POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Master Degree course in Engineering and Management

Master Degree Thesis

LLM-based Generation and Evaluation of UML Class Diagrams

Supervisors
Prof. Riccardo Coppola
Prof. Giacomo Garaccione

Candidate
Roberta Soldati

Academic Year 2024-2025

Abstract

In the digital era, information systems play an essential role in supporting the operational processes of modern organizations. Among the critical phases of software development, conceptual modeling remains one of the most decisive, especially during the early stages, when system requirements are still being explored and formalized. The heart of conceptual modeling lies in the use of diagrammatic notation, with the Unified Modeling Language (UML) class diagrams emerging as one of the most widely adopted standards for representing the static structure of object-oriented systems. These diagrams enable designers to abstractly represent classes, their attributes, and interrelations such as associations, generalizations, and aggregations, providing a visual formalism that facilitates analysis, communication, and software specification. However, despite their expressive power and standardization, UML class diagrams pose considerable cognitive challenges for learners and inconsistencies for evaluators. In educational contexts, instructors frequently compare the difficulty of manually assessing diagrammatic solutions submitted by students, where equally valid structural variations can lead to subjectivity in grading. From an industrial standpoint, early-stage software modeling is often constrained by time and the cost of human effort especially when modeling from informal requirement specifications. This thesis tries to answer the question of to what extent artificial intelligence can assist or even automate the generation and evaluation of UML class diagrams from textual requirements. In recent years, the rise of Large Language Models (LLMs), based on transformer architectures, allowed us to generate syntactically correct and semantically plausible outputs in a variety of formats, from natural language to domain-specific representations like JSON one. However, their capabilities are deeply influenced by the formulation of the input, commonly known as the prompt, which defines the task context, constraints, and expected structure. This has led to the emergence of prompt engineering as a critical practice for aligning LLM outputs with specific domain goals, especially when precision and interpretability are critic. Hence, this thesis investigates the feasibility and effectiveness of employing LLMs for the automatic generation and evaluation of UML class diagrams. Specifically, it explores whether and how prompt-engineered LLMs can generate class diagrams that comply with syntactic, semantic, and pragmatic standards of quality, and whether these outputs can be assessed through replicable and objective evaluation mechanisms. The research is structured around a twofold objective: designing and optimizing a prompting strategy that enables LLMs to generate UML diagrams in JSON format, fully compatible with the Apollon and UML-Modeler tools and developing an evaluation pipeline that can assess the correctness and similarity of such diagrams, with respect to the reference ones, through rubric-based scoring. The findings confirm the potential of LLMs in educational modeling tasks, although limitations remain. Diagram generation is sensitive to prompt formulation, and rubric-based evaluations, while practical, introduce subjectivity. Future work could involve extending to other modeling languages, incorporating visual assessment, and averaging multiple human evaluations to reduce bias and enhance reliability.

Contents

Chapter 1

Introduction

In the information age we live, information systems provide core mechanisms for supporting operational business processes of organizations through the collection, managing and distribution of information (data) within different system processes. The design or *modeling phase* of a software process is one of the most crucial to ensure the quality of the final product since it focuses on constructing abstract representations of a system to define its structure, behavior, and interactions. Indeed, software process modeling languages are an important support for describing and managing software processes in software-intensive organizations as they rely on different levels of abstraction, allowing stakeholders to understand and possibly redefine system behavior before its implementation. Modeling languages can be classified into different categories based on their purpose and level of abstraction. The design of a software process at the highest level of abstraction initiates through the conceptual modeling languages, such as Unified Modeling Language (UML) or Business Process Model and Notation (BPMN), which focus on the main entities and their relationships, not entering into details. They are the base for the following logical modeling languages, including Data Flow Diagrams (DFD), which provide a more structured view of system behavior and interactions, highlighting data flows and functionalities. Lastly, physical modeling languages, such as hardware description languages (HDLs) and database schema models, define system implementation details, ensuring that the designs translate into practical applications.

