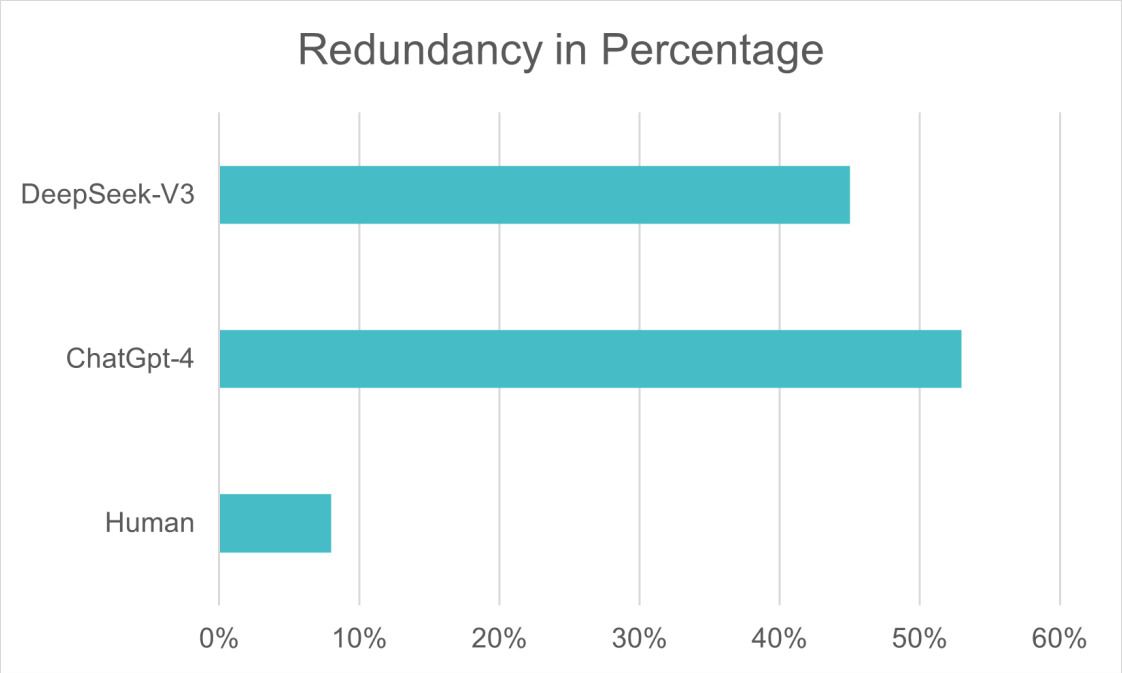

Figure 4.8. Redundancy Percentage

- Pragmatic errors increase redundancy, due to their adding of unneeded elements that reduce clarity and accuracy.
- Syntactic errors, even if less frequent, contribute to structural inaccuracies, but in this instance, had a minimal numerical impact.

In summary, the human solution demonstrated the best balance between accuracy and abstraction. LLMs, while capable of generating syntactically correct diagrams, showed significant limitations in interpreting domain intent and maintaining modeling discipline, causing reduced completeness and increased redundancy.

From the two figures 4.7 and 4.8, we can see how the percentages of each solution are distributed.

4.4 Conclusion

This chapter provided a detailed evaluation of use case diagrams (UCDs) generated by humans and large language models (LLMs), particularly ChatGPT-4 and DeepSeek-V3. The analysis was carried out in both quantitative (completeness and redundancy rates) and qualitative (syntactic, semantic, and pragmatic) terms, offering a panoramic view of

the validity of the model. LLMs showed promising results, especially in terms of syntactic correctness. Despite this, they had considerable difficulty with semantic interpretation and pragmatic clarity. This often led to missing essential elements, embedding irrelevant content, and producing structurally correct but conceptually imperfect models. In particular, ChatGPT-4 and DeepSeek-V3 revealed lower completeness rates and greater redundancy than human reference. The in-depth case study in Exercise 12 reinforced the overall results. It was illustrated how semantic and pragmatic barriers directly affect quantitative metrics. Although LLMs sometimes produced diagrams with full syntactic validity, their failure to distinguish essential from non-essential information reduced their overall effectiveness in modeling. In a nutshell, LLMs show potential for assisting in UCD generation, particularly in educational and prototyping contexts. However, their current weaknesses in terms of semantic precision and abstraction highlight the continued importance of human expertise in modeling activities. These findings provide a basis for future improvements, both in prompt engineering strategies and in the design of hybrid validation frameworks that combine the efficiency of LLMs with human supervision.

