POLITECNICO DI TORINO

Master's degree in Engineering and Management

Master's Degree Thesis

Evaluating Large Language Models in Software Design: A Comparative Analysis of UML Class Diagram Generation

Supervisors

Prof. Riccardo COPPOLA

Doct. Giacomo GARACCIONE

Candidate

Daniele DE BARI

Academic Year 2023-2024

Summary

This master's thesis aims to assess the feasibility of utilizing Large Language Models (LLMs) to generate Unified Modeling Language (UML) Class Diagrams. UML is a standardized visual language widely used in software engineering to depict the structure and design of a software system, enabling clear communication and documentation of system components and their relationships. By comparing the LLM-generated diagrams with those produced by humans, particularly during the crucial initial phase of requirement gathering, the research assesses whether AI can support or enhance traditional software modeling practices. The study utilizes a two-part methodology complemented by a statistical analysis. In the first part, the diagrams are examined for syntactic (adherence to UML rules), semantic (accuracy of meaning and concepts), and pragmatic (usefulness and applicability) quality errors, in the second part the semantic distance between the generated diagrams and the given solutions is calculated by using a specific algorithm.

The findings suggest that LLMs are capable of generating UML class diagrams with a level of syntactic and pragmatic quality comparable to that of human-produced diagrams, due to no statistically significant differences in these areas. However, the analysis also uncovers a noticeable gap in semantic quality, where human-generated diagrams outperform those generated by LLMs, highlighting the current limitation of LLMs in understanding the semantic nuances required to generate an accurate diagram.

Moreover, the study finds that LLM-produced diagrams are typically more distant from the reference solution, emphasizing the challenges that LLMs face in domain-specific knowledge that human experts are able to bring in the creation of the diagram.

With those findings the thesis contributes to the ongoing discussion on the role of AI in software engineering, showing that, while LLMs show promise in certain aspects of UML class diagram generation, there is a gap in semantic accuracy, underlining the necessity of further advancements in AI capabilities to reach the same quality level of human-generated diagrams.

Contents

List of Tables 7

List of Figures 9

1 Introduction 11

2 Background 13

2.1 Introduction to Modelling 13
2.1.1 Introduction to the importance of modeling 13
2.1.2 Software Modeling Phases 13
2.1.3 Requirements Gathering 14
2.1.4 Relationship between modeling and requirements gathering 16
2.2 UML Class Diagram 17
2.2.1 Introduction 17
2.2.2 Benefits for developers and stakeholder 18
2.2.3 Core components of UML class diagrams 18
2.2.4 Creating UML Class Diagrams 20
2.2.5 Evolution of UML Modelling Tools 21
2.2.6 Challenges and Limitation of UML class diagrams . . . 22
2.3 Large Language Models 23
2.3.1 Introduction 23
2.3.2 Definition and Core Concepts 23
2.3.3 Evolution of Language Models 25
2.3.4 Architecture and Mechanics 26
2.3.5 Applications 27
2.3.6 Challenges and Ethical Considerations 27
2.4 Application of Large Language Models for Requirement Gathering: Literature Review 29

2.4.1 Introduction . . . 29
2.4.2 AutoScrum: Automating Project Planning Using Large Language Models (2023) . . . 30
2.4.3 An Analysis on Large Language Models in Healthcare: A Case Study of BioBERT (2023) . . . 30
2.4.4 From User Stories to UML Diagrams Driven by Ontological and Production Model (2021) . . . 32

3 Methodology 33
3.1 Possible evaluation Criteria . . . 33
3.1.1 Introduction . . . 33
3.1.2 Overview of Articles and Methods . . . 33
3.1.3 Analysis of Proposed Methods . . . 37
3.2 Approach to Evaluate Diagrams . . . 38
3.2.1 First Part . . . 39
3.2.2 Second part . . . 39
3.3 Collection of Exercises . . . 44
3.3.1 Different Levels of Difficulty . . . 45
3.3.2 Given Solutions . . . 46
3.3.3 Example of Exercise . . . 46
3.4 Diagrams made by Human . . . 46
3.4.1 Tool Used . . . 46
3.4.2 Established Rules . . . 48
3.4.3 Limitation . . . 49
3.5 Diagrams made by LLM . . . 49
3.5.1 Chosen LLM . . . 49
3.5.2 Prompt . . . 50
3.5.3 PlantUML . . . 51
3.6 Statistical Analysis . . . 51
3.6.1 T-Test . . . 53
3.6.2 Linear Regression . . . 54

4 Results 57
4.1 First Part . . . 57
4.1.1 Example . . . 57
4.1.2 Results first part . . . 58
4.2 Second Part . . . 62
4.2.1 Example . . . 62
4.2.2 Results second part . . . 65

4.3 Statistical Analysis . . . 68
4.3.1 T-Test . . . 68
4.3.2 Linear Regression . . . 70
4.3.3 Conclusion . . . 77

5 Conclusions and future developments 79

A Collection of Exercises and Calculation 81

B Exercise 10 83

C T-Test 87
C.1 First Method . . . 87
C.1.1 Syntactic Quality . . . 87
C.1.2 Semantic Quality . . . 88
C.1.3 Pragmatic Quality . . . 89
C.2 Second Method . . . 91

D Linear Regression 93
D.1 First Part . . . 93
D.2 Second Part . . . 101

Bibliography 105

List of Tables

3.1 Errors in various quality categories [1] 40
3.2 Example of Distribution of various type of errors [1] 40
3.3 Class and Interface Distances[2] 41
3.4 Elements of Comparison Vector for Class Attributes[2] 42
3.5 Elements of Comparison Vector for Class Methods[2] 43
3.6 Relation Comparison Vector Elements[2] 44
4.1 Each row represents a different exercise and for each of them are reported the number of errors both for Human and LLM for all the three types of quality. 59
4.2 Sums of errors for each Quality 60
4.3 Distances for each element in Exercise 10 63
4.4 Total Distances Exercise 10 65
4.5 Distances from the given solutions, and differences 66
4.6 Average of the distances and of the differences 66
4.7 Results Paired Samples T-Test. 68
4.8 Data from the given solutions. 70
4.9 Estimated Difficulties. 71
4.10 P-values Human Syntactic Errors 73
4.11 P-values LLM Syntactic Errors 73
4.12 P-values Human Semantic Errors 74
4.13 P-values LLM Semantic Errors 75
4.14 P-values Human Pragmatic Errors 75
4.15 P-Values LLM Pragmatic Errors 76
4.16 P-Values Human Distances 77
4.17 P-Value LLM Distances 77
C.1 Descriptives 87
C.2 Paired Samples T-Test 87
C.3 Descriptives 88
C.4 Paired Samples T-Test 89
C.5 Descriptives 90

C.6 Paired Samples T-Test 90
C.7 Descriptives 91
C.8 Paired Samples T-Test 92
D.1 Model Summary - Human Syntactic Errors 93
D.2 Coefficients 93
D.3 Model Summary - LLM Syntactic Errors 94
D.4 Coefficients 95
D.5 Model Summary - Human Semantic Errors 96
D.6 Coefficients 96
D.7 Model Summary - LLM Semantic Errors 97
D.8 Coefficients 97
D.9 Model Summary - Human Pragmatic Errors 98
D.10 Coefficients 99
D.11 Model Summary - LLM Pragmatic Errors 100
D.12 Coefficients 100
D.13 Model Summary - Distance Human 101
D.14 Coefficients 101
D.15 Model Summary - Distance LLM 102
D.16 Coefficients 103

List of Figures

2.1 Example of a Class Diagram for a production work[3] 19
2.2 Overview of potential applications for LLMs in medicine [4] . . 24
2.3 Autoscrum process. The green icons represent parts of the process that can be augmented by the use of language models [5] 31
2.4 The shortcut planning approach [5] 31
2.5 Architecture of the Proposed Approach[6] 32
3.1 Example of a Graph Matching[7] 34
3.2 Text of the Exercise [8] 47
3.3 Given Solution [8] 48
3.4 Solution made by human using Microsoft Visio 50
3.5 Solution made by ChatGPT-4 in PlantUML 52
4.1 Number of errors divided in the three Qualities 61
4.2 Number of errors divided between Human and LLM 61
4.3 Bar chart of the distances. 67
4.4 Bar chart reporting the difference. 68
B.1 Text Exercise 10 83
B.2 Given-solution of exercise 10 84
B.3 Human solution of exercise 10 84
B.4 LLM solution of exercise 10 85
C.1 Bar-plot Syntactic Quality 88
C.2 Barplot Semantic Quality 89
C.3 Barplot Pragmatic Quality 91
C.4 Barplot for the distance to a reference solution 92
D.1 Residuals Vs. Human Syntactic Errors 94
D.2 Residuals Vs. LLM Syntactic Errors 95
D.3 Residuals Vs. Human Semantic Errors 96
D.4 Residuals Vs. LLM Semantic Errors 98
D.5 Residuals Vs. Human Pragmatic Errors 99

D.6 Residuals Vs. LLM Pragmatic Errors 100
D.7 Residuals Vs. Distance Human 102
D.8 Residuals Vs. Distance LLM 103

Chapter 1

Introduction

The software engineering landscape is constantly evolving, and the design phase is one of the most crucial in the lifecycle of software development because inside it the foundation of the software system is conceptualized, designed, and documented, setting the basis for the subsequent development process. At the center of this phase is the utilization of Unified Modeling Language (UML), a standardized visual language that has been largely adopted to make clear and concise documentation of software designs. In particular, UML class diagrams are an essential tool in defining the structure of the software system, offering a visual representation of classes, their attributes, methods, and the relationship between them. However, the realization of accurate and comprehensive class diagrams is 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.

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

The scope of this thesis is to bridge this gap by assessing the feasibility and effectiveness of using LLMs to generate UML class diagrams, aiming to evaluate whether LLMs can support or potentially enhance the traditional practices of UML diagram generation. The investigation is structured around seeking to compare the syntactic, semantic, and pragmatic qualities of LLM-generated diagrams with those produced by humans, and to assess LLM's ability to replicate domain-specific knowledge within these diagrams.

The methodology adopted in this study employs a two-part approach designed to provide a comprehensive evaluation of the diagrams generated by LLMs. The first part involves an examination of the diagrams' quality, focusing on their adherence to UML standards (syntactic quality), the accuracy and appropriateness of the depicted concepts and relationships (semantic quality), and their practical applicability and understandability (pragmatic quality). The second part of the methodology, by the use of a specific algorithm, calculates the semantic distance of diagrams generated by LLMs from a set of reference diagrams and it confronts it with the distance of Human-generated diagrams from the same set, offering an objective measure of the LLMs' ability to accurately reflect domain-specific knowledge.

This research, by showing the capabilities and limitations of LLMs in this context, contributes valuable insights into the potential integration of AI in software engineering practices. The findings could pave the way for more efficient, accurate, and automated design processes, potentially transforming the role of human designers and the overall efficiency of software development. Furthermore, by identifying areas where LLMs still lag behind, particularly in the understanding and replicating domain-specific knowledge, it highlights critical avenues for future advancements in AI, aiming to reduce the gap between AI-generated and human-generated diagrams.

This thesis is structured across different key sections: an extensive **Background** chapter lays the theoretical foundation and locates the study within the existing body of research. The **Methodology** chapter illustrated the comprehensive approach taken to evaluate the diagrams. The **Results** chapter presents the findings of this evaluation and a statistical analysis to find the elements that most impact the quality of the generated diagrams. The work culminates in a **Conclusion** that synthesizes the insight gained from the research and proposes directions for the future.

Chapter 2

Background

2.1 Introduction to Modelling

2.1.1 Introduction to the importance of modeling

In the software development field, modeling refers to the process where abstract representations of a system are created. It helps with understanding, visualizing, and communicating the system structure and behavior. It can be considered as a blueprint utilized to guide developers and stakeholders throughout the software development lifecycle.

Modeling plays a crucial role in the lifecycle of software development. Starting from the design phase, until the phases of deployment and management, it provides a structured approach that helps in the visualization and understanding of the complexities of a system. For example, in Cloud Service Brokerage (CSB), a new technology whose notoriety has increased thanks to cloud computing (this technology acts as an intermediary between cloud service consumers and providers, facilitating the selection, procurement, and management of services from multiple vendors), modeling is important across all the stages of software development. Recent research in CSB makes it evident that there is an imbalance in the software development stages, due to a major focus of the research on the design stage. This example shows the importance of modeling not only in the design but throughout the entire software development cycle.[9]

2.1.2 Software Modeling Phases

There are three different phases in Software Modeling:

The first phase is Conceptual modeling, which is defined as the process of developing high-level representations of a system to focus on the main concepts and relationships, not entering into the details of the implementation. It is the base for the successive modeling phases. Its primary objective is to capture the essence of the system and to ensure that all the stakeholders understand in the same way the scope and the functionalities of the system. As an example, in the database field, the design of traditional relational databases begins with conceptual modeling, utilizing the Unified Modeling Language (UML) or the Entity-Relationship Diagram (ERD) for visualizing the representation of data and relationships.

The second phase is Logical modeling, which is about the way in which the system will be realized, highlighting data flow, functionality, and the relationships between all the different components. It can be considered as a bridge between the conceptual model (that is more abstract) and the physical model. Application requirements and access patterns are often taken into account in this phase, to ensure the alignment of the system with the intended use. For example, in NoSQL databases like MongoDB, logical modeling is used to determine the structure of the data and how they are going to be accessed, by also considering performance, scalability, and flexibility.

The last phase is the physical modeling, which regards the actual implementation details of the system. In this phase, decisions are taken about the database design, infrastructure, storage mechanisms, and other technical aspects. Retaking databases as an example, this phase is about decisions on indexing, partitioning, and data distribution strategies. In NoSQL databases, physical modeling becomes complicated because those databases are too flexible. All the decisions taken will have an impact on the performance and scalability of the system.[10]

2.1.3 Requirements Gathering

Requirements gathering is defined as the process of identifying, collecting, and documenting the needs and requests of stakeholders for a project. It is utilized as the basis on which the software development process is built. By assuring that the requirements are properly defined, the alignment with the business needs of the software product is respected and the product will meet the expectations of the user. Requirements gathering is also important for its role in the risk mitigation for the development of the software. Requirements engineering should ensure that potential ambiguities are addressed at the beginning, in order to lead to the creation of a final software product that is secure and successful.[11]

There are different techniques adopted in the requirements-gathering process, for different contexts and types of projects. The most common are:

- Interviews: direct conversations with the stakeholders to understand their needs.
- Questionnaires: surveys that are distributed to the stakeholders to gain information about specific topics.
- Prototyping: Creating a preliminary version of the software to gather feedback and refine requirements.
- Observation: by watching the users in their natural environment, it is possible to gain knowledge about their tasks and challenges.
- Workshops: stakeholders and development teams meet in collaborative sessions to brainstorm and define the requirements.
- Use Cases: detailed description of the way in which the users are going to interact with the software.

A technique is chosen based on the nature of the project, the stakeholders involved, and the challenges encountered in the requirements-gathering process.

During the requirements gathering several challenges can emerge:

- Vague or Ambiguous Requirements: often stakeholders don't have a clear idea of what they want.
- Changing Requirements: during the progression of the project, stakeholders might change their minds leading to the emergence of new requirements.
- Conflicting Requirements: there may be conflicting needs or priorities between different stakeholders.
- Sometimes stakeholders might not even be aware of certain requirements until later stages.

Possible solutions to those challenges are:

- Regular Communication: regular communication with all the stakeholders must be assured to ensure that everything is going in the right way and to reduce ambiguities.

- Prioritization: it is necessary to ensure to address first the most critical requirements since not all of them can be implemented together.

- Documentation: it is necessary to document in a proper way all the requirements and any changes to them ensuring clarity, obtaining in this way a reference to utilize in the development process.

- "Stakeholder Engagement" or "Regular Stakeholder Reviews" to ensure that the development team and stakeholders are aligned throughout the process.[12]

2.1.4 Relationship between modeling and requirements gathering

In software development, modeling provides an abstract representation of the system, showing its main concepts and relationships. This is beneficial for the requirements gathering. The visual representation of the system allows designers, stakeholders, and development teams to explore the system even before it is fully designed. The objective is to ensure the alignment of the software product with the business needs to meet the expectations of its users. Modeling can also serve as a communication tool: the visual nature of models can help bridge the gap between technical and non-technical stakeholders, facilitating clearer understanding and discussions around requirements.

What follows is that there is a strong interaction between the process of modeling and requirements gathering. While the system is being modeled, new requirements might emerge, or the refinement of existing ones might be needed. It can also happen that, while requirements are collected and understood, the model has to be updated in order to reflect them in a more correct way, opening the possibility, for the software, to be continually improved in order to meet the users' needs. The iterative nature of the relationship between modeling and requirements gathering fits well with modern agile software development methodologies.

2.2 UML Class Diagram

2.2.1 Introduction

The Unified Modelling Language (UML) is a standardized visual language designed to model and document software systems. It was developed in the mid-1990s, and since then it has become one of the most used tools for software engineers, by offering a logical and understandable means to visualize, specify, construct, and document the components of software systems. UML is not only used for object-oriented design but also for the modeling and the functional aspects of systems. [13]

In the UML framework, class diagrams are really important. They are one of the most used UML diagrams, and they give a visual representation of the structure of a system, by showing the system's classes, their attributes, methods, and the relationship between them. They are used for:

- System visualization: they simplify, for developers and stakeholders, the understanding of the architecture of a system by showing a clear picture of its structure.

- Documentation: they are a documentation tool, providing a reference for the developer during the coding phase, ensuring consistency and alignment with the system's design.

- Design blueprint: used during the design phase, they highlight the system's architecture before the beginning of the coding, in order to ensure the alignment of the system's design with its intended functionality.

- Optimization: modern CASE (Computer-Aided Software Engineering) tools, which are advanced software solutions designed to assist in various stages of the software development lifecycle, utilize class diagrams for optimization. By employing design patterns (best practice templates for solving recurring design issues) and anti-patterns (commonly recurring solutions that are ineffective and detrimental to the overall design), class diagrams can be verified, optimized, and even transformed automatically in some instances.

Therefore, class diagrams not only help in the visualization of the system but also act as a bridge, ensuring that the system design aligns with its intended functionality. Class diagrams, while crucial, are just one of many UML diagrams that collectively ensure this alignment.[13]

2.2.2 Benefits for developers and stakeholder

Class Diagrams have several benefits for the developers and the stakeholders. Two main benefits are:

- Clear communication: Class diagrams act as a universal language, reducing the gap between technical and non-technical stakeholders. By providing a clear and concise visual representation they ensure that everyone is on the same page. Class diagrams can also help in identifying potential design issues or areas of improvement early in the development process, as visualizing the system can make these issues more apparent

- Quality assurance: the incorporation of security and other requirements into class diagrams ensures that these aspects are considered from the beginning, leading to a more secure and robust software product. For instance, class diagrams can also help ensure that the system respects certain design principles or patterns, which can further improve the quality and maintainability of the software.[14]

2.2.3 Core components of UML class diagrams

UML Class Diagrams are composed of the following elements:

- Classes: a class represents an object or a set of objects with a common structure and behavior. They are the building blocks of object-oriented software systems. They contain data and behavior, serving as the “blueprints” from which objects are created. They define the nature of the objects and determine how they will interact within the system.

- Attributes: attributes are the properties or variables of a class, They define the characteristics or state of an object. For example, if we consider a “Machine” class, attributes could include “machine ID” and “Destination”. They capture fundamental data about an object, in this way, the system can store, retrieve, and manipulate this data when it is needed.

- Methods (or Operations): methods define the functions that a class can perform, dictating the behavior of an object. For example, if we consider again a “Machine” class, it might have methods like “start()” and “stop()”. They enclose the behavior of objects, in order to ensure that they act in a consistent manner with their design and purpose.

- Relationships: relationships represent the connections between classes. By providing a structure to the system, they ensure that objects interact in a coherent and meaningful manner. Common relationships include:
  - Association: a bi-directional relationship between two classes.
  - Aggregation: represents a “whole-part” relationship, where one class is a component of another.
  - Composition: a stronger form of aggregation where the parts cannot exist without the whole. When the whole is destroyed or deleted, its parts are as well.
  - Inheritance (Generalization): represents an “is-a” relationship between a base class (parent) and a derived class (child).[3]
- Multiplicity: it denotes a possible range of instances that can participate in an association between two classes, by defining the minimum and maximum number of instances of one class that may be associated with a single instance of another class. For example, "0..1" indicates none or one, "1" means exactly one, and "*" or "0..*" means many or an unlimited number.

In Figure 2.1 it is shown an example of a class diagram.

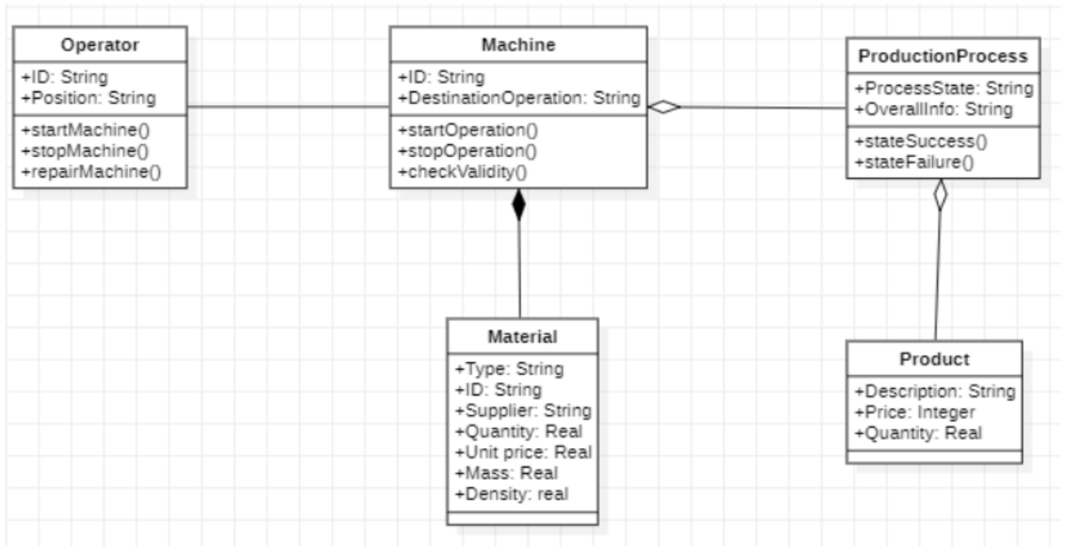


Figure 2.1: Example of a Class Diagram for a production work[3]

2.2.4 Creating UML Class Diagrams

The following steps are involved in the design of a class diagram:

- Requirement Analysis: it is necessary to understand the system's requirements before starting with the design, it is possible to do it by identifying the main entities or classes that will be part of the system.
- Identifying Relationships: determine how the classes will interact with each other by understanding the type of relationships.
- Determining the "Multiplicity" of relationships.
- Defining attribute and methods: for each class, it is necessary to list down the attributes (properties) and methods (functions) it will have.
- Drawing the diagram: start with classes, then add attributes and methods, and finally, represent the relationships.
- Review and Refine: once the initial diagram is ready, review it for accuracy and completeness. Make necessary refinements.