In particular, UML has been proposed by the OMG standard in the early nineties as the general-purpose software modeling language that adopts the object-oriented software engineering paradigm. An object-oriented system unifies the data structure and behavioral features into a single object structure. The UML captures information about the static structure and dynamic behavior of a system. A system is modeled as a collection of discrete objects that interact to perform work that ultimately benefits an outside user. The static structure defines the kinds of objects important to a system and its implementation, as well as the relationships among the objects. Dynamic behavior defines the history of objects over time and the communications among objects to accomplish goals. Modeling a system from several separate but related viewpoints permits it to be understood for different purposes.

Chapter 2

Background

2.1 UML class diagrams

Today, UML is considered the preferred software modeling language by professionals who work in various industries. Indeed, several recent survey studies ([6, 11, 16]) which compared UML with other formal and informal software modeling languages pointed out that most of the practitioners from diverse industries use UML for their software modeling activities, since it offers a visual notation set that consists of various diagrams for the specifications of software systems from different viewpoints (e.g., logical, physical, deployment, and behavior).

Regarding the static view, the main elements are classes and their relationships, which the main are association, generalization, and various kinds of dependency.

- Classes: classes are a description of a concept in the application domain or in the application solution. In fact, they are the center around which the class view is organized; other elements are owned by or attached to classes. The static view is displayed in class diagrams, so-called because their main focus is the description of classes. They are drawn as rectangles, but lists of attributes and operations are shown in separate compartments.

- Attributes: attributes are the properties or variables of a class, They define the characteristics or state of an object. They capture fundamental data about an object, in this way, the system can store, retrieve, and manipulate this data when it is needed.

- 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, which is a bidirectional relationship between two classes, aggregation, which represents a 'whole-part' relationship, where one class is a component of another, and generalization: represents a *is-a* relationship between a base class (parent) and a derived class (child).

- 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. The cardinality constraints allowed are zero or one, exactly one, zero or more, one or more, exactly n and from m to n inclusive.

An illustrative example of a UML class diagram is proven in Figure 2.1. It models a telecommunications billing system and highlights several superior functions of UML diagrams. In this diagram, the PhoneBill class is composed of one or more PhoneCall instances, represented through an aggregation relationship. The multiplicity 1..1 at the PhoneBill facet and 1..* at the PhoneCall side expresses that each bill references at least one phone name, and every telephone call belongs to precisely one bill. The PhoneCall class is further specialized via the subclass MobileCall, following an inheritance relationship, which allows the model to distinguish between daily calls and those originating from cell gadgets. A PhoneCall is related to an Origin through a specification, which records the region attribute. This is likewise prolonged with the aid of the MobileOrigin, allowing similarly specification in the context of mobile communications.

Each PhoneCall is made from exactly one Phone, and a cellphone may additionally have participated in 0 or more calls. The Phone istance itself is an abstract class, with two disjoint and complete specializations: CellPhone and FixedPhone. This generalization is classified with complete, disjoint, that means that each example of Phone need to be both a CellPhone or a FixedPhone, and no example may be each.

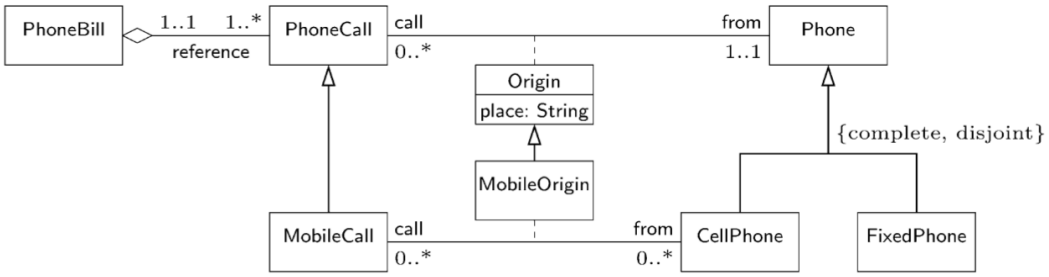

Figure 2.1. UML class diagram example [2]

The construction of a class diagram evolves through various steps:

- Gathering of requirements: It is necessary to understand the requirements of the stakeholders before starting with the design of the system. Consequently, they can be associated with classes and their attributes.
- Identifying Relationships: determine how the classes will interact with each other by understanding the type of relationships. This phase also include the definition of the multiplicity of the relationships and their hierarchy.
- Drawing the diagram: start with classes, then add attributes, and finally represent the relationships.
- Review and Refine: once the initial diagram is ready, review it for accuracy and completeness. Make necessary refinements.