Chapter 5

Conclusions and future developments

This thesis examined the efficiency of large language models (LLMs), in particular ChatGPT-4 and DeepSeek-V3, for generating Unified Modeling Language (UML) compared to manually made diagrams. The analytical review used a methodology that included both quantitative and qualitative assessments. Hence, this allowed the study of relevant rates, such as completeness and redundancy, and the evaluation of the syntactic, semantic, and pragmatic qualities of the diagrams.

The research found that LLMs are capable of generating UML class diagrams with syntactic correctness almost same to that of humans, even though there are a difference in semantic and pragmatic correctness. The challenges that LLMs face in representing the complex semantic connections and pragmatic clarity necessary to accurately create use case diagrams are highlighted. These results suggest that further developments in AI technology are needed to fully automate diagram creation.

LLMs showed notable potential, particularly in facilitating quick prototyping, as well as supporting iterative design processes for Use Case diagram. Instead, from a theoretical point of view, it lays the foundation for further advances in AI-based design techniques, highlighting the need for more sophisticated models with a deeper understanding of semantic connections and pragmatic clarity. However, their strengths in syntactic accuracy suggest they could be also effectively integrated into educational settings as tools for initial diagram generation, allowing students and educators to focus on refining semantic and pragmatic aspects.

Future Developments

The work of this thesis was restricted on assessing the use case diagram exercises solutions. Further research should focus on the efficiency of Large Language Models in creating class, sequence, distribution, and state diagrams. Some research has already explored image-to-UML and multi-diagram evaluations, offering insights for expanding the evaluation framework beyond UCDs [28]. Moreover, with the advances in AI technology, assessing enhanced and recent language models could show improvements in semantic understanding and precision in the production of UML diagrams, mitigating the issues found in the results of this study.

The new developments in artificial intelligence and machine learning suggest a transformative impact on software design, for instance the generation of UML use case diagrams. Advances in LLM architectures and natural language understanding, such as in LLaMA models, are bridging the semantic gap between textual descriptions and formal representations of diagrams. Thus, as these models evolve, we can expect more rigorous, context-sensitive, and dynamic UML generation capabilities that can adapt to ongoing project changes and development activities. This could lead to real-time diagram creation, updates, and intelligent design suggestions seamlessly integrated into software development environments [29].

Current models, such as GPT-4 and DeepSeek-V3, are unable to autonomously detect and correct constraint violations, which contributes to the correctness gaps observed during the solutions of the UML diagram exercises. Upcoming study could explore metrics such as the number of constraints, prompt length, and the types of errors to more systematically measure the complexity of the prompt. An important direction for future research concerns the integration of feedback loops and self-correction mechanisms into LLM-based UML diagram generation [30]. The semantic and pragmatic gap found in this thesis can be resolved with the help of this self-correction method. Future LLMs could be trained to create UML diagrams and then modify them iteratively based on semantic input, using iterative comments and corrections approach.

Final Reflection

Steady research and development in this field can fill existing gaps, improve AI capabilities in software design, and ultimately transform how software modeling is taught and practiced. The large-scale integration of large language models has excellent potential for long-term effects on software design methods. Thanks to their ability to understand, model, and produce complex software design objects, large language models could simplify the software development lifecycle as they grow.

In addition to increasing efficiency, this integration will make software design more accessible to everyone, making complex design assignments more accessible to those with less technical knowledge. Therefore, there is great room for improvement for the results obtained.

Appendix A

Collection of Exercises and Calculation

The collection with the twenty-three exercises (for each exercise is given the text, given solution, human solution, ChatGpt-4 solution, DeepSeek-V3 solution, JSON format exercises codes, and the Excel with all the calculations can be found in the following link: Exercises Thesis Dataset

Appendix B

Exercise 12

EXERCISE 12 https://www.uml-diagrams.org/examples/hospital-management-use-case-diagram-example.html?context=uc-examples

Description:

Hospital Management System is a large system including several subsystems or modules providing a variety of functions. Purpose: Describe major services (functionality) provided by a hospital's reception. The Hospital Reception subsystem or module supports some of the many job duties of hospital receptionists. Receptionist schedules patient's appointments and admission to the hospital, collects information from patient upon patient's arrival and/or by phone. For the patient that will stay in the hospital ("inpatient") she or he should have a bed allotted in a ward. Receptionists might also receive patient's payments, record them in a database and provide receipts, file insurance claims and medical reports.