Some of the most popular tools and software used in the realization of class diagrams are IBM Rational Rose, Microsoft Visio, StarUML, and Lucidchart. These tools offer features like drag-and-drop elements, templates, and collaboration capabilities. There are also open-source options like "ArgoUML" or web-based tools like "draw.io".

In order to realize a comprehensive and effective class diagram it is suggested to comply with the following Best practices:

- Maintain clearness and conformity by using standard UML notations for the illustration of classes, relationships, etc.
- Aim for simplicity in the class diagram, and if it becomes too complicated, consider breaking it into smaller, more manageable segments.
- Choose descriptive and meaningful names for classes, attributes, and methods, to clearly describe their functionalities.
- Maintain a constant style in the diagram, including naming conventions, symbols, and layout, to improve readability and understanding.
- Add explanatory notes in order to explain complex sections or to provide additional information where necessary.

- Avoid Redundancy to ensure that the same information isn't represented in multiple places, which can lead to inconsistencies.[15]

2.2.5 Evolution of UML Modelling Tools

UML modeling tools evolution started from standalone tools and then they passed to repository-based model sharing, and web-based model sharing until they reached real-time model sharing. This evolution was caused by the increasing need for UML editors to support collaborations.

Repository-based model sharing is able to provide robust control of the versions, support concurrent model access with conflict resolution, ensure secure access control, and centralize model management. It is useful to maintain the historical integrity and security of UML models.

Web-Based Model sharing offers access to UML models that are platform-independent, through a browser. In this way it facilitates the use of a great number of stakeholders, improving collaboration by the use of discussion features. It is ideal for large teams distributed in different locations.

Real-time model sharing makes instantaneous collaboration possible. Users are able to see and interact with changes in the model in real-time. This reduces miscommunication and increases productivity. It is a collaborative environment that mimics an interactive whiteboard experience.

Tools that incorporate collaboration features for UML can be divided into different categories:

1. UML Modeling with Enhanced Collaboration Features like Enterprise Architect, and EclipseUML.They include real-time collaboration, allowing multiple users to work on the same model simultaneously.

2. UML Tools with Version Control Integration like Rational Rose, and Microsoft Visio. This feature allows for tracking changes over time and coordinating work among different team members.

3. Tools with Central Repositories such as Enterprise Architect, and EclipseUML. They enable teams to store and manage all UML models and related documents in a single, accessible location.

There are also cloud-based UML tools that are designed for collaboration from the ground up (Ex. LucidChart, Draw.Io). These tools meet the demand of modern software development, in which the members of the team require real-time collaboration capabilities.[16]

The advent of advanced techniques in deep learning generated a great interest in the exploration of methods to automatize the classification of UML class diagrams. Those advancements can improve the efficiency of software development processes and the utility of UML diagrams.

2.2.6 Challenges and Limitation of UML class diagrams

There are several challenges and limitations in using a class diagram:

- Complexity: if the complexity of a software system is too high, the diagrams can become difficult to read and understand, especially for people not familiar with UML notation.

- Ambiguity: UML provides a standardized notation but there can be ambiguities in how certain aspects are represented, leading to misinterpretations (especially when diagrams are shared among different teams or stakeholders). Another source of ambiguity is the level of detail. For example, the decision between a high-level overview versus a detailed representation depends on the diagram's audience and purpose.

- Evolution challenges: it can be a challenge to keep the diagrams up-to-date with the evolution of the software, especially in environments that change fast.

- Integration with other tools: even if there are a lot of tools to create and view UML diagrams, it is challenging to integrate them with other software development tools.

- Limitations in Reverse Engineering: there exist tools that can reverse engineer code into UML diagrams, but they are not always able to understand the design decisions related to the code.

- Overhead: the creation of detailed UML diagrams can add overhead to the software development process because even if they are useful for design and documentation, they require time and effort to create and maintain.

2.3 Large Language Models

2.3.1 Introduction

Large Language Models (LLMs) are a subset of artificial intelligence (AI) tools designed to process and generate text. They gained great attention after the public release of OpenAI's ChatGPT in November 2022. These models are able to perform tasks such as answering questions, summarizing, paraphrasing, and translating text at a level that often rivals human capabilities.

The fact that nowadays it is possible to actively interact with models like ChatGPT has made LLMs diffused tools in various domains, including engineering and medicine (some of the applications are reported in Figure 2.2). Even if they have the ability to democratize knowledge and improve access to information, there are several concerns. For example, they might accidentally spread misinformation or contribute to scientific misconduct due to a lack of accountability and transparency. While LLMs can be powerful tools, they are only as good as the data they are trained on. The presence of biases in the training data can lead to biases in the output of the model.[4]

LLMs' growth has been very fast. LLMs make use of neural network computational models that take inspiration from the human brain, and they are composed of interconnected nodes and process data through layers in order to execute tasks such as classification, pattern recognition, and decision-making) and have evolved since natural language processing (NLP) models (ex. BERT). OpenAI released GPT-1 in 2018, followed by other important models from companies such as Google and Meta. The release of ChatGPt was relevant due to its public accessibility, user-friendly nature, and human-like output. This was possible thanks to a unique approach known as reinforcement learning from human feedback (RLHF), which produces more reliable outputs compared to earlier models.

2.3.2 Definition and Core Concepts

Language models are defined as computational structures used to process and generate text. The rise of large language models influenced recent improvements in the NLP field. By training these models on a great amount of unlabeled text, they can do a great number of tasks without requiring much-specialized training. The ability of LLMs to execute several tasks without specialized training is often called "few-shot" or "zero-shot" learning. [17]

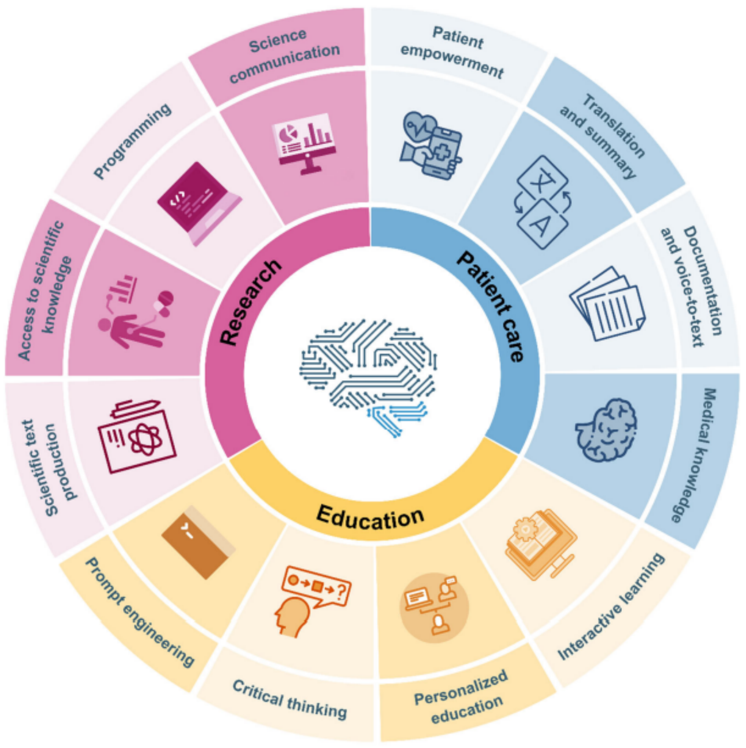


Figure 2.2: Overview of potential applications for LLMs in medicine [4]

In the beginning, traditional language models relied on statistical methods for which task-specific training was needed. In contrast, the newest generation of large language models (especially those that are based on the Transformer architecture) can be used for a lot of tasks with almost no specialized training. The range of these models has been continuously expanding, and some of them now brag about a number of parameters that are hundreds of billions.

Some of the core concepts of LLMs are:

- Training Data: a model's effectiveness is influenced by the quality and quantity of the training data. LLMs are usually pre-trained on massive datasets with different textual content.
- Tokenization: the process of converting input text into smaller units (tokens). It is considered one of the most important steps in the preparation of data for training language models and allows them to process and generate text effectively.
- Embeddings: they are vector representations of words, or tokens, that

catch the semantic meaning of words. They are essential for the ability of the model to understand and generate text that is relevant to the context, allowing models to understand relationships between words, such as synonyms, antonyms, and other semantic relationships.

- Attention mechanism: it allows the model to focus on specific parts of the input text when generating output. It is essential in order to understand the context and relationships within the text.

2.3.3 Evolution of Language Models

Over the years, Language Models have undergone great changes:

1. Statistical Models: The first language models were mostly statistical. By using the frequency of words and phrases in a text they were able to predict the next word in a sequence. Some of the techniques utilized were the Bag-of-Words and N-Grams.

2. Machine Learning Algorithms: they were integrated in order to capture characteristics of the language that purely statistical methods, due to their limitations, were unable to catch. Examples of ML algorithms are Naive Bayes and decision trees.

3. Recurrent Neural Networks (RNNs): due to the increase in the utilization of neural networks, RNNs started to be a diffused choice for language modeling. Their design includes the ability to remember past information, making them the ideal tool for sequential data (ex. text).

4. Long Short-term Memory (LSTM): a special type of RNN that resolved the vanishing gradient problem of traditional RNNs. It became a foundation in the performance of several NLP tasks (ex. text classification and machine translation).

5. Transformers and Attention Mechanism: the transformer architecture, introduced for the first time in [18], caused a relevant shift in NLP. By relying a lot on attention mechanisms, it allows the model to focus on different parts of the input text, leading to a better understanding of the context.

6. BERT, GPT, and Beyond: BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained transformer)

are two of the most important models which are based on the Transformer architecture. BERT is designed to understand the context of the words in a sentence by looking at its surroundings in both directions. Instead, GPT is focused on the generation of coherent and contextually relevant text. While BERT is more focused on understanding context (making it great for tasks like question-answering), GPT's design is more generative, making it adept at tasks like text completion or generation.

7. Transfer Learning in NLP: it became prominent thanks to models like BERT, GPT, ELMo, and ULMFit. These models are pre-trained on a very large amount of data and can be fine-tuned for specific tasks, leading to what many refer to as the "ImageNet moment" for NLP (it refers to the transformative impact on the field, similar to how pre-trained models on ImageNet revolutionized computer vision). [19]

2.3.4 Architecture and Mechanics

Transformer-based language models usually go under a two-phase training process (Pre-training and Fine-Tuning). The first phase is about pre-training the model on a very large amount of text or code with no supervision. In models that have over a hundred million parameters, this phase is followed by a short, task-specific training phase ("fine-tuning"). Pre-training allows the model to learn general language patterns, while fine-tuning adapts this general knowledge to specific tasks, making the model more effective.

Over the years the size of language models has continuously increased. The last generation of models (such as GPT-3) can be utilized without further specification (a technique referred to as "prompting"). While larger models like GPT-3 can be used with simple prompts, their size also brings challenges in terms of computational requirements and potential environmental concerns.

LLMs generate text based on the probabilities assigned to each word in their vocabulary. By considering the context and using their internal knowledge derived from training data they produce coherent and contextually relevant sentences. The model's generation can be controlled to some extent using techniques like temperature adjustment, which can make the output more deterministic or more random.[17]

2.3.5 Applications

LLMs have seen a fast adoption and have been acclaimed globally. In only five days after its release, ChatGPT gained one million users, indicating significant anticipation and a high demand for this technology.

ChatGPT was designed to perform several tasks. It can explain, analyze, and generate knowledge from the internet, replacing traditional keyword searches. The model is able to produce both creative texts and fact-based scientific essays. Moreover, it can program other algorithms, becoming a useful tool for both average consumers and specialized researchers. But while ChatGPT can generate knowledge, it is based on the data it was trained on, and it does not "know" in the traditional sense or actively browse the internet.

ChatGPT was also integrated into Microsoft's Bing search engine, combining traditional keyword search with the model's advanced syntax and interactive conversation capabilities.

ChatGPT's potential also extends to medicine and research. Due to the fact that it is able to generate fact-based texts in real-time, it is useful for medical professionals seeking evidence or case reports. It is important to verify any information generated by ChatGPT, especially in critical fields like medicine, to ensure accuracy and reliability.

There are also potential applications in other fields, such as education (e.g., tutoring, answering student queries) or entertainment (e.g., generating stories or dialogues).[20]

2.3.6 Challenges and Ethical Considerations

Simulating Moral Agency

New advancements in LLMs have enabled applications to simulate the possession of full moral agency. They can report context-sensitive moral assessments in open-domain conversations, suggesting a semblance of moral reasoning. While LLMs can simulate moral reasoning, they don't possess genuine understanding or consciousness, which can lead to potential pitfalls.

Automating Moral Decision Making

The idea of automating moral decision-making presents both methodological and ethical challenges that arise in areas such as:

- Bias Mitigation: it is necessary to ensure that the models do not perpetuate or amplify existing biases.

- Missing ground truth for moral “correctness”: the determination of what is morally correct is a subjective matter and it can vary across different cultures and individuals.

- Effects of Bounded ethicality in machines: it is necessary to understand that machines have limitations in making ethical decisions.

- Changes in moral norms over time: moral standards evolve over time, and models need to adapt accordingly.

- Risks of using morally informed AI systems as Advice: there are potentially negative consequences to relying on AI for moral guidance.

The potential for AI to make moral decisions raises concerns about accountability. Who is responsible when an AI makes a morally questionable decision?

Values Embedded in Technical Artifacts

AI models are technical artifacts and thus have values embedded within them. Some values like performance and efficiency are prioritized instead of moral values such as for example justice and diversity; even if the latter can be integrated by the utilization of algorithmic decision-making, their operationalization into clear success criteria is a challenge. These embedded values can also be unintentional and are the result of biases in the training data or from a mistake of the developer.

Training Morally Informed AI Systems

For the development of morally informed AI systems, one solution is to train them to answer human questions about moral decision-making in an autonomous way. The models are also trained to apply social norms, usually derived from large language corpora, to complex real-world situations. Training on social norms can be beneficial, but there is the risk of boosting existing societal biases if the procedure is not done carefully.[21]

Looking at the future, there is a really high potential for the integration of large language models with other software tools. For example, by the combination of these models with UML diagrams it might be possible to improve software development processes with the assistance of the models in requirements gathering, software design, and code generation.

2.4 Application of Large Language Models for Requirement Gathering: Literature Review

2.4.1 Introduction

Utilizing large language models for requirement gathering in software engineering is an innovative area full of potential. Requirement gathering aims to identify the user's needs in order to translate them into a set of specifications that can be methodically transformed into a working software system.

One of the main benefits is the possibility to automate the generation of UML diagrams from natural language descriptions. This process, which usually needs great manual effort and expertise, can be automated by using NLP techniques. An example is the approach of Jaiwai and Sammapun [22], who show the ability of NLP to realize requirements in languages also different from English, such as Thai, and automatically extract UML class diagrams. This accelerates the development process and improves the accessibility of UML modeling tools to non-English speakers.

There are also several challenges. The accuracy of the generated UML diagrams depends on the quality of the input requirements. Ambiguities in the language and the complexity of software requirements can create problems for automated systems. It is necessary to ensure that the generated models reflect the intended functionality. There is a necessary need for a balance between automation and human expertise. Large language models can manage a great portion of the modeling process, but they are not capable of replacing the decision-making capabilities of software engineers. The challenge is to develop systems that can complement human expertise.

In the next sections, three case studies of applications of Large Language Models in Requirements Gathering (or in similar fields) are analyzed.

2.4.2 AutoScrum: Automating Project Planning Using Large Language Models (2023)

This paper discusses an example of how to use LLMs to automate the agile project planning process.

Agile methodologies, with their focus on iterative development and flexibility, provide a robust framework for managing complex projects, and within this spectrum, Scrum stands out as a highly structured yet adaptable approach, emphasizing team collaboration and frequent reassessment of project goals to ensure efficient and effective outcomes.

Nowadays, LLMs are so advanced that they are able to perform advanced reasoning tasks. It is possible to make use of these capabilities to design high-level project plans by understanding both the actual state of the project and its desired state.

A scrum-based approach is proposed to automate the requirements-gathering process, the mapping of user stories, the identification of features, the decomposition of tasks, and the generation of question and search terms for domain-specific information. This approach is shown in Figure 2.3.

Alongside it, the paper also presented a shortcut plan approach able to generate more adequate tasks that permit to move from the current to the desired state in the most efficient way. This approach uses a single language program that suggests the next most urgent task based on tasks completed so far and the difference between current and desired situations. This language program is executed iteratively until the desired state is reached. This approach is shown in Figure 2.4.

By using those approaches and LLMs it is possible to automate different Scrum processes such as backlog creation, backlog refinement, task decomposition, and scrum execution. The authors of the paper conclude that LLMs can be used to support the processes of agile project planning and execution.

It is also important to notice, that the paper introduces the concept of “Language programs”. They are programs written in natural language and are designed to process input data through the language model. [5]

2.4.3 An Analysis on Large Language Models in Healthcare: A Case Study of BioBERT (2023)

The paper provides a look into the application of LLMs in healthcare, focusing on BioBERT.

Starting from a discussion on the evolution of natural language processing

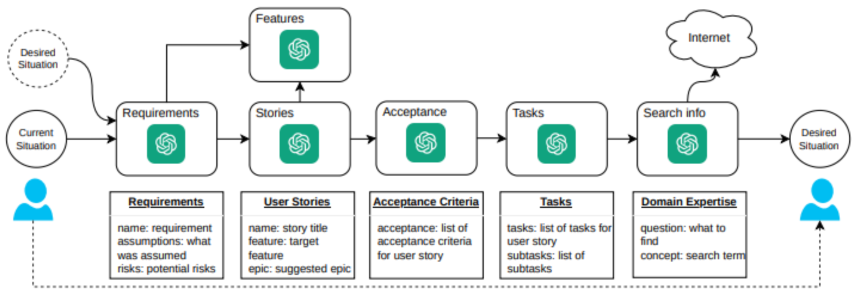


Figure 2.3: Autoscrum process. The green icons represent parts of the process that can be augmented by the use of language models [5]

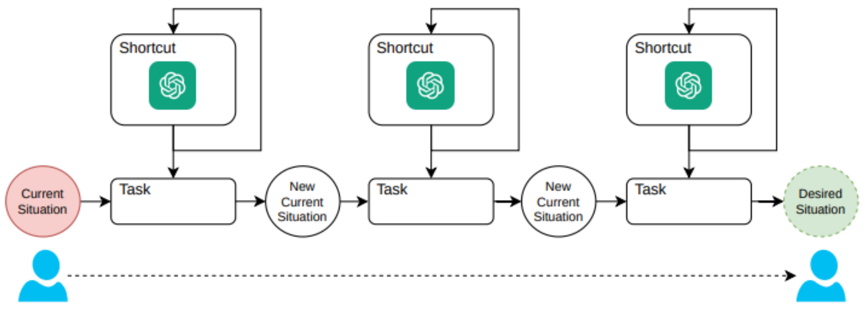


Figure 2.4: The shortcut planning approach [5]

(NLP), starting from early rule-based systems and up to the introduction of the Transformer Architecture that led to the development of LLMs, the paper shows the transformative impact of LLMs on healthcare and biomedical applications, highlighting inefficiencies in the current healthcare system, such as the limited access to information and unmanageable documentation processes. It is suggested that LLMs are able to surpass these challenges by providing rapid, context-aware responses to medical questions, extracting information from unstructured data, and automating clinical documentation.

The paper carefully examines BioBERT, a language model realized for biomedical text mining. By using a systematic methodology, BioBERT is fine-tuned to meet the particular needs of the healthcare domain, including data gathering from diverse healthcare sources, data annotation, and specialized preprocessing techniques.

Other than the benefits of using BioBERT in healthcare, it is also necessary to recognize the challenges related to data privacy, integrity, bias mitigation, and transparency. Ethical aspects such as patient privacy and data security are also considered.

It can thus be said that LLMs can be tailored to specific domains, such as healthcare, and this can open the door to applications in requirement gathering for software engineering, especially in UML class diagrams. [23]

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

This paper illustrates an approach to automatically transform user stories into UML diagrams by addressing the challenges encountered. User stories are usually used in Agile methodologies to express requirements.

The authors used Stanford Core NLP (a tool for natural language processing) to process user stories written in natural language. It used an approach that combines rules formulated as predicates and an ontological model to interpret the user stories.

Prolog rules were used to find relationships between classes and to improve them aiming to reduce errors. An ontology is created to represent the components of the user stories, which helps to find equivalent relationships and to include use cases. The approach is shown in Figure 2.5.

The tool developed for this process was implemented in Python and it was applied in several case studies, demonstrating its validity.

There exist previous approaches to generate conceptual models from user stories, but they have a lot of limitations, like incomplete models or the lack of relationship between use cases. [6]

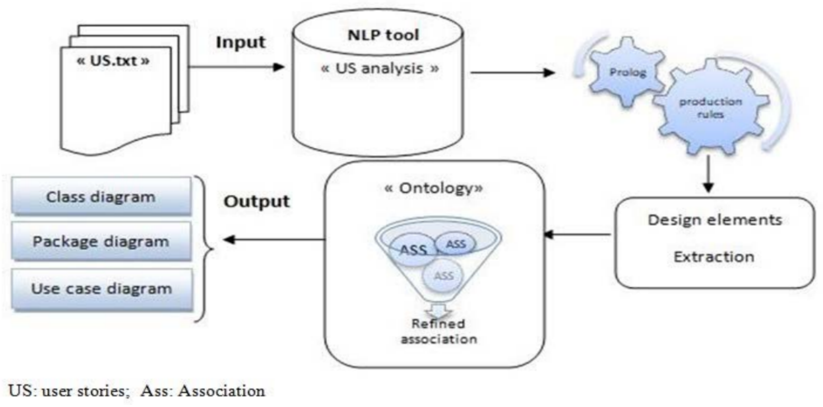


Figure 2.5: Architecture of the Proposed Approach[6]

Chapter 3

Methodology

3.1 Possible evaluation Criteria

3.1.1 Introduction

In the fields of software engineering and system design, Unified Modeling Language (UML) class diagrams play a crucial role. It is essential to assess their quality and this will be one of the main focus of this research. Evaluating UML class diagrams goes beyond checking for syntax errors; it involves determining how well they represent the system, how effectively they communicate information, and whether they adhere to design practices. The quality of a class diagram has an impact on the development process affecting how easily the system can be implemented, maintained, and scaled.