2.1.1 Difficulties and Ambiguities in UML Modeling

Despite its standardization and widespread adoption in both academic and industrial contexts, building correct and meaningful UML diagrams is still a challenging task, particularly when translating natural language descriptions into conceptual models. One of the main challenges lies in interpreting unstructured natural language and translating it into representations that adheres with the UML's formal constraints. Since textual descriptions are often incomplete or ambiguous, this process requires extrapolating structure and semantics that are not explicitly stated, while maintaining both syntactic correctness and conceptual clarity.

- One of the primary issues encountered is the *semantic vagueness* inherent in textual requirements. Descriptions often lack explicit information about cardinalities, the directionality of relationships, or the distinction between aggregation and composition. This leads to ambiguous interpretations even among experienced modelers. As highlighted by [5], such ambiguities contribute significantly to inconsistent and erroneous student diagrams, which are difficult to evaluate systematically.

- Another frequent source of error is the difficulty in distinguishing between *attributes* and *associations*. Elements such as "address", "contract type", or "responsible person" can be validly represented either as class attributes or as linked entities, depending on the intended semantic granularity and normalization level. This duality often results in divergent solutions across students and represents a critical challenge for LLMs, which struggle to apply consistent thresholds for class abstraction, especially in the absence of domain-specific constraints [1].

- Ambiguities also arise in handling *specializations*. Without explicit cues, modelers may overlook generalizable concepts or, conversely, introduce unnecessary hierarchies. The decision to model a concept as an abstract class, a subclass, or an enumeration is often subjective and significantly affects the logical structure of the diagram. These modeling choices, while formally correct, may result in semantically divergent interpretations — a phenomenon noted in comparative studies of student-submitted diagrams [14].

- Lastly, naming conventions themselves are a potential source of ambiguity. Generic or inconsistent naming (e.g., "Data", "Entity", "Info") reduces readability and hinders the interpretation of the model, especially when diagrams are generated or compared automatically. Misaligned terminology between class names and attribute names further contributes to semantic confusion, and often correlates with lower comprehensibility scores in manual assessments [11].

These challenges not only hinder the educational process in conceptual modeling but also serve as critical indicators of the current limitations and future opportunities for generative AI systems applied to UML diagram construction.

2.1.2 Evaluation and quality measures of UML class diagrams

One of the most time-consuming task for teachers and students is creating, correcting and evaluating exercises. Traditional assessment methods are limited by several factors, such as subjectivity, especially considering that conceptual modeling permits many equally valid variants of the same domain. In response to these challenges, it is essential to provide a quantitative and objective approach for evaluating the quality of exercises. According to Narasimha Bolloju and Felix S.K. Leung conceptual model quality framework, [3] the quality of UML artifacts can be measured from syntactic, semantic and pragmatic perspectives.

Table 2.1. Common UML Modeling Errors Categorized by Quality Dimension [3]

Model Type	**Syntactic Quality**	**Semantic Quality**	**Pragmatic Quality**
Use Case Models	Invalid notation; improper naming (e.g., missing action verbs)	Incorrect use of include, extend or generalization relationships	UI details in descriptions; incomplete scenarios; poor layout
Domain Models	Inappropriate naming of classes or associations	Missing or incorrect cardinalities; use of aggregation instead of association	Redundant attributes or associations; weak subclass distinction
Dynamic Models	Improper placement of objects in sequence diagrams	Incomplete specification of message parameters	Inappropriate delegation of responsibilities

Hence, the syntactic correctness of a model implies that all statements in it depend on the syntax of the language, capturing how a given model adheres to the language rules or to the syntax. Therefore, fewer errors and deviations from the rules indicate better syntactic quality. Semantic quality is described in terms of validity and completeness goals. The validity goal specifies that all statements in the model are correct and relevant to the problem domain. The completeness goal specifies that the model contain all statements about the problem domain that are correct and relevant. However, it may be that these two goals cannot be achieved, unlike syntactic correctness, which can be achieved. Pragmatic quality captures how the model has selected *from among the many ways to express a single meaning* and essentially deals with making the model easy to understand. The comprehension goal specifies that all interpreters completely understand the statements in the model that are relevant to them.