Figure B.1. Exercise 12 Text Description

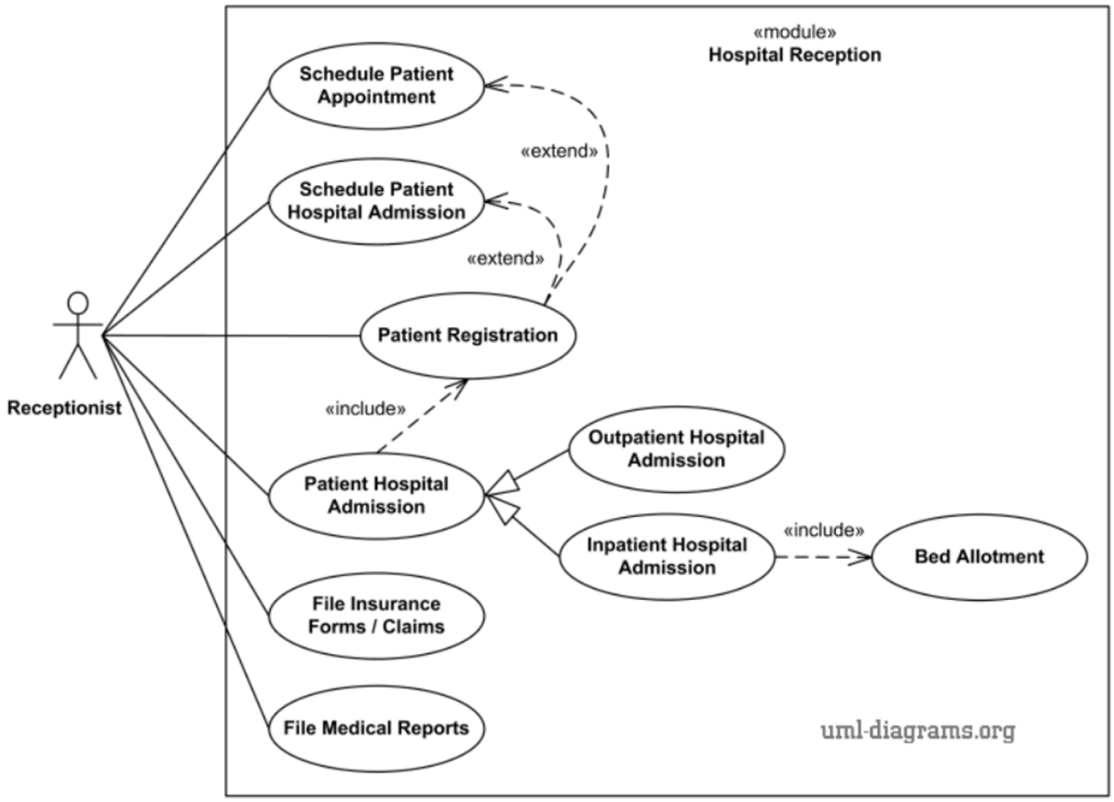

Figure B.2. Exercise 12 Reference Solution

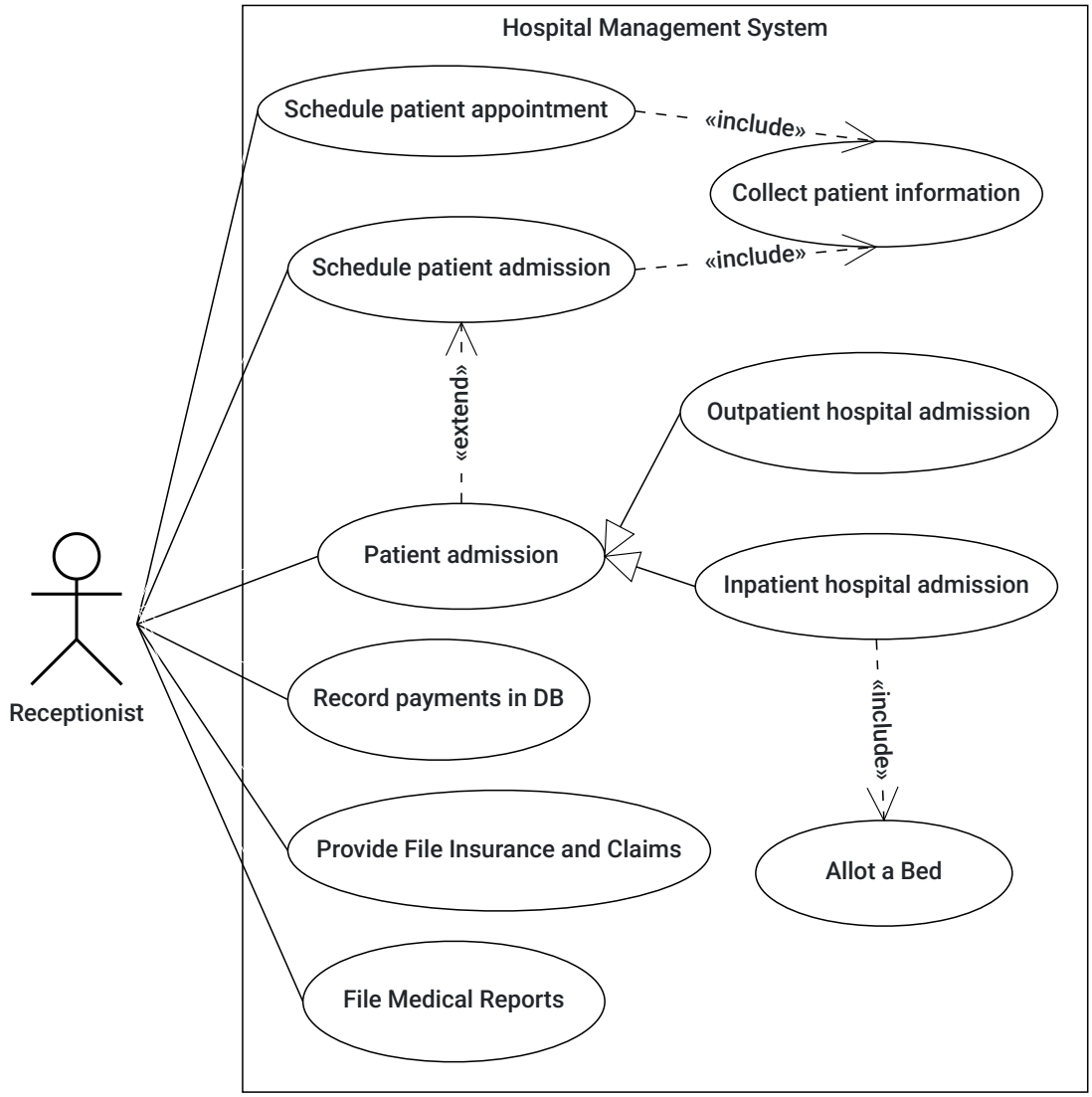

Figure B.3. Exercise 12 Human Solution

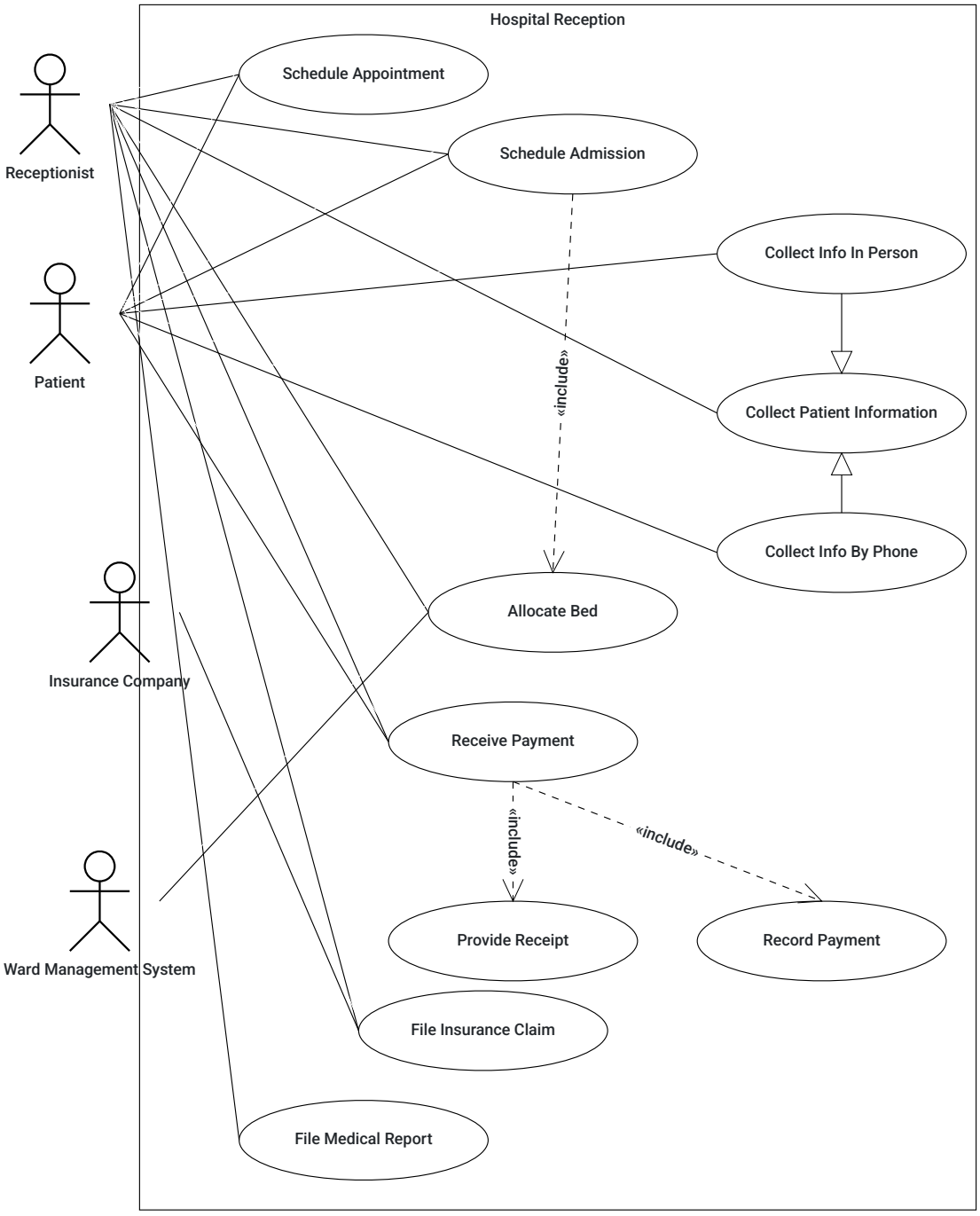

Figure B.4. Exercise 12 ChatGpt-4 Solution

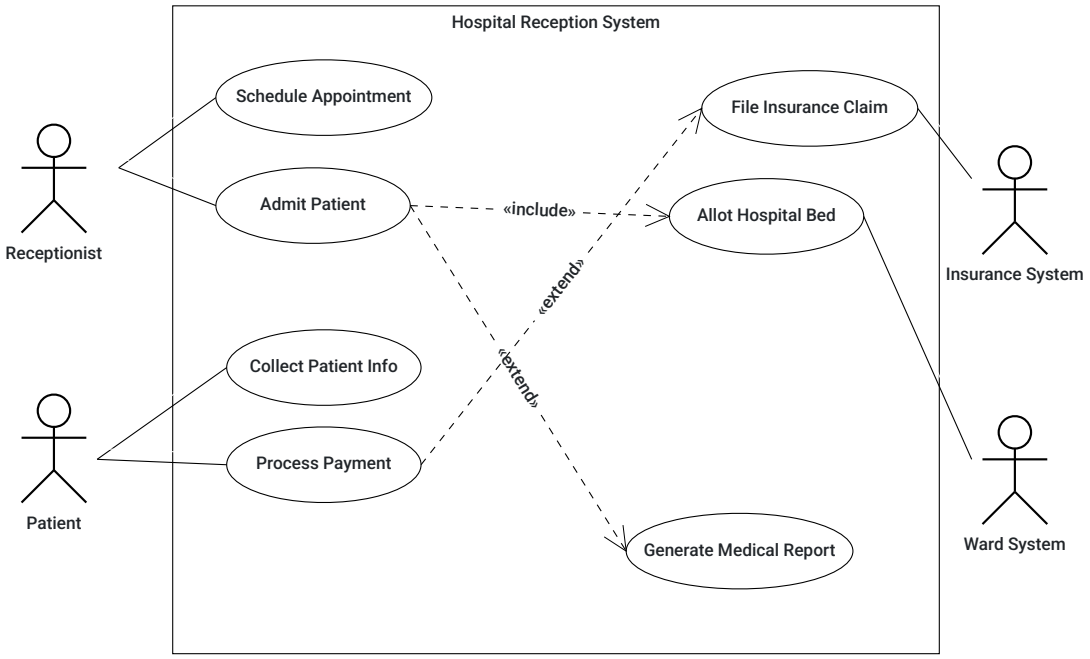

Figure B.5. Exercise 12 DeepSeek-V3 Solution

Appendix C

JSON format

```
{"id": "05678086-30e5-4d79-a378-7f6b3fd8ed8a",
  "title": "Esercizio 18",
  "model": {
    "version": "3.0.0",
    "type": "UseCaseDiagram",
    "size": {
      "width": 780,
      "height": 480
    },
    "interactive": {
      "elements": {},
      "relationships": {}
    },
    "elements": {
      "1092f089-8416-4896-bd5f-eb0bb3fd2e26": {
        "id": "1092f089-8416-4896-bd5f-eb0bb3fd2e26",
        "name": "Quiz System",
        "type": "UseCaseSystem",
        "owner": null,
        "bounds": {
          "x": -240,
          "y": -220,
          "width": 430,