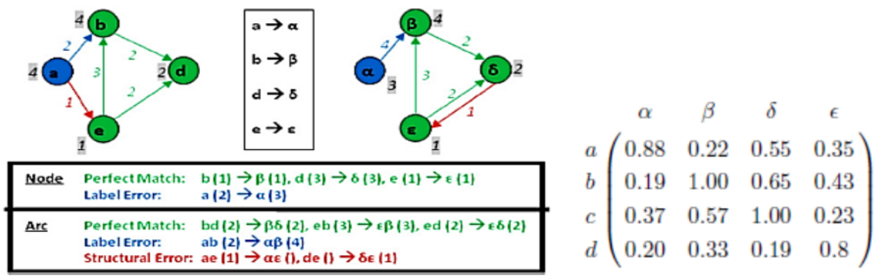
During the literature review, for this research project, evaluation methods from five academic articles are taken into consideration. These methods provide an overview of the state of the art regarding the evaluation of UML class diagrams.

3.1.2 Overview of Articles and Methods

New Method for Summative Evaluation of UML Class Diagrams Based on Graph Similarities (2021)

This article shows an approach to evaluate UML class diagrams generated by students. The evaluation system operates semi-automatically, comparing student-generated diagrams with teacher-provided ones.

The method adopts different graph-matching approaches, such as graph isomorphism and the search for larger common subgraphs and it is structured into three sequential steps: preprocessing the input diagrams into graphs, the matching process to calculate element similarities, and finally, a formative evaluation that includes a list of differences, and errors, and a summative evaluation to classify compared diagrams. An example is shown in figure 3.1.



Node	PerfectMatch:	b (1) → β (1), d (3) → δ (3), e (1) → ε (1)
	Label Error:	a (2) → α (3)
Arc	PerfectMatch:	bd (2) → βδ (2), eb (3) → εβ (3), ed (2) → εδ (2)
	Label Error:	ab (2) → αβ (4)
	Structural Error:	ae (1) → αε (), de () → δε (1)

$$\begin{array}{c|cccc} & \alpha & \beta & \delta & \epsilon \\ \hline a & 0.88 & 0.22 & 0.55 & 0.35 \\ b & 0.19 & 1.00 & 0.65 & 0.43 \\ c & 0.37 & 0.57 & 1.00 & 0.23 \\ d & 0.20 & 0.33 & 0.19 & 0.8 \end{array}$$

Figure 3.1: Example of a Graph Matching[7]

The similarity measure is key to this approach, aiming to identify semantic correspondences between graph elements. This is achieved through a set of comparison functions that evaluate similarities between two nodes, taking into account their attributes and neighboring nodes. Weights and thresholds are assigned to each property to adjust the significance of different similarity aspects.

The tool uses syntactic, structural, and semantic similarity functions. Syntactic similarity measures lexical similarity (e.g., class and attribute names), structural similarity assesses the similarity of properties (like attributes and operations), and semantic similarity evaluates the relations of elements with their neighbors.

The method proved capable of measuring similarities between various UML class diagrams, detecting differences, correcting errors, and matching class diagrams. The results demonstrated that the method is largely effective, with a significant majority of matches aligning with expected outcomes.[7]

On Evaluating the Layout of UML Class Diagrams for Program Comprehension (2005)

The authors of this paper present key criteria and guidelines for the effective layout of UML class diagrams and evaluate two UML tools (Borland Together and Rational Rose) to show how these criteria can improve readability. Among those criteria, the most important are:

- Law of Good Figure: Join together inheritance arcs of the same class, and select carefully which information to show.
- Law of Continuation: Minimize edge crossings and bends to facilitate comprehension.
- Law of Proximity: Place related elements (like parent and child nodes) close to each other, and ensure that the length of the edges is neither too long nor too short.
- Law of Connectedness: Avoid overlapping of nodes and edges to maintain clear visual distinctions.
- Law of Orientation: Prefer vertical or horizontal orientations for elements and labels, and draw arcs orthogonally.
- Law of Contour: Prevent overlapping of objects for easier recognition.

It is important to notice that some criteria may conflict with each other, and so it is necessary to make potential trade-offs. [24]

An Approach to Compare UML Class Diagrams Based on Semantical Features of Their Elements (2015)

In their research, the authors analyzed existing methods for UML class diagram comparison, focusing on those that provide numerical metrics to describe model differences, and then combined two methods: one that uses several similarity metrics like Shallow Lexical Name Similarity and Attribute Similarity, and another which focuses on attribute and operation sets to define differences.

The obtained comparison method involves pairing elements from two diagrams based on their semantical meaning, calculating the distance between them, and then aggregating these distances into a model difference vector. This vector's length is used to estimate the final difference between the diagrams. The approach includes evaluating distances between classes, attributes, methods, and relationships. The criteria for calculating these distances vary depending on whether elements are semantically equal, have different names, or are absent in one of the models. The final output is a number that defines the distance between the compared diagrams, with a larger number indicating more differences.

To validate their method, the authors created three simple UML class diagrams and compared them using their approach, demonstrating in this way that the method is effective in differentiating between diagrams based on both name and structural differences. [2]

Assisting Novice Analysts in Developing Quality Conceptual Models with UML (2006)

The objective of this scientific article is to help inexperienced analysts in the creation of UML artifacts, focusing on their quality. Those models can be useful to represent correctly the system requirements improving the success of the system development process. Different techniques for modeling using UML are discussed: entity-relationship diagrams, class diagrams, use case diagrams, and sequence diagrams.

The authors analyzed UML artifacts realized by undergraduate students in their final year, who had previously attended a course in object-oriented analysis/design. They were taken by 15 team-project reports, and the class diagrams contained in them included an average of 14 classes with up to 50 attributes and 23 operations.

Errors usually made by novice analysts are highlighted in the findings of this paper, and they can be used to improve the quality of the artifacts developed, thanks to training programs that focus on them. In this way, semantic and pragmatic errors can be avoided.

The authors also suggest that those errors can be used to create checklists and guidelines useful to quality assurance teams and systems analysis instructors. [1]

Understanding Quality in Conceptual Modeling (1994)

This paper focuses on the quality of conceptual models, by introducing the need for a systematic approach for the identification of quality improvement goals and means to achieve them.

The authors propose a framework that introduces a comprehensive approach that includes the following components:

- Syntax: focus on whether language rules and their implications on the construction of the model are respected. It is formed by an alphabet and a grammar.
- Semantics: consider the relations among statements and their meaning. It is assessed based on the model's adherence to the domain's characteristics and requirements. Goals for semantic quality include validity and completeness.

- Pragmatics: focuses on how the model is understood by various stakeholders involved in the development process. Its goal is to ensure comprehension of the model by all concerned parties, not just select groups.

The authors also discuss the feasibility of achieving these quality goals, acknowledging the trade-offs between benefits and drawbacks in pursuing model quality. [25]

3.1.3 Analysis of Proposed Methods

The previous section presented five distinct articles, each offering a different method to evaluate UML class diagrams. The current objective is to make a comparative analysis to select the most suited method(s) to adopt in this study. First, the three methodologies that were discarded, with the motivations behind such a decision, will be discussed, and then an explanation for the two selected methodologies will be provided.

Methodologies Not Selected

"New method for summative evaluation of UML class diagrams" [7] was discarded because it employs graph matching techniques to evaluate diagrams. This is too complex and unsuited for the scenario that is considered, where simpler or more direct comparison methods are preferred. Moreover, it involves a multi-step process that can be time-consuming and may require significant computational resources.

"On Evaluating the Layout of UML Class Diagrams for Program Comprehension" [24] was not selected because it is more focused on the tool used to make the diagram than on who made it, with the final example in the article being a confrontation of the same diagram made by the same user with different tools, judging perceptual aspects of them.

"Understanding Quality in Conceptual Modeling" [25] was deemed not suitable because it lacks specificity to UML class diagrams. While offering broad perspectives on model quality, it does not offer the UML-specific criteria necessary for an exhaustive evaluation in this study.

Chosen Methodologies

"An Approach to Compare UML Class Diagrams Based on Semantical Features of Their Elements" [2] was selected because it is centered on evaluating the semantical features of UML class diagram elements, ensuring that the comparison goes beyond superficial aspects like naming conventions, focusing instead on the underlying meaning and relationships within the diagrams. The method systematically compares various elements such as classes, attributes, methods, and relationships, in this way all critical aspects of the diagrams are evaluated. Moreover, it effectively manages different versions of UML diagrams, is in line with current software development trends, and has been validated in practical scenarios.

"Assisting Novice Analysts in Developing Quality Conceptual Models with UML" [1] was chosen due to its effectiveness in identifying and addressing common modeling errors that can occur in UML diagrams. The methodology uses a systematic framework to evaluate UML artifacts from syntactic, semantic, and pragmatic quality perspectives. This evaluates the overall quality of the diagrams, considering not only their structural correctness but also their relevance to the domain and their comprehensibility to various stakeholders.

The analytical process has resulted in the selection of two methodologies that align with the goals and context of the research. These chosen approaches offer a robust framework for evaluating UML class diagrams. The methodologies not selected, while valuable in their contexts, do not align as closely with the specific aims and requirements of the research in the evaluation of UML class diagrams.

3.2 Approach to Evaluate Diagrams

The previous section focused on the identification of studies on which to lay the basis for the analysis that will be done in this thesis. Starting from them, a two-part approach was elaborated, to compare class diagrams realized by humans with the ones made by LLMs.

The approach starts with the first part focused on an analysis of the frequency and nature of errors across different quality categories. The objective is, by quantifying these errors, to measure if the diagrams of the best quality are produced by humans or LLMs.

Then, the second part shifts to a direct comparison of the diagrams based on the semantic features of their elements. By using mathematical calculations, it aims to measure the distance between humans or LLM in the realization of a more accurate and effective diagram related to the same system.

With this dual-faceted approach, the study aims to establish the potential of LLMs in the generation of UML class diagrams.

3.2.1 First Part

In the first part, the focus is on evaluating the quality of diagrams across different aspects:

1. Syntactic Quality: assess if the class diagrams follow the syntactic structure of UML with minimal errors or deviations;
2. Semantic Quality: evaluate the accuracy and completeness of the diagrams in representing the intended domain. It is measured in terms of:
   - Validity: all elements and relationships in the diagrams should accurately represent the domain.
   - Completeness: the diagrams should include all necessary elements and relationships, it is important to identify which are incorrectly represented or omitted.
3. Pragmatic Quality: focus on the understandability of the diagrams from the perspective of stakeholders. The comparison revolves around how each diagram presents information (it can make it easy or challenging for the users to understand).

In Table 3.1, typical errors under the three quality dimensions (syntactic, semantic, and pragmatic) are categorized and listed.

A negative correlation will be considered between the number of errors in class diagrams and their overall quality, meaning that the fewer errors are present, the higher the quality of the diagram. Such comparison highlights the strengths and weaknesses of LLMs compared to human-generated diagrams.

During the analysis, the number of errors for each distinct quality will be reported in a specific table (Table 3.2 is an example), which serves as a tool to understand how the frequency and types of errors vary between humans and LLM. [1]

3.2.2 Second part

The approach in this second part will be focused on comparing and analyzing the semantic content of elements and relationships in the diagrams.

Syntatic Quality	Semantic Quality	Pragmatic Quality
• Missing cardinality details for associations • Inappropriate naming of classes and associations • Incorrect use of UML symbols	• Incorrect cardinality/multiplicity specification • Used aggregation in place of association • Wrong location of attributes • Wrong location of operations • Operation cannot be realized using existing attributes and relationships	• Redundant attributes and associations • Specialization with little distinction among subclasses • Inconsistency in Styling and Conventions

Table 3.1: Errors in various quality categories [1]

	Syntatic Quality	Semantic Quality	Pragmatic Quality
Human	26	32	33
LLM	28	30	27

Table 3.2: Example of Distribution of various type of errors [1]

This is essential because even if elements have different names they can be semantically identical.

The elements that will be analyzed are:

- Classes and interfaces.
- Class attributes.
- Methods.
- Relations between elements.

The steps of the comparison algorithm that will be used for each element are:

1. Pairing elements from the solution given with the diagrams generated by a human and those generated by LLM, based on their semantic meanings.

2. Calculate the distance between the elements of each pair. The distance is a measure of how differently are the elements represented.

3. Aggregate all the calculated distances into a model difference vector, it will be used to calculate the final difference.

Only after those steps, is obtained a vector whose length will be assessed to determine the distance between the solutions given and the diagrams made by humans and LLM. The diagram with a minor distance from the solution will be considered the best.

Classes and Interfaces

Distances between classes and interfaces are calculated using the guidelines in Table 3.3

Criteria	Distance
Semantically equivalent elements with the same names are present in both models	0
Semantically equivalent elements are present in both models but their names differ	0.5
One of the model does not contain a semantically equivalent class with the other model	1

Table 3.3: Class and Interface Distances[2]

Class Attributes

To calculate the distances between the class attributes, it is necessary to create a temporary vector:

$$<a|s|n|t> \tag{3.1}$$

The vector is composed of different characteristics of the attributes, with its length being used as a way to measure the distance between corresponding class attributes in the solution versus the ones in the diagrams made by humans and LLM. Table 3.4 explains how to construct and interpret this vector.

Element	Criteria	Distance
a	Difference between access modifiers of appropriate class attributes	0 for the same, 1 for different
s	Static modifier flag	0 if both attributes share the same static modifier, 1 otherwise
n	Name difference	0 for the same attribute names, 1 for different
t	Attribute type difference	0 for the same type, 1 for different

Table 3.4: Elements of Comparison Vector for Class Attributes[2]

All the elements of the vector are assigned a value of 1 if the class attributes or the class itself exists only in one of the two diagrams compared.

Methods

The following vector has to be created to compare the class methods:

$$<o|a|s|n|p|r> \tag{3.2}$$

Its components are explained in Table 3.5.

The importance of the method arguments needs to be highlighted by adding a factor of 0.2 to the overall difference if there is a mismatch, and 0.5 if an argument is present in one method but not in the other. The following formula shows this:

$$p = 0.2 * a_t + 0.5 * a_m \tag{3.3}$$

Inside it, 'a_t' is the number of method arguments with mismatching types, and 'a_m' is the number of cases where a method argument is present in one diagram but absent in the other.

It's important to note that the order of arguments is not considered in this analysis.

Element	Criteria	Distance
o	Owning class difference	If a method is defined in semantically equal classes (interfaces) – 0, 1 otherwise
a	Difference between access modifiers	0 for the same, 1 for different
s	Static modifier flag	0 if both methods share the same static modifier, 1 otherwise
n	Name difference	0 for the same method names, 1 for different
p	Difference between method arguments	0.2 for each mismatching attribute type, 0.5 for missing argument
r	Difference between return type	0 when the return type is semantically equal, 1 otherwise.

Table 3.5: Elements of Comparison Vector for Class Methods[2]

Relations Between Elements

The following vector is realized to compare the relations between elements:

$$<s|t|y|m> \tag{3.4}$$

Its components are explained in Table 3.6

Length and Distance

For every n-dimensional vector, its length is calculated by using an Euclidian distance formula:

$$l = \sqrt{\sum_{i=1}^{n} e_i^2} \tag{3.5}$$

The final output of the comparison is a numerical value calculated as the length of the model difference vector created by the aggregation of the distances of all the elements analyzed. It indicates the distance between the diagram in the solution and the ones made by humans and LLM. A larger number indicates a greater disparity, while a smaller one indicates that they are similar.[2]

Element	Criteria	Distance
s	Relation source difference: denotes if the relation is outgoing from the semantically equal class in both models	0 for the same class, 1 for different
t	Relation target difference: denotes if the relation is incoming into the semantically equal class in both models	0 for the same class, 1 for different
y	Relation type difference	0 if both relations are of the same type, 1 otherwise
m	Multiplicity difference	0 if relations have the same multiplicity, 1 otherwise.

Table 3.6: Relation Comparison Vector Elements[2]

3.3 Collection of Exercises

For the analysis conducted in this thesis, a collection of exercises, selected from several internet sources, has been gathered. These exercises range in different sectors to give different representations of the challenges that can be encountered in the realization of UML class diagrams.

This collection comprises 20 distinct exercises each of them composed of three crucial elements:

- A detailed text description of the system for which the class diagrams will be constructed.
- A direct link to its internet sources, to allow for further exploration of the context of each exercise and acknowledge the original creator.
- A solution, that has been provided alongside the original content. It will be used as the basis for the comparison between diagrams made by humans with the ones made by LLMs.

The link for the digital appendix containing this collection can be found in appendix A.

3.3.1 Different Levels of Difficulty

Exercises of different difficulty levels were included in the collection, this diversity is essential to test and compare the capabilities of both human intellect and language model algorithms in generating UML class diagrams.

The difficulty of each exercise is determined by a numerical scale ranging from 1 to 5. The definitions for each point on the scale are as follows:

- 1 [Very Easy]: exercises that involve basic concepts and a small number of classes and relationships. They are straightforward, with clear requirements.
- 2 [Easy]: exercises slightly more complex than the previous ones. They might include a few more classes or relationships but they remain simple in terms of interactions and concepts.
- 3 [Moderate]: exercises that present a balanced difficulty level, with a moderate number of classes and relationships. They introduce some complexities in the interactions between classes. A good understanding of UML principles is required to solve them.
- 4 [Hard]: complex exercises that require a deep understanding of UML methodologies. They have a large number of classes and intricate relationships.
- 5 [Very Hard]: the most difficult exercises, they mimic real-world applications and involve extensive systems with numerous classes, complex relationships, and advanced concepts, requiring expert knowledge and experience to model effectively.

The evaluation process is made by a sample of three individuals, each with experience or familiarity with UML class diagrams. These evaluators independently review the exercise descriptions and assign a difficulty rating based on their initial assessment, before any attempts at generating corresponding UML class diagrams. This precaution is taken to avoid potential biases that might arise from the diagram generation process itself, thereby aiming for a more objective initial evaluation.

The final Estimated Difficulty, for each of the twenty exercises, is calculated as the average of the three scores provided by the evaluators. This value serves as a standardized measure of the exercise's difficulty, offering a point of reference that includes the collective judgment of the evaluators.

3.3.2 Given Solutions

It is crucial to note that in this analysis, the provided solutions are regarded as the primary benchmark for correctness. Despite their potential limitations, such as being incomplete or overly simplistic, these solutions are essential as they offer a standardized approach to UML modeling of the described system. Their role as a comparative framework is central, and they are used as the sole point of reference against which all generated diagrams are evaluated.

3.3.3 Example of Exercise

This section shows one of the twenty exercises in the collection. The exercise is on a real-life application in the medical field, called MyDoctor, and it has an estimated difficulty of "4" (Hard). The text is shown in figure 3.2, and the solution given is in figure 3.3.

3.4 Diagrams made by Human

3.4.1 Tool Used

In this research, the tool selected to represent the human-made UML class diagrams was the Microsoft Visio tool, which presents both pros and cons:

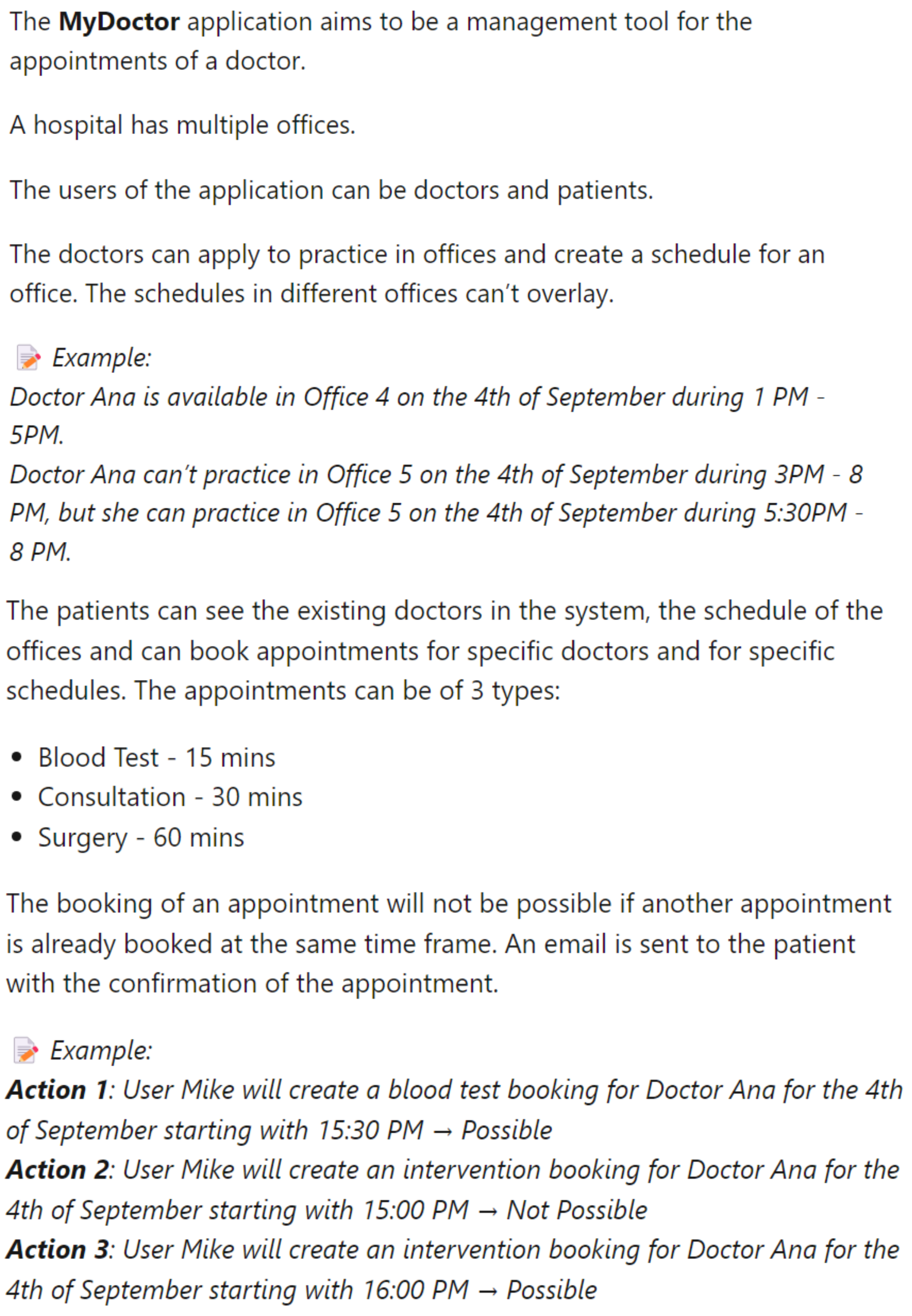
Pros

- No additional cost: Visio is included in the Microsoft Office suite provided by the university for which this research is done, meaning that no extra expenses are needed.