Considering these perspectives of quality, Narasimha Bolloju and Felix S.K. Leung analyzed the quality of many UML artifacts in 15 team-project reports submitted by undergraduate students taking a course in object-oriented analysis and design in the Department of Information Systems at the City University of Hong Kong and noticed that errors related to relationships, such as association and specification were frequent, especially the cardinality details. Most of the pragmatic errors observed were related to derived or redundant attributes, the use of keys and the insufficient distinction among subclasses.

Another main contribution in this route is the method proposed by means of Nikiforova et al. [10], which proposes a new approach for the summative assessment of UML class diagrams based totally on graph similarities. This method addresses the challenge of objectively evaluating a diagram made by a student to a reference solution by remodeling both into categorized graphs, in which nodes constitute classes and edges denote relationships, including associations and generalizations.

To compare the two graphs, the method computes multiple layers of similarity:

- Node similarity is evaluated by comparing class names and their associated attributes and methods. This involves both lexical comparison techniques (such as string matching or edit distance) and semantic similarity measures (e.g., using WordNet to recognize that "Client" and "Customer" are semantically related).
- Edge similarity is determined by analyzing the types of relationships between classes, including their directionality, roles, and cardinalities. The system checks whether two edges connect semantically or structurally equivalent nodes and whether they serve the same conceptual role.

All similarity scores are stored in a similarity matrix, where each cell $S(i, j)$ represents the similarity between a node i in the student graph and a node j in the reference graph. An optimal node-to-node mapping is then computed using a graph alignment algorithm, which maximizes the total similarity score while avoiding redundant or conflicting matches. This step is conceptually similar to solving a weighted bipartite matching problem (e.g., via the Hungarian algorithm).

Finally, the method combines the results into an overall similarity score, computed as a weighted sum of node similarity, edge similarity, and attribute similarity:

$$Sim(G_s, G_r) = \alpha \cdot Sim_{\text{nodes}} + \beta \cdot Sim_{\text{edges}} + \gamma \cdot Sim_{\text{attributes}}$$

where G_s and G_r are the student and reference graphs, respectively, and α, β, γ are configurable weights that reflect the relative importance of each component.

This quantitative evaluation process enables the system to assess diagrams accurately, even when they differ syntactically but are semantically equivalent. The method was validated through experiments on real student submissions, demonstrating a strong correlation between automatic similarity scores and the grades assigned by expert human evaluators. Furthermore, it lays the groundwork for integrating computerized evaluation gear into instructional systems, thus helping each instructor and freshmen in the teaching and gaining knowledge of conceptual modeling.

However, providing an assistant tool based on Gen-AI that automates such tasks would optimize teachers performances both in a time cost perspective and in error tendency.

2.2 Large Language Models

"Large Language Models (LLMs) are neural architectures that leverage transformer-based training over extensive textual corpora to model linguistic patterns and perform generative tasks" [4]. These models are trained on very large datasets and have been shown to

perform quite well at language processing tasks. They generate new text by predicting word sequences, based on the input they receive.

2.2.1 Transformer Architecture of Large Language Models

Most of the Large Language Model (LLM) are based on a 'Transformer Architecture' which, unlike earlier architecture that processed tokens sequentially (word by word), employs a mechanism called self-attention that enables it to consider all positions in a sequence simultaneously. The Transformer architecture was originally proposed by Vaswani et al. in the influential paper introducing the Transformer model [15] and it is expected that each word or token in the input is converted to a continuous vector representation initially and added to positional encodings for retaining token order information since the architecture itself does not incorporate any sequence bias.