          "height": 430
        }
      },
      "5dd68caa-21f1-4d3b-8864-74753f24f3d0":..
```

Listing C.1. First part of the JSON code for the Exercise 18

Bibliography

[1] Bouali, N., Gerhold, M., Ul Rehman, T., & Ahmed, F. Toward automated UML diagram assessment: Comparing LLM-generated scores with teaching assistants. In *Proceedings of the 17th International Conference on Computer Supported Education (CSEDU 2025)*, pp. 158–169. SCITEPRESS.

[2] Khan, J. A., Qayyum, S., & Dar, H. S. Large language model for requirements engineering: A systematic literature review. *ResearchGate Preprint*, 2024.

[3] Ludewig, J. Models in software engineering : an introduction. *Software and Systems Modeling, 2(1), 5-14. Springer-Verlag*, 2003.

[4] Garaccione, G., Vega Carrazan, P. F., Coppola, R., & Ardito, L. Evaluating large language models in exercises of UML use case diagrams modeling. *Conference paper*, 2024.

[5] Garaccione, G., Coppola, R., Ardito, L., & Torchiano, M. Gamification of conceptual modeling education: an analysis of productivity and students' perception. *Software Quality Journal, 33(3), 1-19. Springer*, 2025.

[6] Burkin, V. Mitigating risks in software development through effective requirements engineering. *Unpublished manuscript, University of Illinois Urbana-Champaign*, 2023.

[7] Vasques, D. G., Galindo, J. F., dos Santos, G. S., Gomes, F. D., Garcia-Nunes, P. I., & Martins, P. S. An educational process for requirements extraction and use case modeling based on problem-based learning and knowledge acquisition. *In Proceedings of the XV Brazilian Symposium on Information Systems (SBSI'19), Aracaju, Brazil, pp. 1-8. ACM*, 2019.

[8] Beimel, D. , & Kedmi-Shahar, E. Improving the identification of functional system requirements when novice analysts create use case diagrams: the benefits of applying conceptual mental models. *In Requirements Engineering, vol. 24, pp. 483-502.* Springer, 2019.

[9] Arifin, M. N., & Siahaan, D. Structural and Semantic Similarity Measurement of UML Use Case Diagram. *In Lontar Komputer, vol. 11, no. 2, pp. 88-100.* Universitas Udayana, 2020.

[10] Madanayake, R. S., Dias, G. K. A., & Kodikara, N. D. Transforming simplified requirement into a UML use case diagram using an open source tool. *In International Journal of Computer Science and Software Engineering (IJCSSE), vol. 6, no. 3, pp. 61-70.* University of Colombo, March 2017.

[11] Elallaoui, M., Nafil, K., & Touahni, R. Automatic transformation of user stories into UML use case diagrams using NLP techniques. *In Proceedings of the 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2018), Porto, Portugal, Procedia Computer Science, vol. 130, pp. 42-49.* Elsevier, 2018.

[12] John, B. A study of UML diagrams in software development. *In Proceedings of the International Conference on Software Engineering Research and Development (SERD 2025), Faridabad, India, pp. 1-8.* ResearchGate, 2025.