- Offline functionality: it is possible to use Visio offline once installed on the laptop. This is essential to ensure that the diagrams can be realized without any kind of interruption.

- User-Friendly: Visio is easy to use for people familiar with the Microsoft Office Suite, with an interface similar to other tools like Excel or Word.

- Variety: a large number of templates for UML diagrams are included in the application, facilitating the creation of professional-looking class diagrams.



The **MyDoctor** application aims to be a management tool for the appointments of a doctor.

A hospital has multiple offices.

The users of the application can be doctors and patients.

The doctors can apply to practice in offices and create a schedule for an office. The schedules in different offices can't overlay.

Example:
Doctor Ana is available in Office 4 on the 4th of September during 1 PM - 5PM.
Doctor Ana can't practice in Office 5 on the 4th of September during 3PM - 8 PM, but she can practice in Office 5 on the 4th of September during 5:30PM - 8 PM.

The patients can see the existing doctors in the system, the schedule of the offices and can book appointments for specific doctors and for specific schedules. The appointments can be of 3 types:

- Blood Test - 15 mins
- Consultation - 30 mins
- Surgery - 60 mins

The booking of an appointment will not be possible if another appointment is already booked at the same time frame. An email is sent to the patient with the confirmation of the appointment.

Example:
Action 1*: User Mike will create a blood test booking for Doctor Ana for the 4th of September starting with 15:30 PM → Possible*
Action 2*: User Mike will create an intervention booking for Doctor Ana for the 4th of September starting with 15:00 PM → Not Possible*
Action 3*: User Mike will create an intervention booking for Doctor Ana for the 4th of September starting with 16:00 PM → Possible*

Figure 3.2: Text of the Exercise [8]

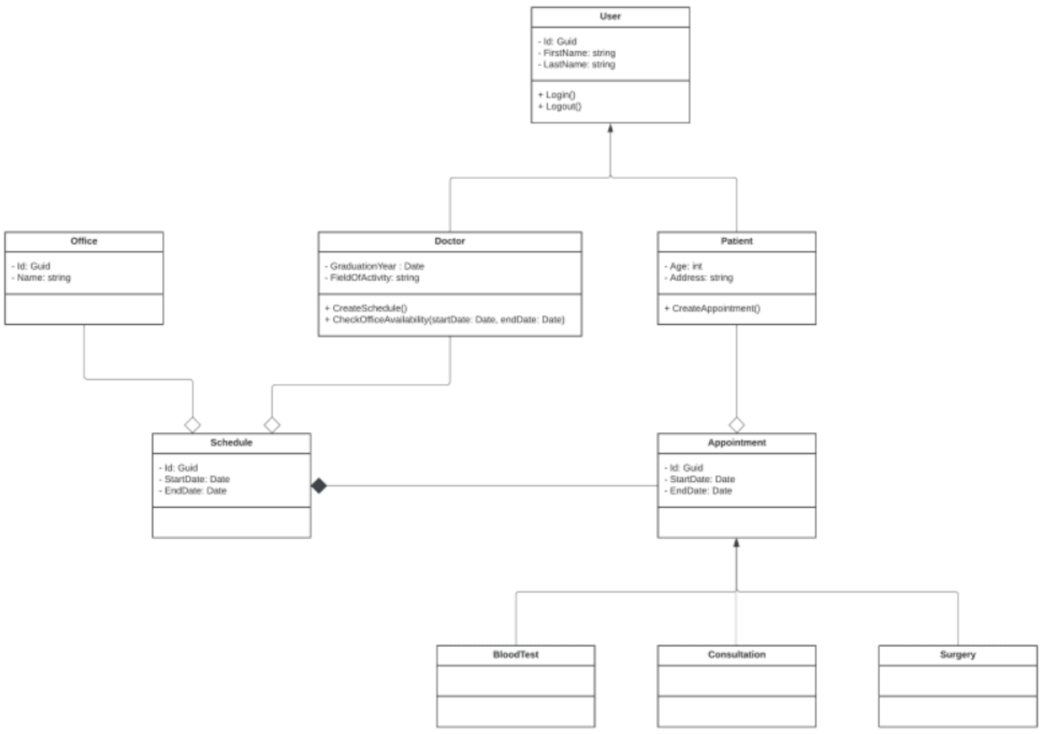


Figure 3.3: Given Solution [8]

Cons

- Learning Curve: the basic features of Visio are straightforward, while more complex ones require a deeper understanding of the tool.
- System compatibility: Visio is a tool designed for Windows, and there might be limitations when used on different operating systems.

Therefore, Microsoft Visio is a cost-effective and capable tool for creating UML class Diagrams, especially thanks to the university-provided access and its offline capabilities.[26]

3.4.2 Established Rules

In the realization of the Class Diagrams, it was pivotal to establish rules to follow, that work as a constraint to simulate a controlled real-world environment like the one of professionals working with time constraints and limited resources.

The rules are the following:

1. Time limitation: the time given to complete a UML class diagram is 30 minutes, in order to evaluate the ability to work under time pressure and to avoid prolonged deliberation.
2. No access to external sources: to ensure that the diagram is made only with personal knowledge, access to the internet, notes, textbooks, or any other external sources during the execution of the exercise was forbidden.
3. No interaction with others: the exercises were completed individually, without help from other people. This was necessary to evaluate the competence of a single human against the LLM.

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

3.4.3 Limitation

The greatest limitation in this part of the research is the use of a single subject.

The risk of this choice is that the findings from a single individual may not be generalizable to a large population, because individual differences can influence the outcomes of the exercises.

In single-subject research, the individual serves as their own control and treatment group, and it can be both an advantage and a disadvantage, as while this allows for a precise examination of individual performance, it means that the findings are dependent on the specific characteristics of the individual.

Results may not replicate when applied to different individuals or even to the same individual in a different time frame.[27]

Retaking the example of the MyDoctor application, the solution realized by the human is shown in figure 3.4.

3.5 Diagrams made by LLM

3.5.1 Chosen LLM

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

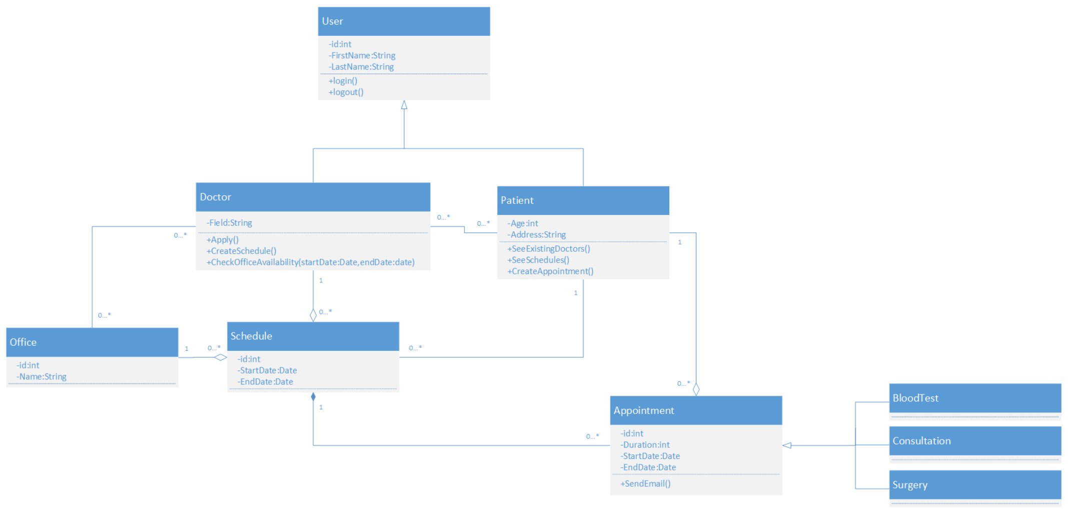


Figure 3.4: Solution made by human using Microsoft Visio

Even if it aims to be accurate, ChatGPT-4 is not infallible and can produce plausible but incorrect responses. The choice of a correct prompt is pivotal to improving the correctness of his responses.

3.5.2 Prompt

In the context of AI, a prompt is an instruction given to the AI model, that serves as the initial input to guide the AI's response, influencing its quality.

Prompt engineering is the field that crafts these inputs aiming to receive the most effective and accurate responses from an AI system.

A well-designed prompt is essential to:

- Provide context and direction by including specific details so that the model can produce output more aligned with the user's intention.
- Enhance AI efficiency by reducing ambiguity, leading to quicker and more precise results
- Facilitate complex task execution by conveying all the necessary information.

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

3.5.3 PlantUML

PlantUML is an open-source tool that enables the creation of diagrams using text-based language and it is especially useful in the developing of UML class diagrams.

In this research ChatGPT will generate, starting from the text of an exercise, a PlantUML code that is inserted in a specific tool to generate the corresponding class diagram. The selected tool for this purpose is PlantText.com, an online free tool that quickly generates UML diagrams starting from a PlantUML text. It was chosen for the following reasons:

- It is easy to use thanks to its user-friendly interface
- Allows real-time visualization as the PlantUML code is entered
- Diagrams can be easily exported in various formats like PNG, SVG, and TXT
- Being a web-based tool doesn't need a software installation

The simplicity of PlantUML's text-based syntax, combined with the accessibility of PlantText.com, provides the ideal tool to realize Class Diagrams by using LLM.

Retaking the example of the MyDoctor application, the solution realized by ChatGPT-4 is shown in 3.5.

3.6 Statistical Analysis

JASP

All the statistical analysis will be conducted using JASP (Just Another Statistical Program), which is an open-source statistical software designed for statistical analyses. It offers an intuitive interface that simplifies the analyses making them accessible for both novices and experienced users, facilitating

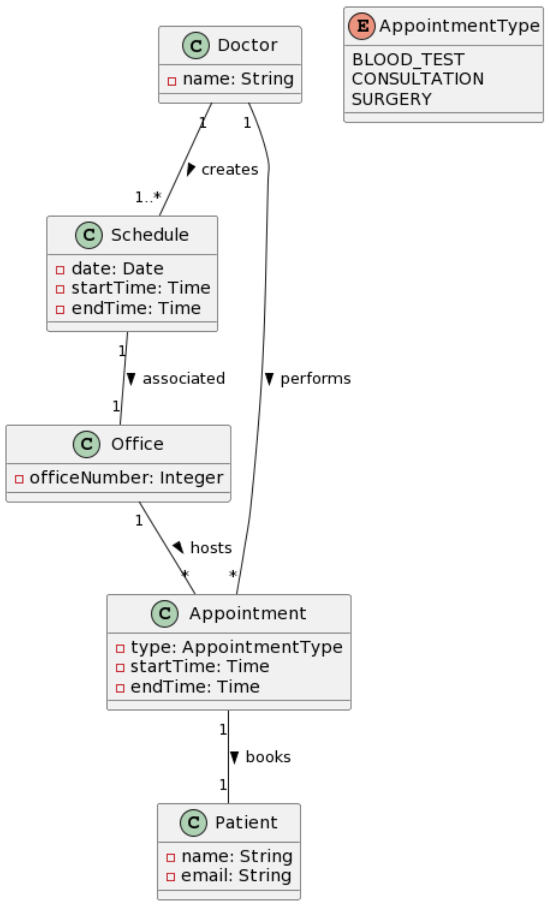


Figure 3.5: Solution made by ChatGPT-4 in PlantUML

the understanding and interpretation of statistical results without the need for extensive programming knowledge.

Jasp was selected for the statistical analysis of this study due to the following reasons:

- User-Friendly Interface: JASP offers a graphical user interface (GUI) with a point-and-click approach that allows for easy model specification so it is more approachable for users who may not be familiar with R's command-line interface.

- Real-Time Analysis: JASP provides real-time results, meaning that as data or analysis parameters are changed, the results update immediately, allowing for a dynamic exploration of data.

- Integrated Output: The output in JASP is presented in a format that is ready for reports or publications saving time on formatting and reducing the chance of transcription errors.

3.6.1 T-Test

To confront human-made diagrams with the LLM ones, a paired-sample t-test will be used.

A paired-sample t-test is a statistical procedure used to compare the means of two related groups. It is commonly used when the same subjects are involved in both groups or when a pair of subjects is matched in terms of a control variable. In this study, the test is applicable because the two sets of diagrams can be considered in a paired design (each human-created diagram has a direct counterpart created by the LLM).

Output

In the Output of JASP for a paired samples t-test, several components provide information about the relationship between the two groups.

First, there is a Descriptive Statistics Table that provides:

- N: The number of paired observations included in the analysis.
- Mean: The average value of the observations for each of the paired groups.
- SD (Standard Deviation): A measure of the variation or dispersion of the observations in each group.
- SE (Standard Error): An estimate of the standard deviation of the sampling distribution of the sample mean, indicating the precision of the calculated mean.
- Coefficient of Variation: This statistic provides a standardized measure of the dispersion of the dataset relative to its mean.

The second output component provided is a Paired Samples T-Test Table whose elements are:

- t: This value represents the calculated t-statistic, which is used to determine if there is a statistically significant difference between the paired groups.

- df: Stands for degrees of freedom, which in a paired samples t-test is related to the number of pairs in the data. It is calculated as N - 1.
- p: The p-value indicates the probability that the results could have occurred under the null hypothesis. A low p-value (typically less than 0.05) suggests that the null hypothesis can be rejected, meaning there is a significant difference between the groups.

Lastly, a Bar Plot that visually displays the means of the two groups with their respective confidence intervals of 95% is provided.

3.6.2 Linear Regression

Linear regression is a statistical method that is used to model the relationship between a dependent variable and one or more independent variables. Its primary purpose is to understand and model the relationship between a dependent variable and one or more independent variables. In this research, it will be used to predict outcomes based on the characteristics of UML class diagrams.

Output

The outputs from JASP for a linear regression analysis include several components. The first is the so-called Model Summary, which provides a snapshot of the overall model's performance. It includes:

- R: The correlation coefficient, which measures the strength and direction of the linear relationship between the dependent variable and the independent variable(s).
- R^2 (R-squared): This represents the proportion of variance in the dependent variable that can be explained by the independent variable(s) in the model.
- Adjusted R^2: Adjusted for the number of predictors in the model, this value compensates for the addition of variables that do not improve the model.
- RMSE (Root Mean Square Error): This is the standard deviation of the prediction errors (residuals). It measures how spread out these residuals are around the best-fitting line. Lower values of RMSE indicate a better fit.

The second component is a table that contains the estimated coefficients of the model, describing the magnitude and direction of the relationship between each predictor and the dependent variable. The table includes:

- Unstandardized Coefficients: These represent the actual units of the dependent variable expected to change as the independent variable(s) change.
- Standard Error: This measures the average amount that the coefficient estimates vary from the actual average value of our response variable. The smaller the standard error, the more precise the estimate.
- Standardized Coefficients (Beta): These coefficients are the estimated coefficients obtained by standardizing the variables involved. They are used to compare the relative strength of the different predictors within the model.
- t: The t-statistic is the coefficient divided by its standard error. It is used to determine the significance of the predictors.
- p: The p-value tells the likelihood or probability that the coefficient for a variable is due to chance. A p-value of less than 0.05 is typically used as the threshold for significance.

Lastly, JASP outputs a plot that visualizes the residuals (the differences between observed and predicted values) against the predicted values (dependent variable).

- Residuals: These are the vertical distances between the data points and the regression line. Ideally, they should be randomly distributed, with no discernible pattern.
- Standardized Residuals: These are the residuals that have been standardized to have a mean of zero and a standard deviation of one. The plot allows checking for homoscedasticity — whether the variance of the errors is consistent across all levels of the independent variables.

Chapter 4

Results

4.1 First Part

4.1.1 Example

To better understand the methodology of the first part, an example will be made using exercise number 10 of the collection, which describes a restaurant information system (the whole exercise with the diagrams can be found in the appendix B).

By confronting the diagram made by the human and the diagram made by ChatGpt with the solution given, the following results were found:

Human-Created Diagram Errors

- Syntactic errors. No syntax errors were found.

- Semantic errors. One semantic error was found: *Incorrect Multiplicity in the Association Between the Classes "Dish" and "Meal"*. The diagram inaccurately represents the relationship between the two Classes, by considering a multiplicity of 1 to * instead of 0 to * on the class "Dish" indicating a potential misunderstanding of the domain or the exercise requirements.

- Pragmatic errors. One pragmatic error was found: *Omission of 'Person' as a Parent Class for the Classes "Client" and "Waiter"*. The lack of a unifying 'Person' class suggests a missed opportunity to abstract common attributes for different person-related entities, which could impede the stakeholders' comprehension and the system's scalability.

LLM-Created Diagram Errors

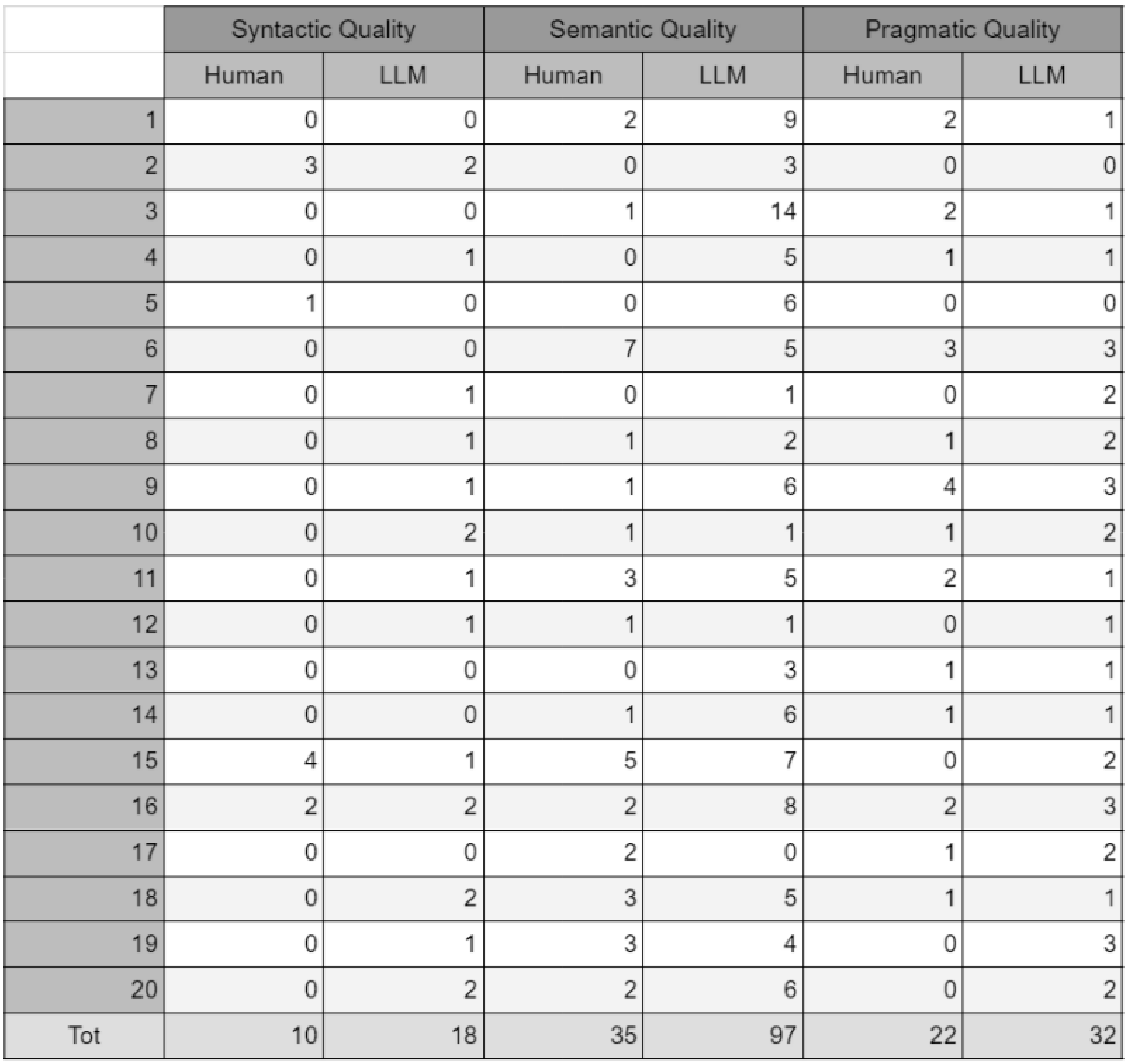
- Syntactic errors. Two syntax errors were found:
  1. *Attributes Ordered Incorrectly in Every Class.* This reflects a deviation from UML standards, which could affect the readability and standardization of the diagram.
  2. *Missing one Multiplicity in the Association Between the classes "Dish" and "Meal".* A syntactic omission that affects the clarity of the relationship between the two classes, potentially leading to confusion in interpreting the diagram.
- Semantic errors. One semantic error was found: *Incorrect Multiplicity in the Association Between the Classes "Ingredient" and "Dish".* This error misrepresents the relationship between ingredients and dishes (by considering a multiplicity of 1 to * instead of 0 to * on the class "Ingredient") which is crucial for understanding the makeup of dishes as specified by the exercise.
- Pragmatic errors. Two pragmatic errors were found:
  1. *Combination of Date and Time into a Single Attribute.* This design choice may reduce the diagram's usability for stakeholders who require distinct interactions with date and time data, as the exercise suggests the importance of a meal's specific timing.
  2. *Omission of 'Person' as a Parent Class for Classes Referring to Person.* The same error was also present in the Human-Created diagram.

4.1.2 Results first part

All the human-generated and LLM-generated diagrams were confronted with the given solutions to evaluate their quality. The results are reported in table 4.1.

Syntactic Quality

Throughout 20 exercises, in the human-generated diagrams, a total of 10 syntactic errors were made, while the LLM-generated diagram exhibited a slightly higher number, totalizing 18 errors. This may indicate a better



	Syntactic Quality		Semantic Quality		Pragmatic Quality	
	Human	LLM	Human	LLM	Human	LLM
1	0	0	2	9	2	1
2	3	2	0	3	0	0
3	0	0	1	14	2	1
4	0	1	0	5	1	1
5	1	0	0	6	0	0
6	0	0	7	5	3	3
7	0	1	0	1	0	2
8	0	1	1	2	1	2
9	0	1	1	6	4	3
10	0	2	1	1	1	2
11	0	1	3	5	2	1
12	0	1	1	1	0	1
13	0	0	0	3	1	1
14	0	0	1	6	1	1
15	4	1	5	7	0	2
16	2	2	2	8	2	3
17	0	0	2	0	1	2
18	0	2	3	5	1	1
19	0	1	3	4	0	3
20	0	2	2	6	0	2
Tot	10	18	35	97	22	32

Table 4.1: Each row represents a different exercise and for each of them are reported the number of errors both for Human and LLM for all the three types of quality.

adherence to UML standards by the human designer, a hypothesis that needs to be verified by further statistical analyses.