Self-attention allows each token to selectively attend to other tokens in the sequence, learning patterns of relationship that depend on the context. This is done by projecting three vectors for each token: a query, a key, and a value. The attention weights are computed by taking the dot product of the query and key vectors, normalizing them, and using them to weight the value vectors. The weighted values are combined to form a context-aware representation of each token. To enhance the capacity of the model to represent diverse aspects of relationships, the Transformer utilizes multi-head attention, which applies the self-attention mechanism in parallel across multiple heads and concatenates the results. Following the attention mechanism, each token is passed through a feed-forward neural network, applied identically but independently to each position.

Both the feed-forward and attention blocks are wrapped in layer normalization and residual connections, which help stabilize training and improve convergence. A standard Transformer consists of stacks of these attention and feed-forward layers. The original Transformer consists of an encoder and a decoder. The encoder reads an input sequence and produces a sequence of representations, and the decoder generates a sequence by attending to the encoder output and the tokens generated so far. In the case of text generation models like GPT, however, only the decoder component is used in a unidirectional, autoregressive fashion, where each token is generated based on only preceding tokens.

Generally, LLMs are part of Neural Language Models (NLMs) category, which includes any language model based on neural networks, a mathematical model made up of layers of interconnected nodes, and of Pretrained Language Models (PLMs), which are NLMs that have been trained in advance on general text and can be fine-tuned. Fine-tuning refers to the process of taking a pretrained language model and adapting it to a specific task or domain by continuing the training process on a smaller, task-specific dataset. While the initial pretraining phase allows the model to learn general linguistic patterns and structures, fine-tuning enables it to specialize in particular tasks such as text classification, question answering, named entity recognition, or domain-specific language understanding. However, while LLMs exhibit impressive capabilities in generating coherent and contextually relevant text. In addition to general-purpose language tasks, LLMs are increasingly applied to structured output generation, such as generating code snippets, business process models, tabular data or spreadsheets formulas . These

tasks require the model to adhere not only to linguistic fluency but also to structural and syntactic constraints defined by domain-specific languages or formal notations. Ensuring output validity in such contexts remains a significant challenge, as minor formatting or structural deviations can render the output unusable by downstream tools. However, they may also produce inaccuracies, ambiguities, or biased outputs, posing risks to the reliability and quality of software requirements. These models are also prone to phenomena such as *hallucination*, where outputs appear fluent and plausible yet are factually incorrect, and *overconfidence*, where the model expresses incorrect information with unjustified certainty. These issues are particularly problematic when outputs are interpreted without human oversight or applied in critical decision-making scenarios. Moreover, ethical considerations such as data privacy, model bias, and transparency justify careful examination when employing LLMs in sensitive domains like software development.

Alternatively, LLMs can perform tasks through *in-context learning* (ICL), where they are prompted with a few examples or instructions at inference time, without requiring any additional training. This approach is especially relevant in academic and industrial contexts where fine-tuning is not feasible due to resource constraints or lack of labeled data. Prompt engineering strategies—discussed in the next section—are designed to exploit the full potential of ICL.

Thus, the main critical aspects of using LLMs are:

- Semantic interpretability: using LLMs effectively means formulating appropriate prompts, the input text or queries provided to these models to generate desired outputs. To have real world semantics, referential mapping is required.
- Domain specialization: poor performance in specialized domains without domain adaptation. This would involve simplifying complex phenomena into more understandable concepts.
- Poor quality data: if data contains biases, the model can lead to incorrect and inconsistent predictions and can internalize and replicate those patterns in its outputs.
- Dependence on training data: if a topic was not covered in the training data, the model's responses may be inaccurate, shallow or entirely wrong.