[13] Speth, S., Meibner, N., & Becker, S. ChatGPT's aptitude in utilizing UML diagrams for software engineering exercise generation. *36th International Conference on Software Engineering Education and Training (CSEE&T 2024)*, IEEE.

[14] Clusmann, J., Kolbinger, F. R., Muti, H. S., Carrero, Z. I., Eckardt, J.-N., Ghaffari Laleh, N., Loffler, C. M. L., Schwarzkopf, S.-C., Unger, M., Veldhuizen, G. P., Wagner, S. J., & Kather, J. N. The future landscape of large language models in medicine. *Communications Medicine, 3, 141*, 2023.

[15] Hadi, M. U., Al-Tashi, Q., Qureshi, R., Shah, A., Muneer, A., Irfan, M., Zafar, A., Shaikh, M. B., Akhtar, N., Al-Garadi, M. A., Hassan, S. Z., Shoman, M., Wu, J., Mirjalili, S., & Shah, M. Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects. *Preprint,* September 2024. In TechRxiv, DOI: 10.36227/techrxiv.23589741.v7.

[16] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. Attention is all you need. *In Proceedings of the 30th Conference on Neural Information Processing Systems (NeurIPS'17),* 2017.

[17] Veeramachaneni, V. Large Language Models: A comprehensive survey on architectures, applications, and challenges. *In Advanced Innovations in Computer Programming Languages, 7(1), pp. 20-39. HBRP Publications,* 2025.

[18] Karabacak, M., & Margetis, K. Embracing Large Language Models for Medical Applications: Opportunities and Challenges. *Cureus, 15(5), e39305*, 2023.

[19] Wang, C., Wang, B., Liang, P., & Liang, J. Assessing UML models by ChatGPT: Implications for education. *arXiv*, 2024.

[20] Hagendorff, T., & Danks, D. Ethical and Methodological Challenges in Building Morally Informed AI Systems. *AI and Ethics, 3(2), pp. 553-566.* 2023.

[21] Wang, B., Wang, C., Liang, P., Li, B., & Zeng, C. How LLMs aid in UML modelling: An exploratory study with novice analysts. *IEEE International Conference on*

Software Services Engineering (SSE 2024).

[22] Pornprasit, C., & Tantithamthavorn, C. Fine-tuning and prompt engineering for large language models-based code review automation. *Information and Software Technology, 175, 107523.* 2024.

[23] Babaalla, Z., Oualla, M., Bouziane, E. M., & Jakimi, A. From text-based system specifications to UML diagrams: A bridge between words and models. *In Proceedings of the 2024 International Conference on Circuits, Systems and Communication (ICCSC), Errachidia, Morocco, pp. 1-8. IEEE,* 2024.

[24] Pereira, J., Lopez, J.-M., Garmendia, X., & Azanza, M. Leveraging Open Source LLMs for Software Engineering Education and Training. *In Proceedings of the 2024 36th International Conference on Software Engineering Education and Training (CSEE&T), Donostia, Spain, pp. 1-9. IEEE, 2024.*

[25] Nasiri, S., Rhazali, Y., Lahmer, M., & Adadi, A. From User Stories to UML Diagrams Driven by Ontological and Production Model. *International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 12, No. 6, pp. 333-340,* 2021.

[26] Shin, J., Tang, C., Mohati, T., Nayebi, M., Wang, S., & Hemmati, H. Prompt engineering or fine-tuning: An empirical assessment of LLMs for code. *In Proceedings of the Conference. arXiv,* 2025.

[27] De Bari, D., Garaccione, G., Coppola, R., Ardito, L., & Torchiano, M. Evaluating large language models in exercises of UML class diagram modeling. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2024)*, Barcelona, Spain.

[28] Conrardy, A., & Cabot, J. From image to UML: First results of image-based UML diagram generation using LLMs. *First Large Language Models for Model-Driven Engineering Workshop (LLM4MDE 2024)*, Enschede, Netherlands.

[29] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., RoziÃ¨re, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. LLaMA: Open and Efficient Foundation Language Models. *In arXiv preprint arXiv:2302.13971,* 2023.

[30] Al-Ahmad, B., Alsobeh, A., Meqdadi, O., Shaikh, N. A Student-Centric Evaluation Survey to Explore the *Impact of LLMs on UML Modeling. Information 2025, 16(7), 565, pp. 1-34. MDPI:* Basel, Switzerland, 2025.