Semantic Quality

The evaluation of the semantic quality shows more evidently the difference between human actors and LLMs: human-generated diagrams accumulated 35 semantic errors across all exercises, whereas LLMs accounted for 97. This difference underscores the challenges faced by the LLM in accurately representing domain-specific knowledge and relationships within the diagrams, suggesting that LLMs may require deeper domain-specific knowledge and a better understanding of the semantics of information systems to match human performance.

Pragmatic Quality

In terms of pragmatic quality, which assesses the understandability of the diagrams, the human-generated diagrams contained 22 errors in total, compared to 32 for LLMs.

Further statistical analyses are necessary to understand if LLMs are behind human-generated diagrams in this area.

The aggregated data from 20 exercises, which are summarized in table 4.2 and in the figures 4.1 and 4.2, offers a robust comparison of human and LLM capabilities in UML class diagram creation. While humans generally produce higher-quality diagrams, the LLM's ability to generate diagrams that are syntactically and pragmatically correct to a certain extent is noteworthy.

	Syntactic Quality	Semantic Quality	Pragmatic Quality
Human	10	35	22
LLM	18	97	32

Table 4.2: Sums of errors for each Quality

The comparative analysis of the diagrams reveals that the human-generated diagrams, despite having errors, generally performed better than the LLM in all three quality dimensions.

The LLM's notably poorer performance in semantic quality, which had more than twice the number of errors compared to the human, suggests that while it can identify and use elements of UML, it struggles with the correct interpretation and application of these elements within specific domains.

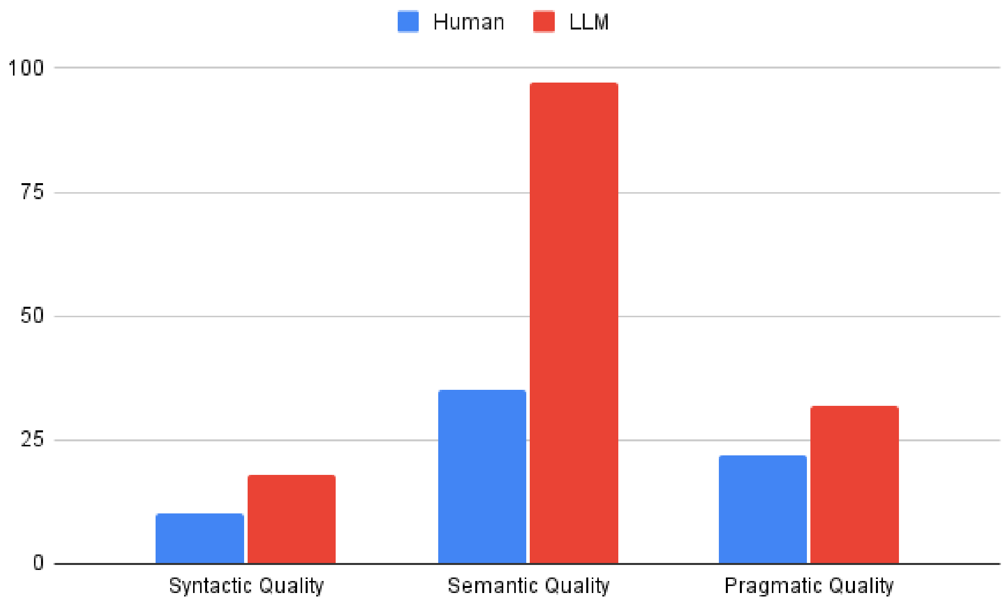


Figure 4.1: Number of errors divided in the three Qualities

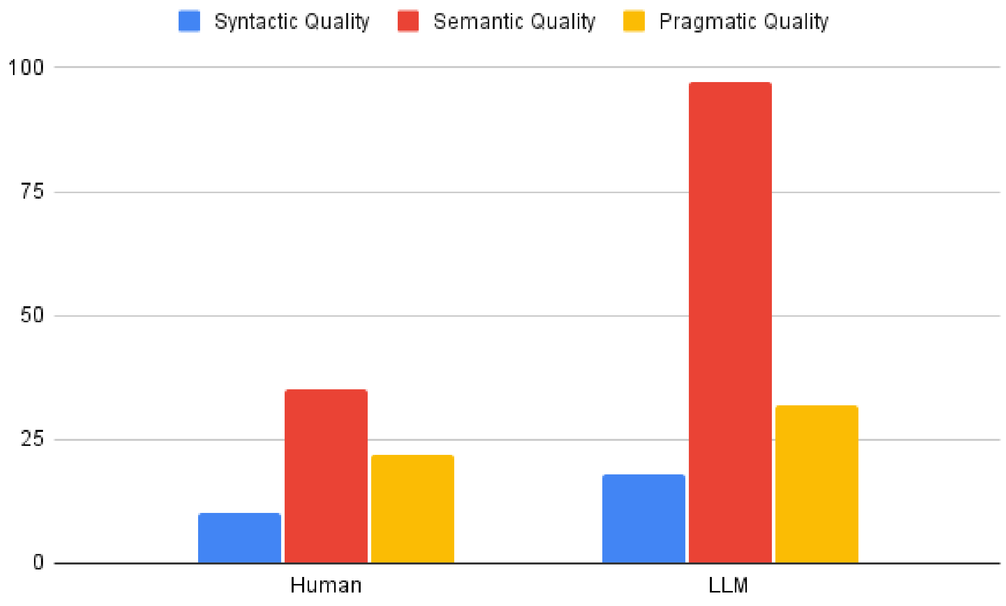


Figure 4.2: Number of errors divided between Human and LLM

Overall Evaluation

The first method's results illustrate the current capabilities and limitations of both human actors and LLMs in producing UML class diagrams: human-made diagrams were found to be more reliable, but not without faults; LLM-made diagrams, while showcasing the potential for automated diagram generation, revealed areas requiring significant enhancement, particularly in semantic representation.

4.2 Second Part

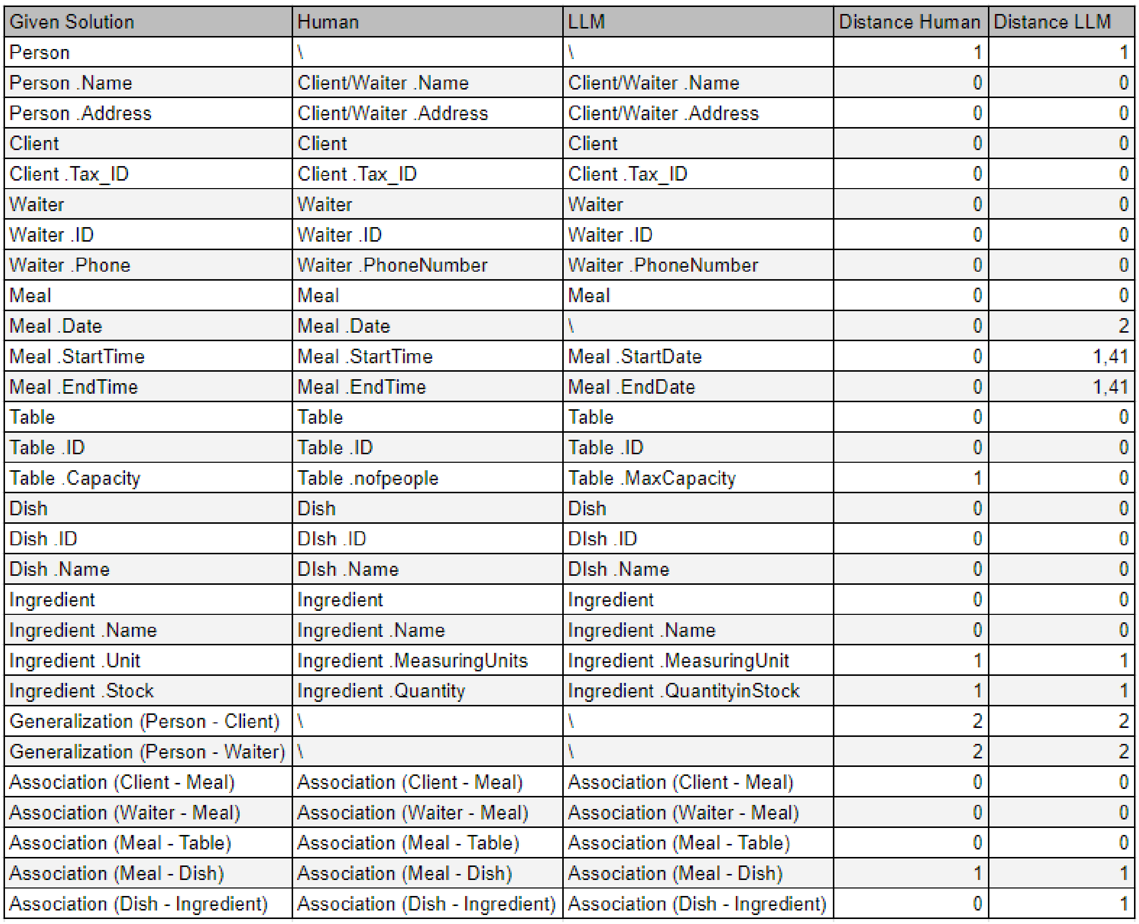
4.2.1 Example

An application of the second part of the analysis will be shown by using the aforementioned exercise, where the comparison algorithm was applied to evaluate the closeness of the diagrams generated by a human and the LLM to the provided solution. The analysis focused on the semantic content of elements and their relationships, crucial for ensuring accurate representation regardless of the naming conventions used. The results are shown in table 4.3.

The results of the analysis of the human-made and the LLM-generated diagram for the example exercise are both presented below.

Human-Created Diagram Analysis

- Distance in Classes. The "Person" class is missing in the human solution, making the distance equal to 1.
- Distance in Attributes.
  1. The attribute *nofpeople* is not present in the class *Table* but there is a semantically equivalent attribute named *capacity*; the temporary vector is <0|0|1|0> resulting in a distance of 1.
  2. The attribute *MeasuringUnits* of the class *Ingredient* is called *MeasuringUnit* by the human actor; the distance is thus 1.
  3. The attribute *Quantity* of the class *Ingredient* is called *Stock*, resulting in another distance equal to 1.
- Distance in Associations.



Given Solution	Human	LLM	Distance Human	Distance LLM
Person	\	\	1	1
Person .Name	Client/Waiter .Name	Client/Waiter .Name	0	0
Person .Address	Client/Waiter .Address	Client/Waiter .Address	0	0
Client	Client	Client	0	0
Client .Tax_ID	Client .Tax_ID	Client .Tax_ID	0	0
Waiter	Waiter	Waiter	0	0
Waiter .ID	Waiter .ID	Waiter .ID	0	0
Waiter .Phone	Waiter .PhoneNumber	Waiter .PhoneNumber	0	0
Meal	Meal	Meal	0	0
Meal .Date	Meal .Date	\	0	2
Meal .StartTime	Meal .StartTime	Meal .StartDate	0	1,41
Meal .EndTime	Meal .EndTime	Meal .EndDate	0	1,41
Table	Table	Table	0	0
Table .ID	Table .ID	Table .ID	0	0
Table .Capacity	Table .nofpeople	Table .MaxCapacity	1	0
Dish	Dish	Dish	0	0
Dish .ID	DIsh .ID	DIsh .ID	0	0
Dish .Name	DIsh .Name	DIsh .Name	0	0
Ingredient	Ingredient	Ingredient	0	0
Ingredient .Name	Ingredient .Name	Ingredient .Name	0	0
Ingredient .Unit	Ingredient .MeasuringUnits	Ingredient .MeasuringUnit	1	1
Ingredient .Stock	Ingredient .Quantity	Ingredient .QuantityinStock	1	1
Generalization (Person - Client)	\	\	2	2
Generalization (Person - Waiter)	\	\	2	2
Association (Client - Meal)	Association (Client - Meal)	Association (Client - Meal)	0	0
Association (Waiter - Meal)	Association (Waiter - Meal)	Association (Waiter - Meal)	0	0
Association (Meal - Table)	Association (Meal - Table)	Association (Meal - Table)	0	0
Association (Meal - Dish)	Association (Meal - Dish)	Association (Meal - Dish)	1	1
Association (Dish - Ingredient)	Association (Dish - Ingredient)	Association (Dish - Ingredient)	0	1

Table 4.3: Distances for each element in Exercise 10

1. *Generalization (Person - Client).* This generalization is missing, due to the absence of the "Person" class, meaning that the distance vector is <1|1|1|1>, leading to a distance of 2.
2. *Generalization (Person - Waiter).* Similarly to the error above, the distance is equal to 2.
3. *Association (Meal - Dish).* There is a difference in the multiplicity on the side of the "Dish" class (the solution expects a value of *0…**, while the human-generated diagram has a value of *1..**). The temporary vector equals <0|0|0|1> and the distance is 1.

- Distance in Operations. There were no differences in the operations between the human-made diagram and the reference solution.

The human diagram showed a total semantic distance of 3.61 from the given solution, indicating a relatively close semantic alignment.

LLM-Created Diagram Analysis

- Distance in Classes. The "Person" class is missing in the human solution, making the distance equal to 1.
- Distance in Attributes.
  1. The attribute *Date* is not present in the class *Meal*, meaning that the temporary vector is equal to <1|1|1|1> and the resulting distance is 2.
  2. The attribute *StartDate* of the class *Meal* has both a different name (Start Date) and a different attribute type; the resulting vector is <0|0|1|1> and the distance is approximately 1.41.
  3. The attribute *EndDate* of the class *Meal* presents the same errors as above, leading to a distance equal to approximately 1.41.
  4. The attribute *MeasuringUnits* of the class *Ingredient* is called *MeasuringUnit*; the distance is thus 1.
  5. The attribute *Quantity* of the class *Ingredient* is called *Stock*, resulting in another distance equal to 1.
- Distance in Associations.
  1. *Generalization (Person - Client).* This generalization is missing, due to the absence of the "Person" class, meaning that the distance vector is <1|1|1|1>, leading to a distance of 2.
  2. *Generalization (Person - Waiter).* Similarly to the error above, the distance is equal to 2.
  3. *Association (Dish - Ingredient).* There is a difference in the multiplicity on the side of the "Ingredient" class (the solution expects a value of *1...**, while the LLM-generated diagram has a value of **..**). The temporary vector equals <0|0|0|1> and the distance is 1.
  4. *Association (Meal - Dish).* The multiplicity is missing in the LLM-made diagram, making the temporary vector <0|0|0|1> and the distance equal to 1.
- Distance in Operations. There were no differences in the operations between the LLM-generated diagram and the reference solution.

The LLM diagram resulted in a higher total semantic distance of 4.58, suggesting a less accurate semantic representation compared to the human. The total distances for both Human and LLM are reported in table 4.4.

	Human	LLM
Total Distance	3,61	4,58

Table 4.4: Total Distances Exercise 10

Comparative Insight

Both diagrams managed to represent most classes, attributes, and methods with no semantic distance, implying a strong understanding of these elements. The human-created diagram generally had lower semantic distances across most elements, indicating a better grasp of the domain semantics compared to the LLM. The higher total distance of the LLM diagram reflects a need for improvement in its ability to semantically pair and accurately represent elements and their relationships.

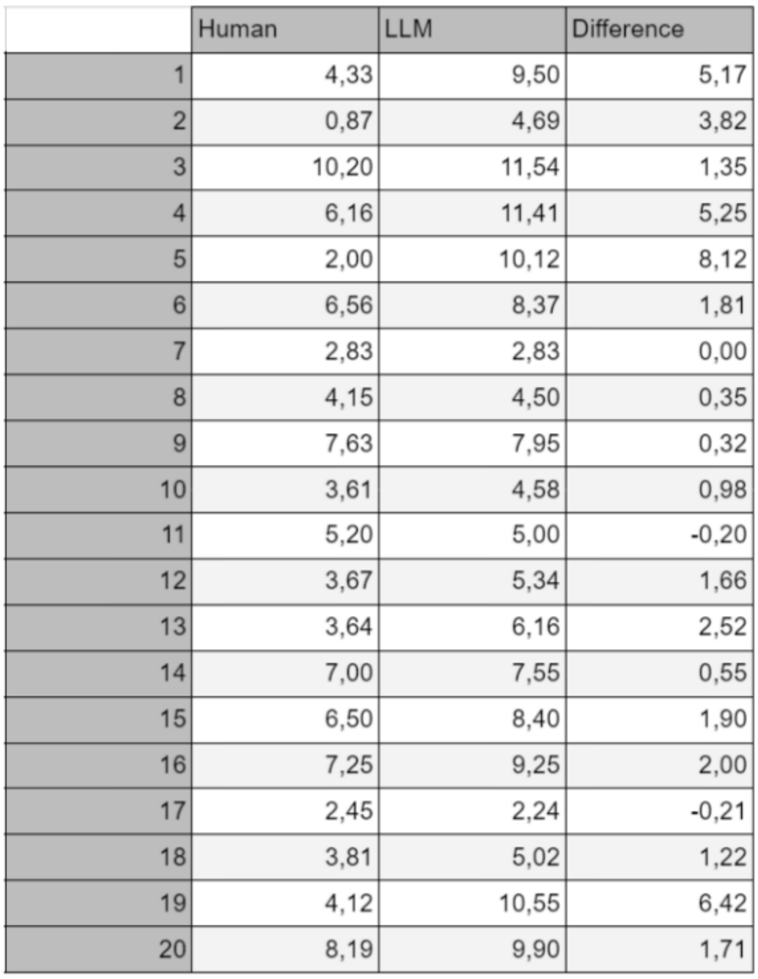
4.2.2 Results second part

The semantic distances from the reference solution have been calculated for all 20 exercises: Table 4.5 shows these distances, as well as the difference between the LLM distance and the human distance for each exercise; the average distances, derived as a general assessment of performance for each actor, are reported in Table 4.6.

General Observations:

Across the 20 exercises, human-generated diagrams had an average semantic distance of 5.01, while LLM-generated diagrams had a higher average distance of 7.25. This indicates that, on average, human-created diagrams were semantically closer to the given solution than those created by the LLM.

In several exercises, there is a great difference in semantic distance between human and LLM diagrams, with the human consistently closer to the solution. Notably, in Exercise 5 the difference between LLM and Human is the greatest (8.12) and in Exercise 2, the human diagram's distance is significantly lower (0.87) compared to the LLM's (4.69), indicating a more accurate semantic representation by the human.



	Human	LLM	Difference
1	4,33	9,50	5,17
2	0,87	4,69	3,82
3	10,20	11,54	1,35
4	6,16	11,41	5,25
5	2,00	10,12	8,12
6	6,56	8,37	1,81
7	2,83	2,83	0,00
8	4,15	4,50	0,35
9	7,63	7,95	0,32
10	3,61	4,58	0,98
11	5,20	5,00	-0,20
12	3,67	5,34	1,66
13	3,64	6,16	2,52
14	7,00	7,55	0,55
15	6,50	8,40	1,90
16	7,25	9,25	2,00
17	2,45	2,24	-0,21
18	3,81	5,02	1,22
19	4,12	10,55	6,42
20	8,19	9,90	1,71

Table 4.5: Distances from the given solutions, and differences

	Human	LLM	Difference
Average Distance	5,01	7,25	2,24

Table 4.6: Average of the distances and of the differences

There are, however, instances where the LLM performs comparably to the human and even outperforms him, as seen in Exercises 11 and 17, where the LLM's diagram has a slightly lower distance from the solution, or in Exercise 7, where there is no difference between the two diagrams.

Comparative Analysis:

The bar charts in figure 4.3 visually represent the variance between human and LLM diagrams in terms of semantic accuracy. In general, human diagrams show lower semantic distances across most exercises.

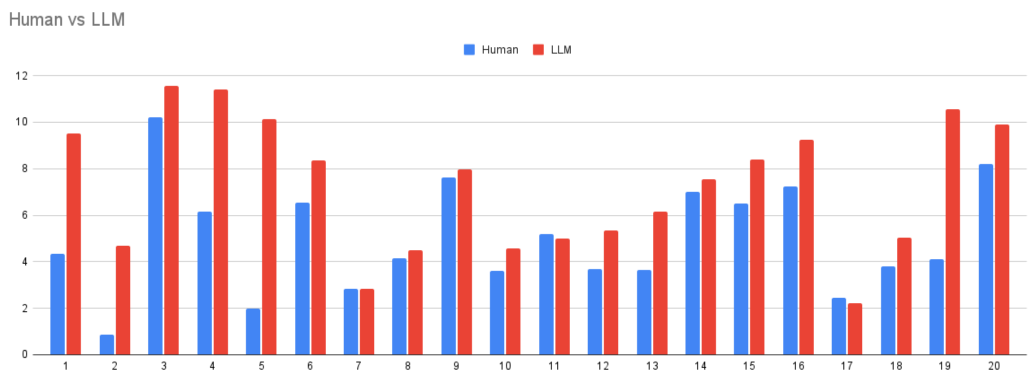


Figure 4.3: Bar chart of the distances.

The difference chart in figure 4.4 highlights exercises where the LLM particularly struggled in comparison to the human diagrams, with notable disparities in Exercises 1, 4, 5, and 19. These exercises might involve more complex semantic relationships that are challenging for the LLM to capture accurately.

Conclusion

The data indicates that, while the human-created diagrams are not without semantic inaccuracies, they tend to provide a more faithful representation of the intended domain semantics as defined by the given solutions.

The LLM shows potential in understanding and generating UML class diagrams but requires further improvements, particularly in its semantic interpretation capabilities.

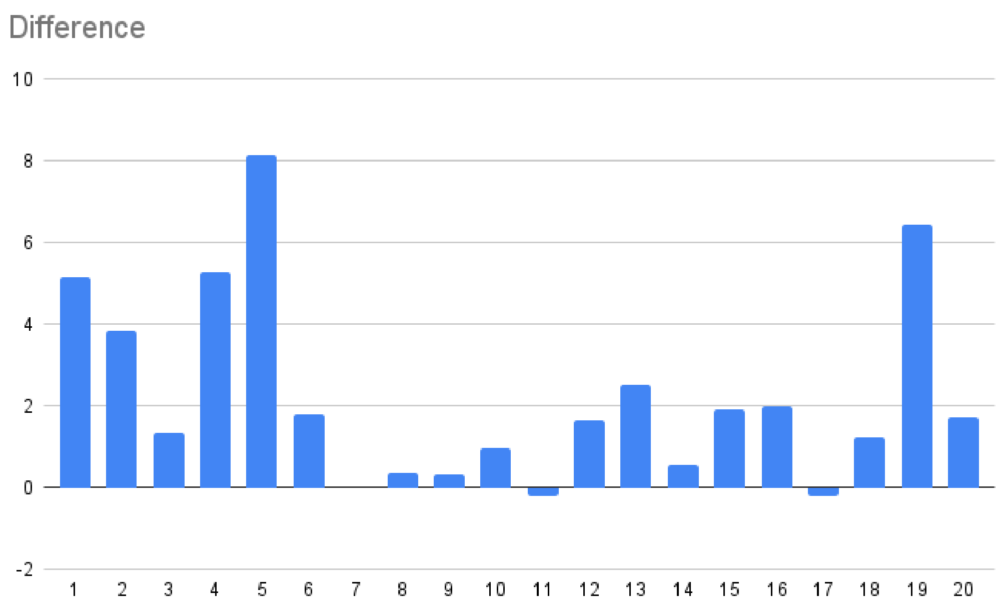


Figure 4.4: Bar chart reporting the difference.