2.3 Prompt Engineering

Prompt engineering plays a particularly critical role in structured output generation tasks, such as the creation of UML class diagrams, where the correctness of the output depends not only on the semantic understanding of the input but also on strict adherence to predefined syntactic formats (e.g., valid JSON or PlantUML). Unlike open-ended text generation, these tasks demand outputs that are machine-interpretable and structurally validated, making prompt design a key factor in determining whether the model succeeds or fails. LLMs are uncertain processes, so the quality of formulating prompts significantly contributes to the content's semantic fidelity and syntactic clarity. In the educational setting, for instance, prompt design is at the core for automated software

modeling assignment generation and feedback on UML diagrams. As a simple example, the content of a prompt can have a huge impact on the output quality if you mention constraints, domain language even in response format. Furthermore, via methods such as few-shot prompting and chain-of-thought prompting on the LLMs further enhance their capability of managing complex reasoning or structured generation. Therefore, prompt engineering sits between what a model is capable of and the user's goal by translating that to actionable work done by a generative tool whilst meeting domain-specific expectations.

2.3.1 Prompting Techniques

Prompt engineering is more effective when the task is clear and the model has been pretrained on it in a given domain. That's why there are known techniques to improve quality. Many of them are shown in the article by Zhu et al. [17] taken as reference:

- Zero-shot prompting: employs an instruction with the minimal use of examples. More effective in most general-purpose tasks or clarity and conciseness is an important drive. For example, if the model is already transferred to UML-related content, 'Generate a UML class diagram from the following requirements' is doable.

- Few-shot prompting: few examples that represent input-output pairs in the prompt are usually good enough. This technique it is useful when the model has to deduce the structure or style of the response it is targeting, by for example translating a requirement into software pattern class diagram, and when helps with prompts that have some constraints such as associations, repetitions or multiplicities.

- Chain-of-Thought prompting (CoT): triggers the model to take intermediate steps while generating answers. For instance, two-way reasoning tasks (e.g. if you have to decide whether a class diagram complies with both syntactic as well semantic quality standards for its semantics). Typically, it firstly identifies the entities, then relationships and assigns attributes with multiplicities properly.

- Common prompting techniques: methods like providing structured prompts in templates such as bullet points or specific sections or delimiters, can train the model towards more deterministic outputs. This is more useful for things you want to be able to iterate on and serve users again, e.g., quizzes or UML feedback forms.

- Role-based prompting: the model can recive a persona (e.g., 'You are a software modeling tutor') to steer it towards answering in a way that is pedagogical and/or didactic.

Sometimes, these prompt engineering techniques can also be used simultaneously, for instance, two or more at the same time, to optimize and refine the initial setup. This approach is aligned with recent advances showing that large language models can perform step-by-step reasoning even in zero-shot settings [7]. It also aligns with promptware engineering principles, where prompts are not considered as isolated, ad-hoc instructions, but rather as reusable, modular, and testable components of a broader software system. This implies that the process of engineering prompts is comparable to the ones typical

of traditional software development. They're indeed versioned, documented, evaluated, and often embedded in pipelines or tools to ensure reliability and reproducibility of the outputs.

2.4 Current solutions for the generation of UML class diagrams

All of the current available solutions for creating and evaluating students exercises are based on DSL. A DSL (Domain Specific Language) is a programming language built for a specific scope in a specific domain. The main differences between generic languages, such as Python or Java, is that a DSL is specialized in a domain, it is easy to use and very expressive. A LLM (Large Language Model) is an AI system trained on vast text data to understand and generate human language. DSLs are rule-based and deterministic, while LLMs are probabilistic and learn patterns from data. An LLM can generate or assist with writing DSL code, but it is not a DSL itself.

Three of the many existing approaches are:

- Java-based tool for the automated evaluation and grading of student UML diagrams through structural and syntactic matching, which is promising to solve the feedback and grades themes, but it is still not using any machine learning or any generative models and it works only with Modelio Diagrams.
- *Wodel-Edu*, a DSL extension of Wodel, which is able to automatically generate modeling exercises (such UML Diagrams). Even if it is promising well, there are some limits in using Wodel-Edu:
  - Evaluation of students answers is not possible. It needs to have an additional tool to generate a tailored feedback.
  - Simplistic Mutation: since they are based on rules manually inserted, they may not be pedagogically efficient (not calibrated on student knowledge) and they may result in too easy or ambiguous.
  - Teachers need to have competencies in writing DSL rules.
- Rule-based approaches for automated assessment of UML class diagrams. In particular, we cite a metamodel born as an extension of *TouchCORE* [13], which is a tool designed for model-driven software development, and *AutoER*, a web-serving application to help in the automated generation and automatic marking of UML database design diagrams. Neither of the two is AI-based, as they compare student-submitted UML class diagrams with an instructor's reference solution. In particular, they use a set of structured comparison algorithms and metamodel to evaluate the syntactic, semantic, and structural similarity between models:
  - Syntactic similarity refers to the textual closeness between element names, such as class or attribute identifiers. This type of comparison handles minor spelling errors, pluralization, typos, or variations in naming conventions. It is typically