4.3 Statistical Analysis

4.3.1 T-Test

In this section, the results of the paired samples t-test analysis will be analyzed. The analysis was made on the data gained from the two methodologies that have been applied. The most important results of this analysis are reported in table 4.7, while all the complete output from JASP can be found in the appendix C.

	t	p
Syntactic Quality	-1.506	0.148
Semantic Quality	-3.9806	< 0.001
Pragmatic Quality	-1.949	0.066
Distance	-4.277	< 0.001

Table 4.7: Results Paired Samples T-Test.

The most critical statistic is the p-value: a low p-value suggests that the observed difference is unlikely to have occurred by random chance, thus allowing the rejection of the null hypothesis. A p-value threshold of 0.05 is used to determine statistical significance, with a p-value below this threshold indicating that the difference between the groups is statistically significant.

Syntactic Quality

The p-value is 0.148 is above the threshold of 0.05, implying that there is no statistically significant difference between the diagrams produced by a human actor and the ones produced by the LLM regarding syntactic quality. It can be assumed that a Large Language Model is able to produce UML class diagrams that follow syntax conventions in a way similar to how a human would.

Semantic Quality

The p-value is lower than 0.001: such a low value denotes that there is a statistically significant difference between the diagrams produced by a human actor and the ones produced by the LLM regarding semantic quality. It can be assumed that a Large Language Model is not yet comparable to a human actor when it comes to semantic correctness as the model produces, on average, more errors.

Pragmatic Quality

The p-value is 0.066 and, since it is greater than the threshold of 0.05, there is no statistically significant difference between the diagrams produced by a human actor and the ones produced by the LLM regarding syntactic quality. It can be assumed that a Large Language Model is able to produce UML class diagrams that are comparable to those produced by a human actor, in terms of pragmatic quality.

Distances

The p-value is < 0.001 indicating that there is a statistically significant difference between the diagrams produced by a human actor and the ones produced by the LLM regarding distance to a reference solution. It can be assumed that a Large Language Model is not able to produce UML diagrams that can be compared to those produced by a human actor according to the second method, as the LLM produces, on average, diagrams that are more distant from the reference solution and have, as a consequence, more errors.

4.3.2 Linear Regression

Data

To generate the linear regression it is necessary to calculate and report in the tables the necessary data.

The first step consists of gathering the total amount of Classes, Attributes, and Associations from the reference solutions: Operations were considered as Attributes to facilitate the calculation, and also because not all of the exercises had Operations in their solution. Table 4.8 contains the total sum of these entities for each exercise and, in the last row, the average values of each column, giving an insight into the average complexity of the exercises.

	Classes	Attributes	Associations
1	6	17	7
2	6	35	5
3	7	12	8
4	6	31	7
5	8	18	11
6	9	12	12
7	6	5	7
8	7	11	8
9	9	13	10
10	7	15	7
11	6	11	9
12	6	11	5
13	8	19	8
14	5	12	5
15	8	6	8
16	11	13	13
17	8	5	8
18	6	10	6
19	9	20	9
20	10	28	10
Average	7.4	15.2	8.15

Table 4.8: Data from the given solutions.

The second step consists of calculating, using a sample of three people, the Estimated Difficulties (EDs). The Estimated Difficulty is an increasing scale that goes from 1 to 5, where 1 indicates that the exercise is perceived as very easy by the person and 5 is perceived as very hard. The EDs were decided before carrying out the exercise (so they were based only on a first impact with the text).

Average values have been calculated for each person (indicating how each evaluator perceived the complexity of the set of exercises) and for each exercise; the average values per exercise are those that have been used to perform the linear regression, to mitigate the potential impact of the text misinterpretation of a single individual. All the EDs are reported in table 4.9.

	ED 1	ED 2	ED 3	AVG ED
1	3	2	2	2.33
2	2	1	1	1.33
3	5	5	4	4.67
4	4	3	3	3.33
5	2	2	3	2.33
6	4	4	3	3.67
7	1	1	1	1
8	2	2	2	2
9	3	3	2	2.67
10	2	3	3	2.67
11	3	4	3	3.33
12	2	3	2	2.33
13	2	1	2	1.67
14	3	2	3	2.67
15	4	5	4	4.33
16	4	3	2	3
17	3	2	2	2.33
18	2	2	1	1.67
19	4	4	4	4
20	5	5	4	4.67
AVG	3	2.85	2.55	2.8

Table 4.9: Estimated Difficulties.

The complete JASP output of the linear regression can be found in appendix D.

In this section, the equations of the linear regression for each dependent variable will be analyzed, together with a table containing the p-value for each independent variable.

The dependent variables that have been analyzed are:

- Y_1 = number of Syntactic Errors in Human-Generated diagrams.
- Y_2 = number of Syntactic Errors in LLM-Generated diagrams.
- Y_3 = number of Semantic Errors in Human-Generated diagrams.
- Y_4 = number of Semantic Errors in LLM-Generated diagrams.
- Y_5 = number of Pragmatic Errors in Human-Generated diagrams.
- Y_6 = number of Pragmatic Errors in LLM-Generated diagrams.
- Y_7 = Distance of the Human-Generated diagrams from the given solution.
- Y_8 = Distance of the LLM-Generated diagrams from the given solution.

The independent variables are:

- X_1 = number of classes in the given solution.
- X_2 = number of Attributes (and operations) in the given solution.
- X_3 = number of associations in the given solution.
- X_4 = average estimated difficulty of the exercise.

ϵ is the error term, which captures the variation in Y not explained by the model.

Human Syntactic Errors

The equation of the Linear Regression is the following:

$$Y_1 = -0.664 + 0.311X_1 + 0.011X_2 - 0.161X_3 + 0.005X_4 + \epsilon \tag{4.1}$$

The p-values for each independent variable are reported in table 4.10.

None of the independent variables show a statistically significant relationship with the dependent variable, as their p-values are all well above the 0.05 threshold.

Model		p
H_1	(Intercept)	0.652
	X_1	0.391
	X_2	0.765
	X_3	0.530
	X_4	0.988

Table 4.10: P-values Human Syntactic Errors

Given the lack of statistically significant predictors in this model, it is possible to conclude that these particular independent variables do not have a significant predictive power on the number of human syntactic errors.

This regression output suggests that further investigation may be necessary to find a model that better fits the data. It may also imply that other variables not included in the model could be influencing the number of errors, or that the relationship between the variables is not linear.

LLM Syntactic Errors

The equation of the Linear Regression is the following:

$$Y_2 = -0.057 + 0.337X_1 + 0.023X_2 - 0.198X_3 - 0.099X_4 + \epsilon \qquad (4.2)$$

The p-values for each independent variable are reported in table 4.11.

Model		p
H_1	(Intercept)	0.951
	X_1	0.149
	X_2	0.324
	X_3	0.230
	X_4	0.608

Table 4.11: P-values LLM Syntactic Errors

None of the p-values is below the threshold of 0.05, suggesting that there is no statistically significant evidence that they individually affect the number of LLM syntactic errors.

According to this model, these predictors may not be sufficient to explain the variation in LLM syntactic errors, or the relationships may not be linear.

Other variables not included in the model might be influencing the syntactic errors.

Human Semantic Errors

The equation of the Linear Regression is the following:

$$Y_3 = -0.269 - 0.029X_1 - 0.079X_2 + 0.161X_3 + 0.757X_4 + \epsilon \tag{4.3}$$

The p-values for each independent variable are reported in table 4.12.

Model		p
H$_1$	(Intercept)	0.884
	X$_1$	0.948
	X$_2$	0.101
	X$_3$	0.615
	X$_4$	0.062

Table 4.12: P-values Human Semantic Errors

None of the predictors are statistically significant at the conventional 0.05 level, but X$_4$ is close to this threshold, having a p-value of 0.062. This means that, while the association between the average estimated difficulties and the human semantic errors is not statistically significant by conventional standards, there could be a potential relationship with a larger sample size or a different sample.

LLM Semantic Errors

The equation of the Linear Regression is the following:

$$Y_4 = 1.241 - 1.085X_1 + 0.032X_2 + 0.743X_3 + 1.817X_4 + \epsilon \tag{4.4}$$

The p-values for each independent variable are reported in table 4.13.

The independent variable X$_4$ shows significance in this model having a p-value of 0.018 which is less than 0.05. So, it is possible to conclude that the average estimated difficulty (AVG ED) is a significant predictor of the number of LLM semantic errors, while the other variables are not. The significance of the AVG ED suggests that it should be a point of focus for understanding or improving the LLM's performance regarding semantic errors.

Model		p
H_1	(Intercept)	0.712
	X_1	0.197
	X_2	0.698
	X_3	0.214
	X_4	0.018

Table 4.13: P-values LLM Semantic Errors

Human Pragmatic Errors

The equation of the Linear Regression is the following:

$$Y_5 = 0.500 - 0.330X_1 - 0.019X_2 + 0.399X_3 + 0.027X_4 + \epsilon \tag{4.5}$$

The p-values for each independent variable are reported in table 4.14.

Model		p
H_1	(Intercept)	0.696
	X_1	0.296
	X_2	0.554
	X_3	0.086
	X_4	0.920

Table 4.14: P-values Human Pragmatic Errors

None of the predictors have p-values below the significance threshold of 0.05, indicating no significant association.

The model does not find any of the predictors to be statistically significant in determining the number of human pragmatic errors. This suggests that either these specific independent variables do not have a strong influence on human pragmatic errors, or the relationship may not be linear.

LLM Pragmatic Errors

The equation of the Linear Regression is the following:

$$Y_6 = -0.792 + 0.485X_1 - 0.045X_2 - 0.091X_3 + 0.079X_4 + \epsilon \tag{4.6}$$

Model		p
H_1	(Intercept)	0.326
	X_1	0.022
	X_2	0.035
	X_3	0.513
	X_4	0.633

Table 4.15: P-Values LLM Pragmatic Errors

The p-values for each independent variable are reported in table 4.15.

The positive coefficient for X_1 (0.485) is significant ($p = 0.022$), implying that an increase in the number of classes is associated with an increase in LLM pragmatic errors.

The negative coefficient for X_2 (-0.045) is significant ($p = 0.035$), indicating that an increase in the number of attributes is associated with a decrease in LLM pragmatic errors. While statistically significant, the small value of the coefficient suggests that the practical impact of this variable on LLM pragmatic errors may be limited.

The number of classes has a significant impact on the number of LLM pragmatic errors. Associations and estimated difficulty, however, do not appear to significantly affect them.

Distance Human

The equation of the Linear Regression is the following:

$$Y_7 = 0.915 - 0.120X_1 - 0.038X_2 + 0.105X_3 + 1.675X_4 + \epsilon \tag{4.7}$$

The p-values for each independent variable are reported in table 4.16.

The coefficient for X_4 is positive (1.675) and is statistically significant ($p < 0.001$), indicating a strong association between the average estimated difficulty and the distance of the human-generated diagram from the given solution. As the average estimated difficulty increases, there is a corresponding significant increase in the distance. The significant p-value for the difficulty implies that it is an influential factor in the model and it is necessary to focus on it to understand or reduce the distance from the given solution.

Model		p
H_1	(Intercept)	0.648
	X_1	0.805
	X_2	0.453
	X_3	0.761
	X_4	< 0.001

Table 4.16: P-Values Human Distances

Distance LLM

The equation of the Linear Regression is the following:

$$Y_8 = -0.005 - 0.548X_1 + 0.139X_2 + 0.567X_3 + 1.635X_4 + \epsilon \tag{4.8}$$

The p-values for each independent variable are reported in table 4.17.

Model		p
H_1	(Intercept)	0.998
	X_1	0.324
	X_2	0.024
	X_3	0.160
	X_4	0.003

Table 4.17: P-Value LLM Distances

The coefficient for X_2 is positive (0.139) and significant ($p = 0.024$), indicating that an increase in the number of attributes is associated with an increase in the distance of LLM from the given solution. Moreover, the coefficient for X_4 is positive (1.635) and significant ($p = 0.003$), showing a strong association with the LLM syntactic error distance. This implies that as the average estimated difficulty of the UML diagrams increases, so does the distance from the given solution.

4.3.3 Conclusion

The statistical analysis described in this section explored the capabilities of LLMs to generate diagrams that adhere to the standards and how they performed when compared to those produced by human actors.

The paired samples t-test analysis found that when it comes to syntactic and pragmatic qualities, the LLM is very similar to a human. This suggests that LLMs have achieved a level of proficiency in following syntactic rules and producing understandable diagrams from a pragmatic point of view. However, the analysis showed a different picture of the semantic quality and the distance metric, with a statistically significant discrepancy between human and LLM-generated diagrams. This implies that LLMs lag in capturing the nuances and contextual accuracy required for semantically correct diagrams.

The linear regression model offered an additional layer of understanding. It is important to notice that the average estimated difficulty emerged as a significant predictor for the distance of diagrams from the given solution, suggesting that, as the complexity of the task increases, so does the difficulty for LLMs to produce error-free diagrams. This relationship was consistently observed across both human and LLM-generated diagrams, indicating that complexity remains a barrier to the accuracy and reliability of automated UML diagram generation.

In conclusion, while LLMs excel in understanding the structural rules (syntax) and practical usage (pragmatics), they have difficulties when it comes to the deeper understanding necessary for semantic accuracy and adherence to complex specifications. These insights not only affirm the potential of LLMs in assisting with software design tasks but also delineate the boundaries of their current abilities.

Chapter 5

Conclusions and future developments

In this thesis, the efficiency of Large Language Models (LLMs) in the creation of Unified Modeling Language (UML) was examined in contrast to manually created diagrams. The comparative analysis used a methodology that included both qualitative and quantitative analyses, also allowing for the analysis of the syntactic, semantic, and pragmatic qualities of the diagrams.

The study revealed that LLMs can generate UML class diagrams with pragmatic and syntactic accuracy comparable to humans, although with a difference in semantic precision, emphasizing the difficulties for LLMs to represent the complex semantic link necessary to accurately realize class diagrams. These results suggest that are necessary more developments in AI technology to fully automate diagram creation.

Both implications in theory and practice can be made from this research. From a practical point of view, the study shows that LLMs can support iterative design processes and quick prototyping of UML class diagrams. Instead, from a theoretical view, it prepares the ground for further advancements in AI-driven design techniques, emphasizing the need for more sophisticated models with a deeper comprehension of semantic links.

Future Developments

Although the focus of this study is limited to class diagrams, similar techniques might be used for other types of UML diagrams, including sequence, activity, and state diagrams, providing insight into the flexibility and application of LLMs at various stages of the software development lifecycle. Additionally, as AI technology improves, assessing more sophisticated and recent language models may show gains in semantic comprehension and UML diagram production accuracy, solving the issues brought to light by the findings of this study. Additional research is needed to determine how to incorporate LLM-generated UML diagrams into software development processes and tools in order to improve automation, effectiveness, and cooperation amongst development teams.

Developments in AI and machine learning will influence software design and the creation of UML class diagrams. Also, the improvements in natural language comprehension will close the semantic gap now in existence, producing diagrams of greater quality. AI technologies have the potential to be perfectly incorporated into software development in the future. They could provide real-time UML diagram creation, updates, and suggestions based on ongoing project progress and code development, making the software development's design process more effective and error-free. To understand the potential advantages of integrating artificial intelligence (AI) into software engineering methodologies, more studies are needed. [28]

The use of iterative LLMs to improve the creation and improvement of UML class diagrams is one of the most important areas for future study. Iterative LLMs can participate in a feedback loop, where the output of an initial model prompt is refined through successive iterations to improve accuracy and relevance. Examples of these LLMs are those covered in the study "GPT-4 Doesn't Know It's Wrong: An Analysis of Iterative Prompting for Reasoning Problems" (2023)[29]. The semantic gap found in this thesis may be resolved with the help of this iterative method. Future LLMs might be trained to create UML diagrams and then iteratively modify them based on semantic input by employing iterative prompting approaches.

Final Thoughts

Large Language Model integration has great potential long-term effects on software design processes. The software development lifecycle may be streamlined by LLMs as they grow further because of their capacity to comprehend, model, and produce complex software design objects. Along with increasing efficiency, this integration will make software design more accessible to anyone, making complex design jobs more approachable for those with less technical know-how.

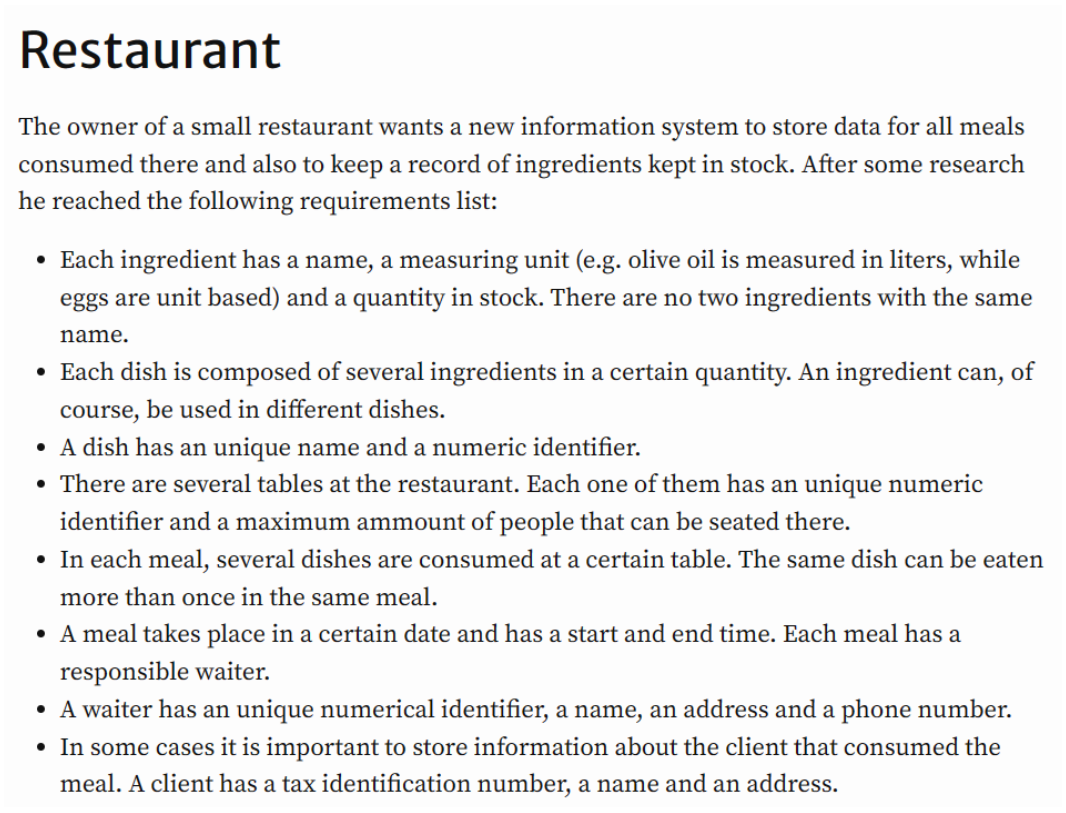
Appendix A

Collection of Exercises and Calculation

The collection with the twenty exercises (for each exercise is given the text, given solution, human solution, LLM solution, PlantUML, source-link) and the Excel with all the calculations can be found in the following link: https://figshare.com/articles/dataset/Thesis_Database/25316440.

Appendix B

Exercise 10



Restaurant

The owner of a small restaurant wants a new information system to store data for all meals consumed there and also to keep a record of ingredients kept in stock. After some research he reached the following requirements list:

- Each ingredient has a name, a measuring unit (e.g. olive oil is measured in liters, while eggs are unit based) and a quantity in stock. There are no two ingredients with the same name.
- Each dish is composed of several ingredients in a certain quantity. An ingredient can, of course, be used in different dishes.
- A dish has an unique name and a numeric identifier.
- There are several tables at the restaurant. Each one of them has an unique numeric identifier and a maximum ammount of people that can be seated there.
- In each meal, several dishes are consumed at a certain table. The same dish can be eaten more than once in the same meal.
- A meal takes place in a certain date and has a start and end time. Each meal has a responsible waiter.
- A waiter has an unique numerical identifier, a name, an address and a phone number.
- In some cases it is important to store information about the client that consumed the meal. A client has a tax identification number, a name and an address.