measured using the Levenshtein distance, which calculates the number of edits needed to convert one string into another.

- Semantic similarity, on the other hand, focuses on the meaning of the names, allowing the tool to recognize conceptually related terms that are not necessarily textually similar. This is achieved through lexical databases like WordNet, which assess how closely two terms are related in a linguistic or conceptual sense. This is especially useful when students use synonyms or domain-relevant alternatives, such as "Teacher" instead of "Instructor."
- Structural similarity examines the relationships, positions, and contents of model elements, such as whether a class contains the correct attributes or methods, whether associations are correctly defined between relevant classes, or if inheritance hierarchies are maintained. This type of matching ensures that the overall model structure aligns with the intended design, even if individual names differ or elements are misplaced within the hierarchy. Together, these three dimensions allow the tool to provide a robust and flexible assessment of student models, accommodating variations while maintaining the accuracy of the rating.

Again, while the proposed systems offer a robust and flexible method for evaluating UML class diagrams and immediate and detailed feedback that enhance formative learning, they also come with several limitations that affect their applicability and accuracy in certain contexts.

- The tools are not AI-driven, meaning they rely entirely or partially on predefined algorithms. They do not learn from past data nor they adapt their grading logic based on student behavior and feedback.
- Word-net is a general-purpose lexical database, and it may not be sufficient when dealing with domain-specific vocabulary.
- The tools do not assess diagram layout or visual quality. Issues such as overlapping elements, disorganized layouts, or missing labels are ignored in the grading process, even though they are often important in real-world modeling tasks.
- Limited flexibility, since they require instructors to prepare mark-up, reference solutions and structured questions manually.

- Template-based generation: Sadigh et al. [12] proposed a framework based on the three main aspects of the exercise life cycle, that is, problem generation, solution analysis and auto grading. In this context, existing exercises are abstracted into parameterized templates and new exercises are generated by instantiating these templates or applying bounded mutations to existing models. While solutions, student feedback and evaluation are made through model checking, simulation and SMT solving. Model checking is a technique used to automatically verify if a system model satisfies certain formal properties by exploring all possible states and behaviors. Simulation refers to running a model with specific inputs to observe

how it behaves under particular conditions, often to provide examples or detect design issues. Satisfiability Modulo Theories solving (SMT solving) is an extension of Boolean feasibility that allows for checking and solving problems involving more complex data types and constraints by encoding them into logical formulas. This system enables scalable and systematic generation of exercises, which is critic in online course environments, such as MOOCs, where thousands of students may need unique but equivalent practice problems. The use of formal methods guarantees a high degree of precision and correctness in both problem generation and evaluation, minimizing the possibility of human error. Additionally, the use of mutation-based problem generation allows instructors to control the difficulty of the exercises and to create a wide range of problem variations from a common template, which enhances both student learning and assessment quality. However, there exist similar limits in using template-based systems for generating exercises of the ones presented before:

- Lack of adaptability: the system is not AI-driven and does not adapt or personalize feedback based on student performance.
- High instructor workload: it requires full formalization of models, properties, and traces, which can increase the time and effort needed from instructors.
- Inconsistent pedagogical value: mutated models may not always be pedagogically meaningful, sometimes resulting in trivial or overly complex problems.
- Domain specificity: the system is limited to domains that can be fully expressed and verified using formal methods, reducing its applicability to more open-ended or creative modeling tasks.