Figure B.1: Text Exercise 10

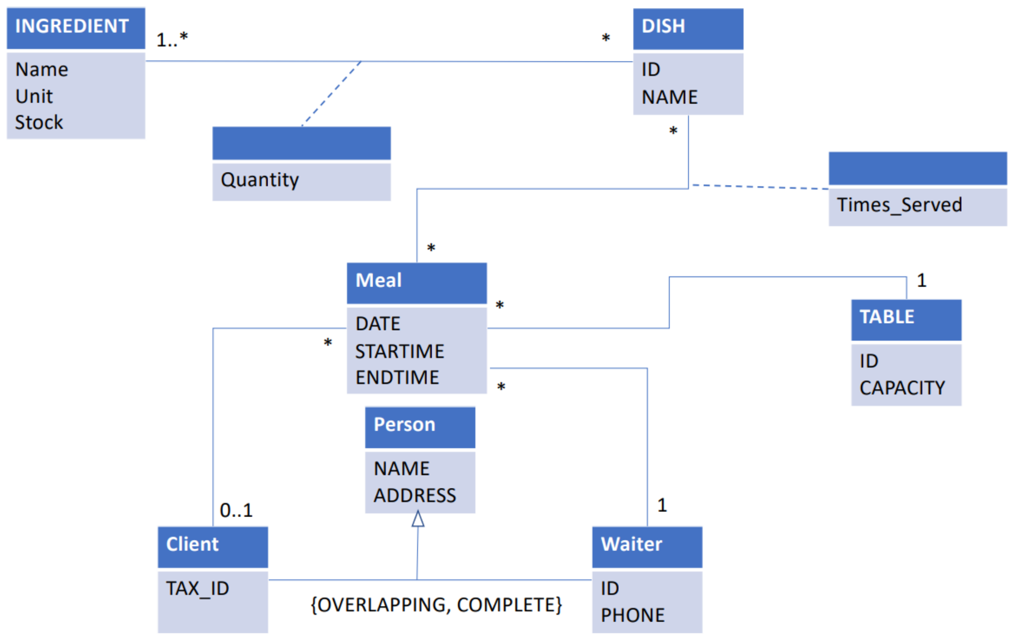


Figure B.2: Given-solution of exercise 10

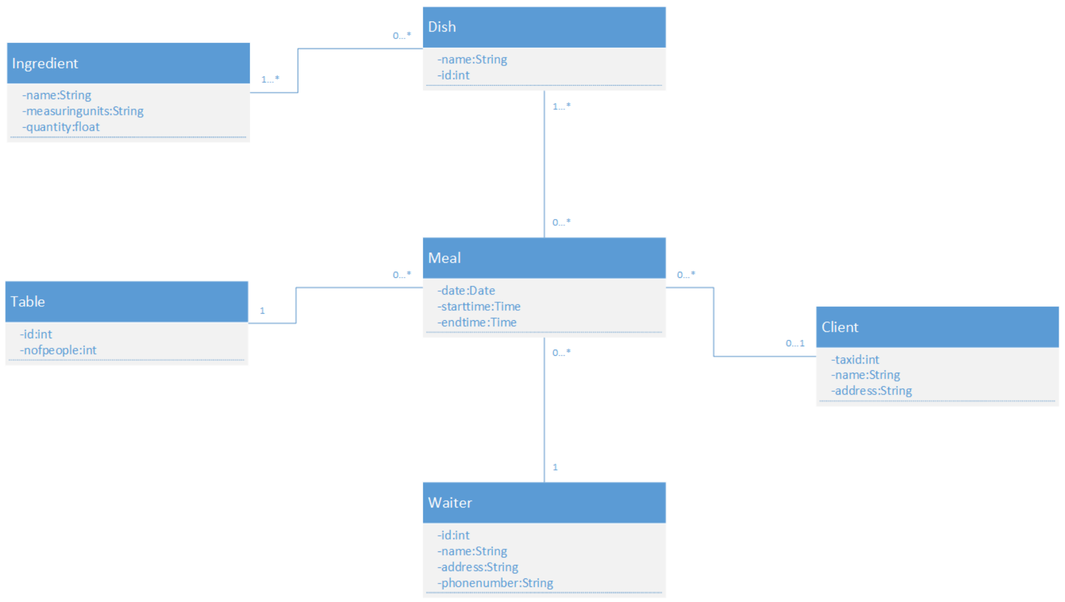


Figure B.3: Human solution of exercise 10

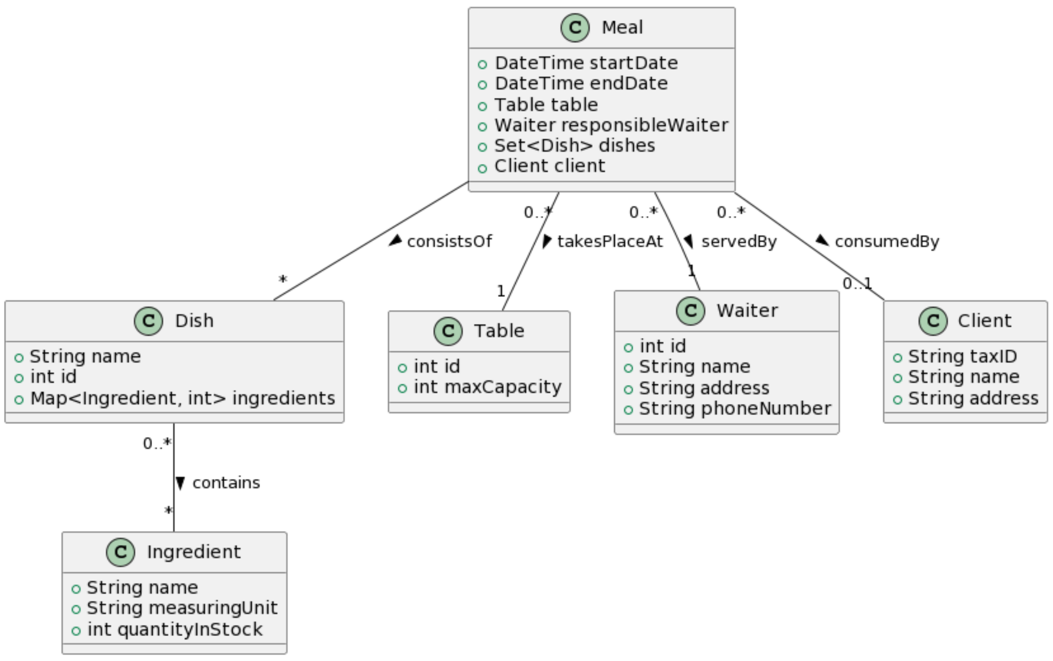


Figure B.4: LLM solution of exercise 10

Appendix C

T-Test

C.1 First Method

C.1.1 Syntactic Quality

Summary statistics for the syntactic quality, depending on the actor who performed the exercise (Human or LLM), are in table C.1.

	N	Mean	SD	SE	Coefficient of variation
Human Syntactic Errors	20	0.500	1.147	0.256	2.294
LLM Syntactic Errors	20	0.900	0.788	0.176	0.876

Table C.1: Descriptives

T-Test Results for the syntactic quality are in table C.2.

Measure 1		Measure 2	t	df	p
Human Syntactic Errors	-	LLM Syntactic Errors	-1.506	19	0.148

Table C.2: Paired Samples T-Test

The Barplot for the syntactic quality is in figure C.1.

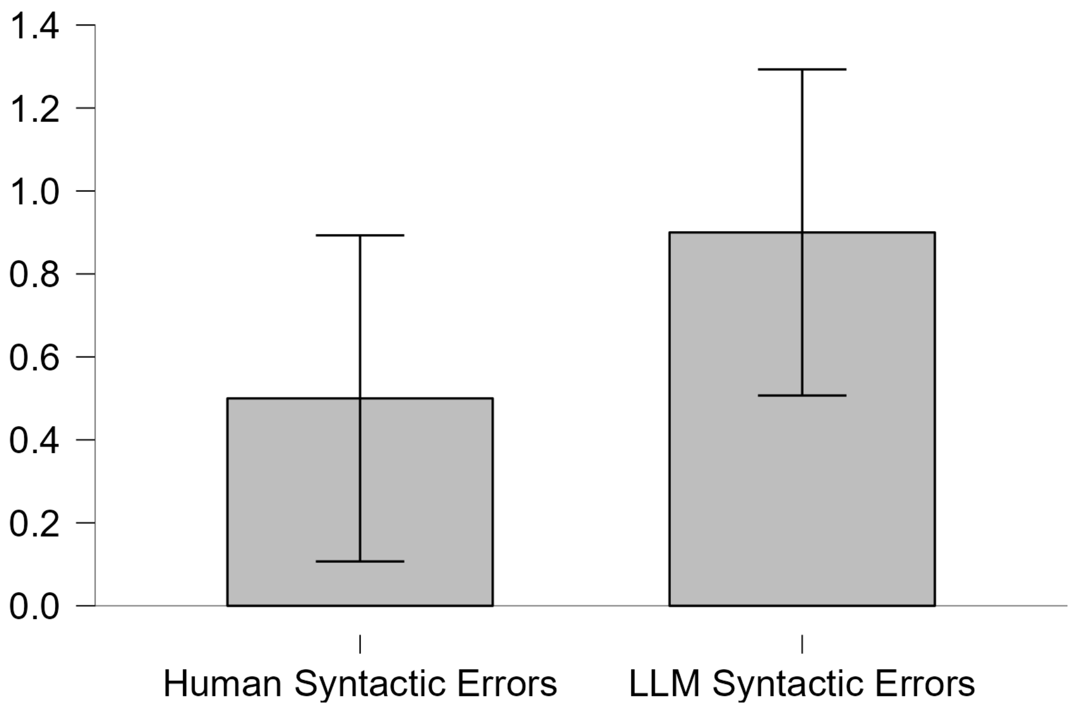


Figure C.1: Bar-plot Syntactic Quality

C.1.2 Semantic Quality

Summary statistics for the semantic quality, depending on the actor who performed the exercise (Human or LLM), are in table C.3.

	N	Mean	SD	SE	Coefficient of variation
Human Semantic Errors	20	1.750	1.803	0.403	1.030
LLM Semantic Errors	20	4.850	3.281	0.734	0.677

Table C.3: Descriptives

T-test Results for the semantic quality are in table C.4.

Measure 1		Measure 2	t	df	p
Human Semantic Errors	-	LLM Semantic Errors	-3.986	19	$< .001$

Table C.4: Paired Samples T-Test

The Barplot for the Semantic quality is in figure C.2.

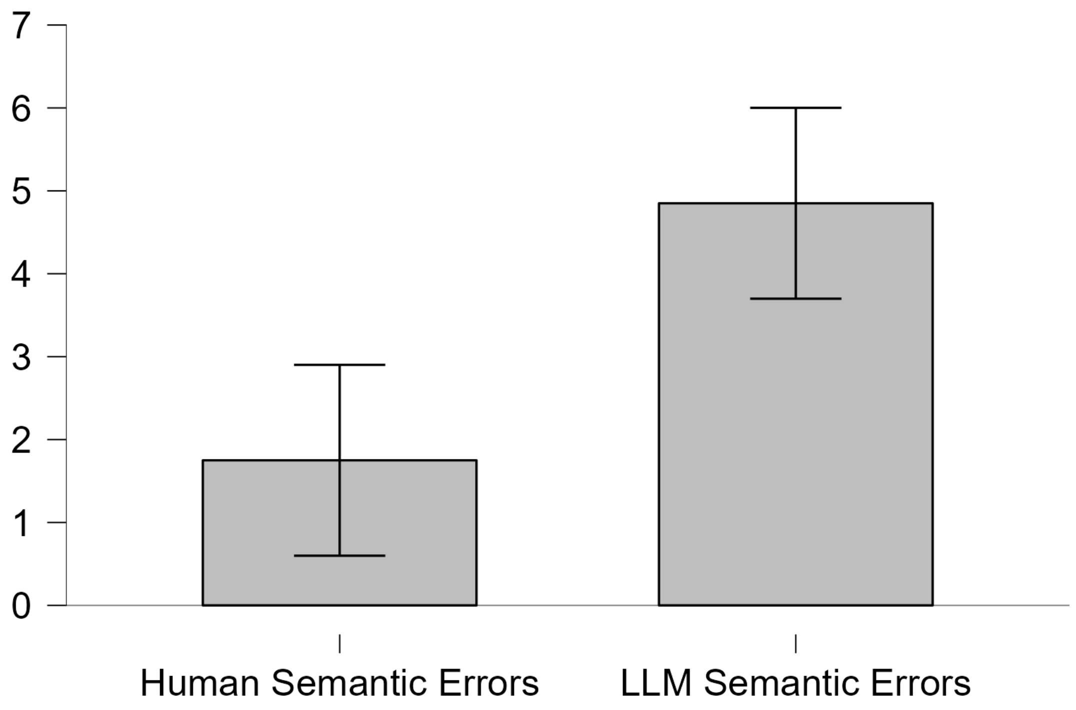


Figure C.2: Barplot Semantic Quality

C.1.3 Pragmatic Quality

Summary statistics for the pragmatic quality, depending on the actor who performed the exercise (Human or LLM), are in table C.5.

	N	Mean	SD	SE	Coefficient of variation
Human Pragmatic Errors	20	1.100	1.119	0.250	1.017
LLM Pragmatic Errors	20	1.600	0.940	0.210	0.588

Table C.5: Descriptives

T-Test Results for the pragmatic quality are in table C.6.

Measure 1		Measure 2	t	df	p
Human Pragmatic Errors	-	LLM Pragmatic Errors	-1.949	19	0.066

Table C.6: Paired Samples T-Test

The Barplot for the pragmatic quality is in figure C.3.

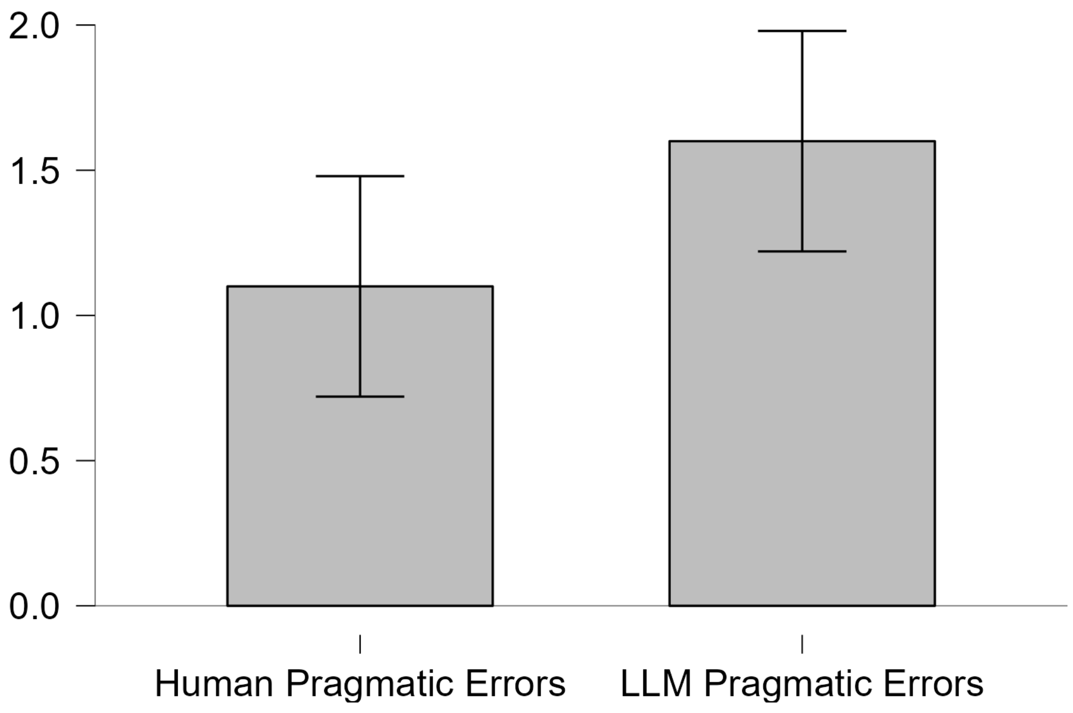


Figure C.3: Barplot Pragmatic Quality

C.2 Second Method

Summary statistics for the distance to a reference solution, depending on the actor who performed the exercise (Human or LLM), are in table C.7.

	N	Mean	SD	SE	Coefficient of variation
Distance Human	20	5.008	2.356	0.527	0.470
Distance LLM	20	7.245	2.855	0.638	0.394

Table C.7: Descriptives

T-Test Results for the pragmatic quality are in table C.8.

Measure 1		Measure 2	t	df	p
Distance Human	-	Distance LLM	−4.277	19	< .001

Table C.8: Paired Samples T-Test

The Barplot for the distance to a reference solution is in figure C.4.

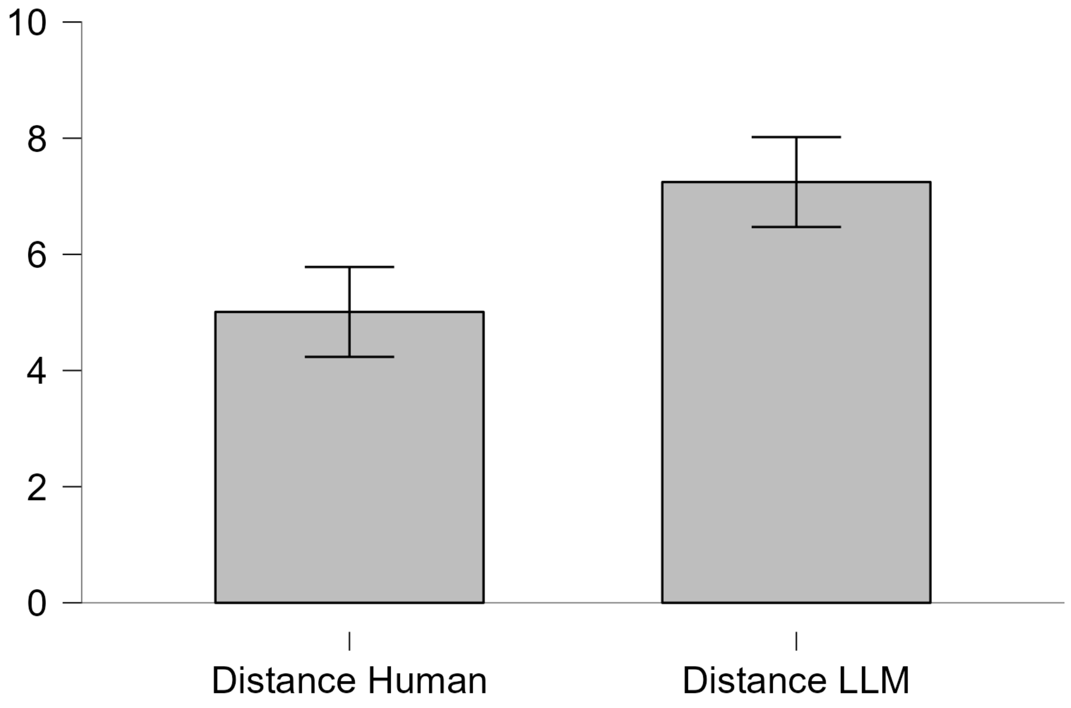


Figure C.4: Barplot for the distance to a reference solution

Appendix D

Linear Regression

D.1 First Part

Human Syntactic Errors

Model	R	R^2	Adjusted R^2	RMSE
H_0	0.000	0.000	0.000	1.147
H_1	0.261	0.068	−0.181	1.246

Table D.1: Model Summary - Human Syntactic Errors

Model		Unstandardized	Standard Error	Standardized	t	p
H_0	(Intercept)	0.500	0.256		1.949	0.066
H_1	(Intercept)	−0.664	1.445		−0.460	0.652
	Classes	0.311	0.351	0.434	0.884	0.391
	Attributes	0.011	0.036	0.078	0.305	0.765
	Associations	−0.161	0.251	−0.313	−0.643	0.530
	AVG ED	0.005	0.300	0.004	0.015	0.988

Table D.2: Coefficients

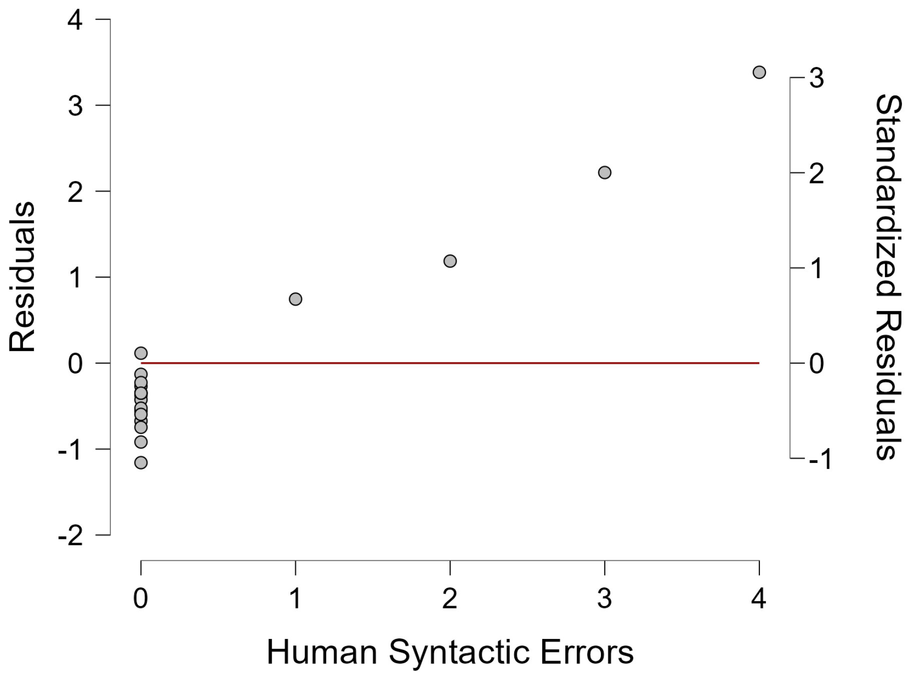


Figure D.1: Residuals Vs. Human Syntactic Errors

LLM Syntactic Errors

Model	R	R^2	Adjusted R^2	RMSE
H_0	0.000	0.000	0.000	0.788
H_1	0.463	0.215	0.005	0.786

Table D.3: Model Summary - LLM Syntactic Errors

Model		Unstandardized	Standard Error	Standardized	t	p
H_0	(Intercept)	0.900	0.176		5.107	$< .001$
H_1	(Intercept)	−0.057	0.911		−0.062	0.951
	Classes	0.337	0.221	0.686	1.523	0.149
	Attributes	0.023	0.023	0.240	1.021	0.324
	Associations	−0.198	0.158	−0.560	−1.253	0.230
	AVG ED	−0.099	0.189	−0.134	−0.523	0.608

Table D.4: Coefficients

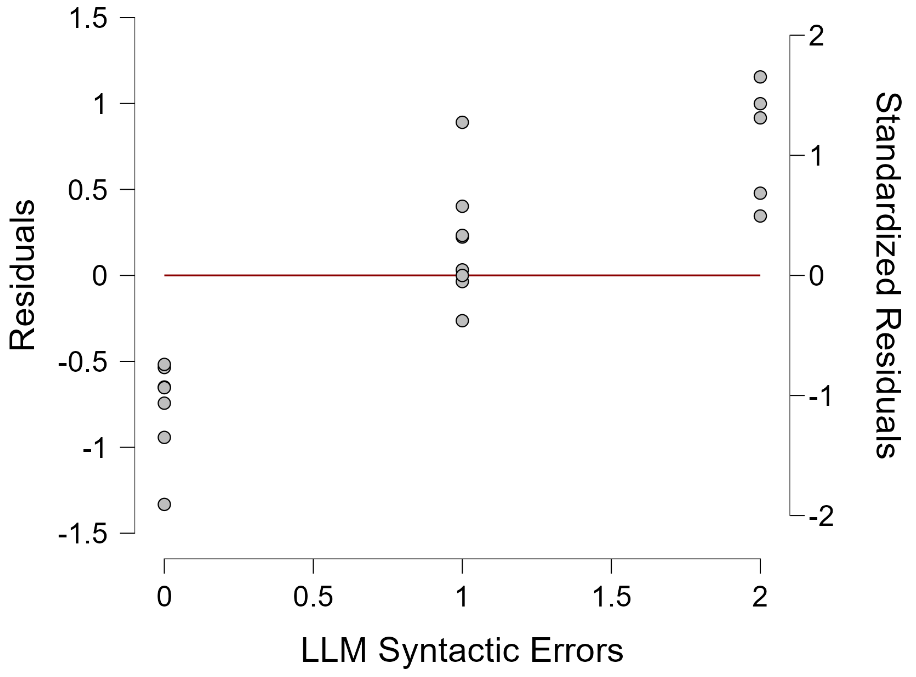


Figure D.2: Residuals Vs. LLM Syntactic Errors

Human Semantic Errors

Model	R	R^2	Adjusted R^2	RMSE
H_0	0.000	0.000	0.000	1.803
H_1	0.641	0.410	0.253	1.558

Table D.5: Model Summary - Human Semantic Errors

Model		Unstandardized	Standard Error	Standardized	t	p
H_0	(Intercept)	1.750	0.403		4.341	$< .001$
H_1	(Intercept)	−0.269	1.806		−0.149	0.884
	Classes	−0.029	0.439	−0.026	−0.067	0.948
	Attributes	−0.079	0.045	−0.356	−1.748	0.101
	Associations	0.161	0.313	0.199	0.513	0.615
	AVG ED	0.757	0.375	0.448	2.017	0.062

Table D.6: Coefficients

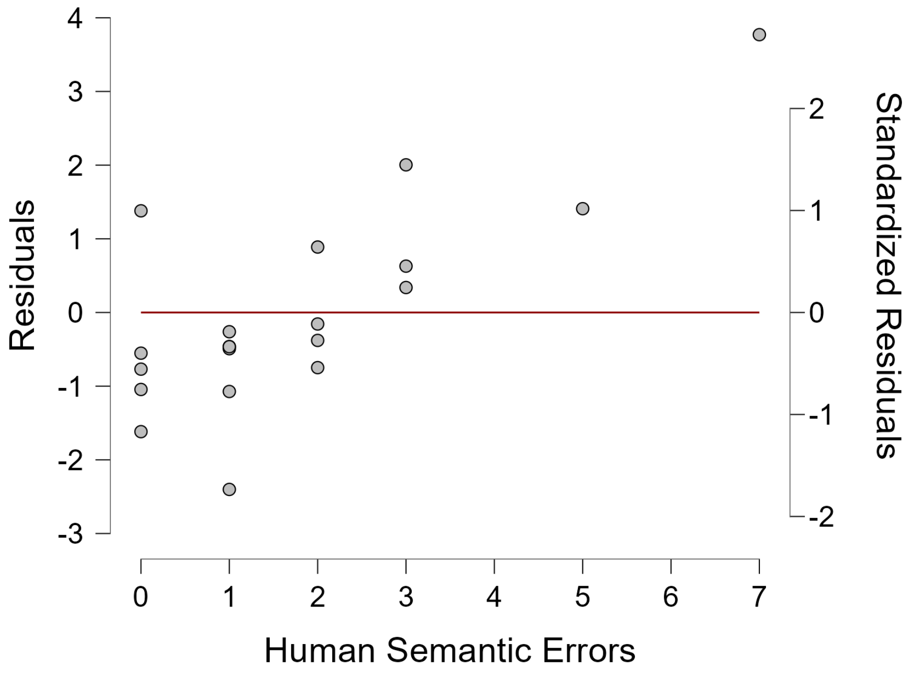


Figure D.3: Residuals Vs. Human Semantic Errors

LLM Semantic Errors

Model	R	R^2	Adjusted R^2	RMSE
H_0	0.000	0.000	0.000	3.281
H_1	0.636	0.405	0.246	2.850

Table D.7: Model Summary - LLM Semantic Errors

Model		Unstandardized	Standard Error	Standardized	t	p
H_0	(Intercept)	4.850	0.734		6.610	< .001
H_1	(Intercept)	1.241	3.304		0.376	0.712
	Classes	−1.085	0.803	−0.530	−1.351	0.197
	Attributes	0.032	0.082	0.081	0.396	0.698
	Associations	0.743	0.573	0.505	1.297	0.214
	AVG ED	1.817	0.686	0.591	2.648	0.018

Table D.8: Coefficients

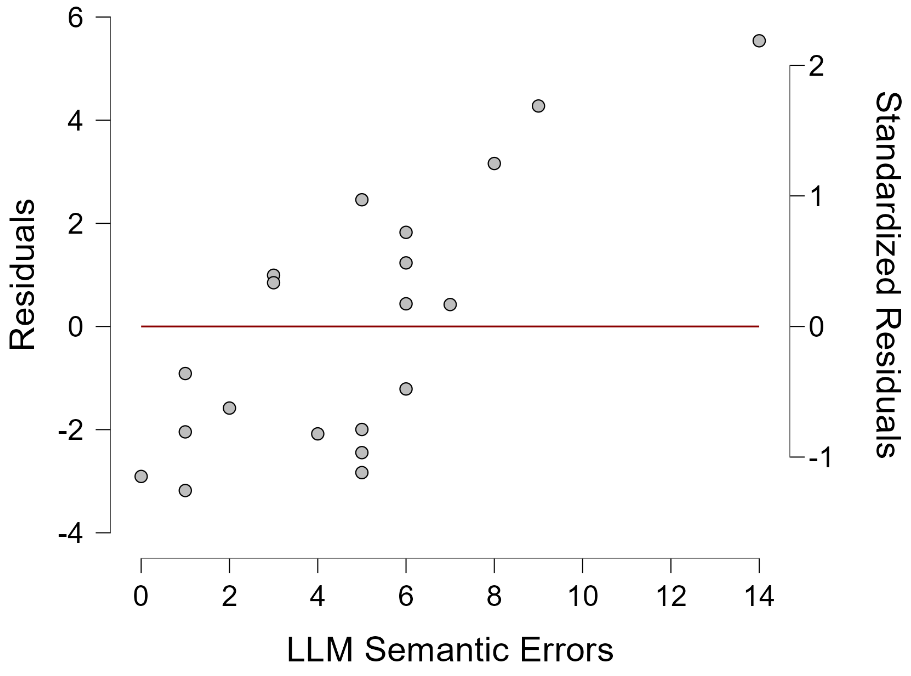


Figure D.4: Residuals Vs. LLM Semantic Errors

Human Pragmatic Errors

Model	R	R^2	Adjusted R^2	RMSE
H_0	0.000	0.000	0.000	1.119
H_1	0.514	0.264	0.067	1.081

Table D.9: Model Summary - Human Pragmatic Errors

Model		Unstandardized	Standard Error	Standardized	t	p
H_0	(Intercept)	1.100	0.250		4.395	< .001
H_1	(Intercept)	0.500	1.253		0.399	0.696
	Classes	−0.330	0.305	−0.472	−1.083	0.296
	Attributes	−0.019	0.031	−0.138	−0.606	0.554
	Associations	0.399	0.217	0.796	1.838	0.086
	AVG ED	0.027	0.260	0.025	0.102	0.920

Table D.10: Coefficients

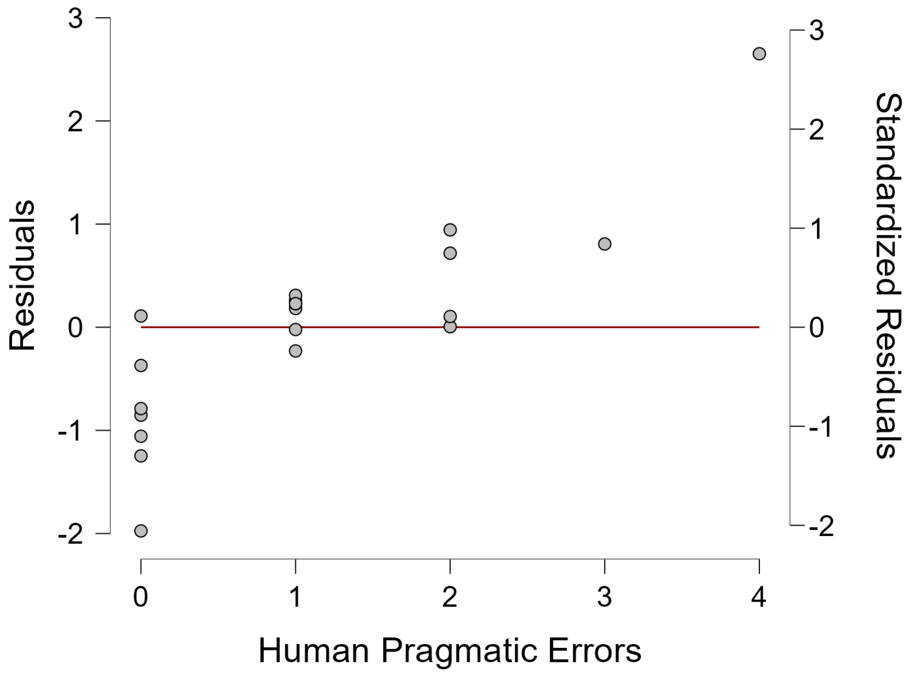


Figure D.5: Residuals Vs. Human Pragmatic Errors

LLM Pragmatic Errors

Model	R	R^2	Adjusted R^2	RMSE
H_0	0.000	0.000	0.000	0.940
H_1	0.773	0.597	0.490	0.672

Table D.11: Model Summary - LLM Pragmatic Errors

Model		Unstandardized	Standard Error	Standardized	t	p
H_0	(Intercept)	1.600	0.210		7.610	$< .001$
H_1	(Intercept)	-0.792	0.779		-1.016	0.326
	Classes	0.485	0.189	0.827	2.565	0.022
	Attributes	-0.045	0.019	-0.390	-2.320	0.035
	Associations	-0.091	0.135	-0.215	-0.671	0.513
	AVG ED	0.079	0.162	0.089	0.487	0.633

Table D.12: Coefficients

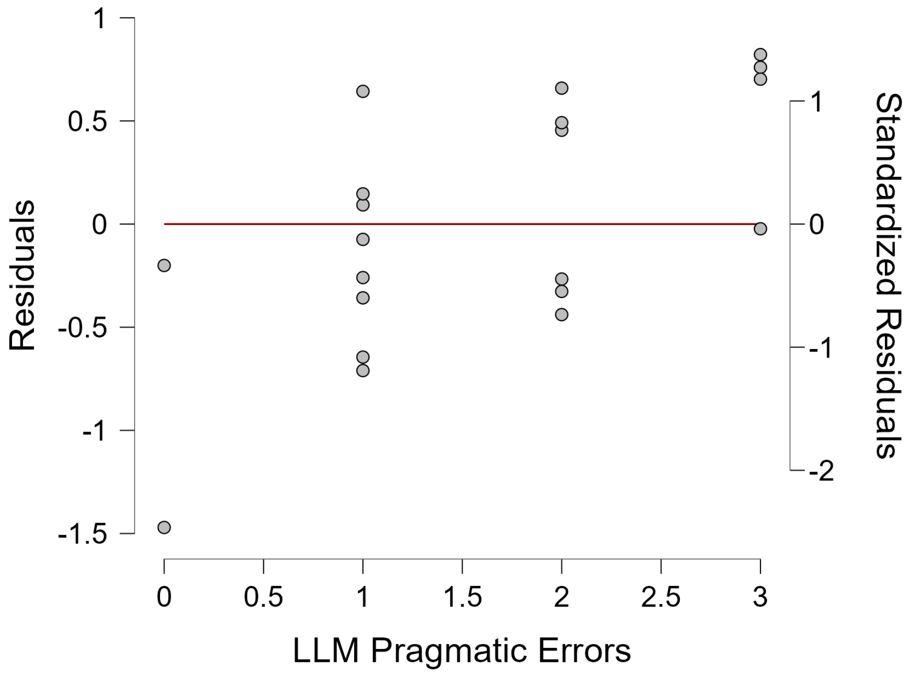


Figure D.6: Residuals Vs. LLM Pragmatic Errors

D.2 Second Part

Human Distances

Model	R	R^2	Adjusted R^2	RMSE
H_0	0.000	0.000	0.000	2.356
H_1	0.770	0.592	0.483	1.693

Table D.13: Model Summary - Distance Human

Model		Unstandardized	Standard Error	Standardized	t	p
H_0	(Intercept)	5.008	0.527		9.508	$< .001$
H_1	(Intercept)	0.915	1.963		0.466	0.648
	Classes	−0.120	0.477	−0.081	−0.251	0.805
	Attributes	−0.038	0.049	−0.130	−0.771	0.453
	Associations	0.105	0.340	0.100	0.310	0.761
	AVG ED	1.675	0.408	0.759	4.108	$< .001$

Table D.14: Coefficients

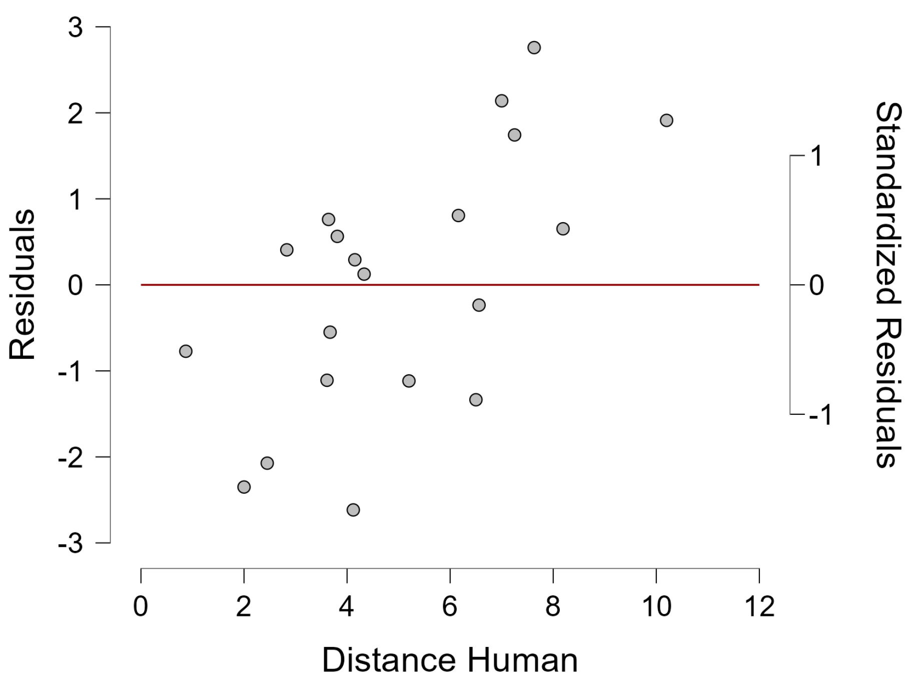


Figure D.7: Residuals Vs. Distance Human

LLM Distances

Model	R	R^2	Adjusted R^2	RMSE
H_0	0.000	0.000	0.000	2.855
H_1	0.804	0.647	0.553	1.909

Table D.15: Model Summary - Distance LLM

Model		Unstandardized	Standard Error	Standardized	t	p
H_0	(Intercept)	7.245	0.638		11.350	< .001
H_1	(Intercept)	−0.005	2.213		−0.002	0.998
	Classes	−0.548	0.538	−0.308	−1.019	0.324
	Attributes	0.139	0.055	0.397	2.519	0.024
	Associations	0.567	0.384	0.443	1.477	0.160
	AVG ED	1.635	0.460	0.611	3.557	0.003

Table D.16: Coefficients

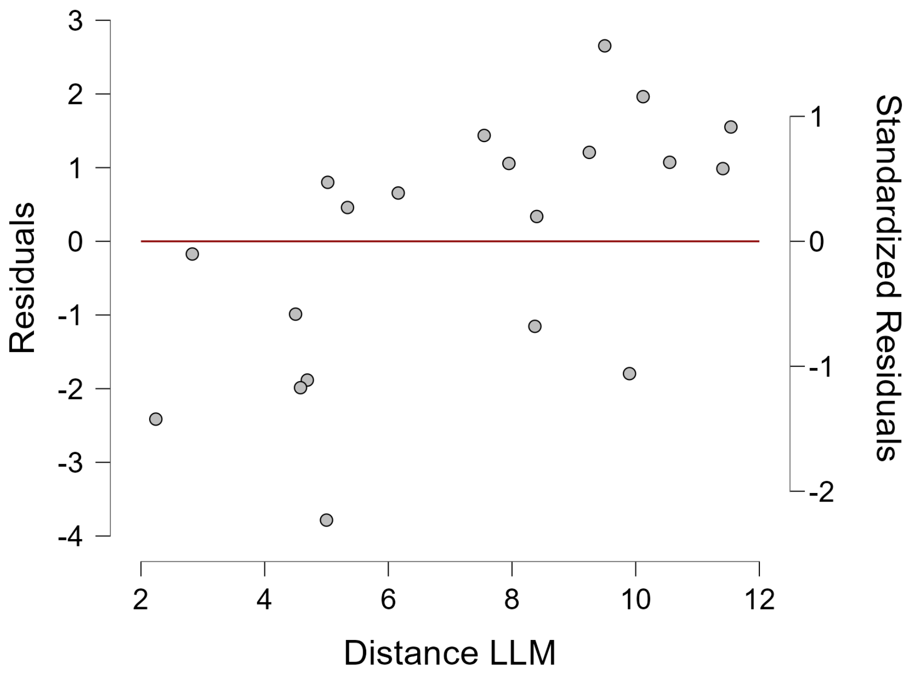


Figure D.8: Residuals Vs. Distance LLM

Bibliography

[1] Narasimha Bolloju and Felix SK Leung. Assisting novice analysts in developing quality conceptual models with uml. *Communications of the ACM*, 49(7):108–112, 2006.

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

[3] Dusanka Dakic, Darko Stefanovic, Teodora Lolic, Srdjan Sladojevic, and Andras Anderla. Production planning business process modelling using uml class diagram. In *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–6, 2018.

[4] Jan Clusmann, Fiona R Kolbinger, Hannah Sophie Muti, Zunamys I Carrero, Jan-Niklas Eckardt, Narmin Ghaffari Laleh, Chiara Maria Lavinia Löffler, Sophie-Caroline Schwarzkopf, Michaela Unger, Gregory P Veldhuizen, et al. The future landscape of large language models in medicine. *Communications Medicine*, 3(1):141, 2023.

[5] Martin Schroder. Autoscrum: Automating project planning using large language models. *arXiv preprint arXiv:2306.03197*, 2023.

[6] Samia Nasiri, Yassine Rhazali, Mohammed Lahmer, and Amina Adadi. From user stories to uml diagrams driven by ontological and production model. *International Journal of Advanced Computer Science and Applications*, 12(6), 2021.

[7] Outair Anas, Tanana Mariam, and Lyhyaoui Abdelouahid. New method for summative evaluation of uml class diagrams based on graph similarities. *International Journal of Electrical and Computer Engineering*, 11(2):1578–1590, 2021.

[8] Alice Folvaiter. Mydoctor, 2022.

[9] Victoria Paulsson, Vincent C. Emeakaroha, John P. Morrison, and Theo

Lynn. Cloud service brokerage: A systematic literature review using a software development lifecycle. In *22nd Americas Conference on Information Systems, AMCIS 2016, San Diego, CA, USA, August 11-14, 2016*. Association for Information Systems, 2016.

[10] Pieter Willem Jordaan and Johann Erich Wolfgang Holm. Reflection on mongodb database logical and physical modeling. In *2019 IEEE AFRICON*, pages 1–8, 2019.

[11] Valentin Burkin. Mitigating risks in software development through effective requirements engineering, 2023.

[12] Ramya Sri Simhadri and Mohammad Shameem. Challenges in requirements gathering for agile software development. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, EASE '23, page 406–413, New York, NY, USA, 2023. Association for Computing Machinery.

[13] Maxim Sergievskiy and Ksenia Kirpichnikova. Optimizing uml class diagrams. In *ITM Web of Conferences*, volume 18, page 03003. EDP Sciences, 2018.

[14] Matias Zapata-Barra, Alfonso Rodriguez, Angelica Caro, and Eduardo B Fernandez. Towards obtaining uml class diagrams from secure business processes using security patterns. *J. Univers. Comput. Sci.*, 24(10):1472–1492, 2018.

[15] Dian Sa'adillah Maylawati, Muhammad Ali Ramdhani, and Abdusy Syakur Amin. Tracing the linkage of several unified modelling language diagrams in software modelling based on best practices. *International Journal of Engineering & Technology (UEA)*, 7(2.19):776–780, 2018.

[16] Victor MR Penichet, Jose A Gallud, Ricardo Tesoriero, and Maria Lozano. Design and evaluation of a service oriented architecture-based application to support the collaborative edition of uml class diagrams. In *Computational Science–ICCS 2008: 8th International Conference, Kraków, Poland, June 23-25, 2008, Proceedings, Part III 8*, pages 389–398. Springer, 2008.

[17] Immanuel Trummer. From bert to gpt-3 codex: harnessing the potential of very large language models for data management. *arXiv preprint arXiv:2306.09339*, 2023.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[19] Aditya Malte and Pratik Ratadiya. Evolution of transfer learning in natural language processing, 2019.

[20] Thomas Hügle. The wide range of opportunities for large language models such as chatgpt in rheumatology. *RMD open*, 9(2):e003105, 2023.

[21] Thilo Hagendorff and David Danks. Ethical and methodological challenges in building morally informed ai systems. *AI and Ethics*, 3(2):553–566, 2023.

[22] Mathawan Jaiwai and Usa Sammapun. Extracting uml class diagrams from software requirements in thai using nlp. In *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–5, 2017.

[23] Shyni Sharaf and VS Anoop. An analysis on large language models in healthcare: A case study of biobert. *arXiv preprint arXiv:2310.07282*, 2023.

[24] Dabo Sun and Kenny Wong. On evaluating the layout of uml class diagrams for program comprehension. In *13th International Workshop on Program Comprehension (IWPC'05)*, pages 317–326. IEEE, 2005.

[25] Odd Ivar Lindland, Guttorm Sindre, and Arne Solvberg. Understanding quality in conceptual modeling. *IEEE software*, 11(2):42–49, 1994.

[26] Chris Roth. *Using Microsoft Visio 2010*. Pearson Education, 2011.

[27] Thomas E Scruggs and Margo A Mastropieri. Summarizing single-subject research: Issues and applications. *Behavior modification*, 22(3):221–242, 1998.

[28] Bethany Gosala, Sripriya Roy Chowdhuri, Jyoti Singh, Manjari Gupta, and Alok Mishra. Automatic classification of uml class diagrams using deep learning technique: convolutional neural network. *Applied Sciences*, 11(9):4267, 2021.

[29] Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. Gpt-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems. *arXiv preprint arXiv:2310.12397*, 2023.