Therefore, despite the promising results of current DSL-based, rule-based, and template-based approaches, several limitations persist. The critic is the lack of adaptability, as none of the systems is AI-driven or capable of customizing feedback. They also require significant manual preparation by instructors and often do not handle domain-specific terminology. In addition, they do not assess the visual quality of diagrams and may generate exercises of inconsistent pedagogical value.

However, there exist other techniques for comparison and evaluation of UML Class Diagrams but based on simpler approaches. UML Class Diagrams can be compared through semantic features of their elements, such as classes, attributes, methods, and relations. These techniques pair elements based on semantic similarity requiring human involvement to ensure that equivalent elements are correctly identified, even if they have different names and calculate distances to quantify differences. The calculated distances are aggregated into a difference vector, which quantitatively represents the discrepancies between the diagrams. The length of this vector provides a final numerical value indicating the degree of difference between the compared diagrams.

Another promising solution for UML generation and evaluation is through Gamification, which is the use of game-design elements and principles in non-game environments to increase user engagement/motivation. UMLegend is an example of gamified tool developed to enhance the teaching and learning of UML Class Diagrams in academic settings. Although it is not based on AI technologies, it introduces an interactive environment

designed to increase student engagement and support formative assessment, composed of drag-and-drop modeling canvas based on the Apollon open-source library, real-time feedback through visual indicators, categorized error lists, and direct diagram annotations, etc. Its internal evaluation engine compares student-submitted diagrams with a predefined reference solution created by instructors.

2.5 Gen-AI

Gen-AI is a term used to describe computational techniques capable of generating seemingly new, meaningful content such as text, images, or audio from training data. (Feuerriegel et al., 2024). Unlike traditional AI systems, which are based on and follow predefined rules, Gen-AI can generate new content by leveraging advanced models such as Generative Adversarial Networks (GANs), Generative Pretrained Transformers (GPT) and Generative Diffusion Models.

- Generative Adversarial Networks (GANs): frameworks composed of two neural networks, a generator and a discriminator, which are collectively trained to provide practical output.
- Generative Pretrained Transformers (GPTs): massive-scale transformer-based language models skilled in huge text corpora, capable of generating coherent and contextually applicable language.
- Diffusion Models: generative models that learn how to denoise random facts through the years to produce sensible photographs and different media content material.

Thanks to their probabilistic nature, Gen-AI systems can manage a wide spectrum of inputs and adapt responses based on different knowledge. Gen-AI models are not constrained by fixed tools or output formats, which promotes interoperability across educational platforms. Furthermore, they can simulate complex real-world scenarios that are difficult to design manually, offering richer learning environments.

Tools such as ChatGPT, which rely on large language models (LLMs) and advanced machine learning algorithms, have played a pivotal role in popularizing generative artificial intelligence (Gen-AI) among both experts and the general public. Since its launch, ChatGPT has reached an unprecedented scale of adoption, surpassing 100 million users by January 2023 and becoming a mainstream interface for interacting with AI.

While these systems hold the promise of democratizing access to information, supporting creativity, and enhancing productivity across various domains, their widespread deployment has also raised significant concerns. One of the most pressing issues is the potential for the unintentional spread of misinformation. Given that LLMs generate text based on statistical patterns in training data, rather than verified facts, they may produce outputs that are factually incorrect, misleading, or outdated — particularly when operating without explicit sources or mechanisms of accountability.

Moreover, the lack of transparency on these models trainings and the proprietary nature of many foundational architectures makes external auditing more difficult. As a result, users often cannot fully understand how a given response was generated or assess

its reliability. The presence of biases in the training datasets can further compound the problem, leading to biased or even harmful outputs in certain contexts.

These limitations highlight the need for clear governance, ethical design, and technical safeguards like alignment and human oversight. LLMs such as ChatGPT offer great potential but require responsible and risk-aware development. In contrast to rule-based and DSL driven systems traditionally used for generating and evaluating modeling exercises, Gen-AI models offer several significant advantages:

1. First, large-language models (LLMs) are capable of generating diverse and realistic problem variants at scale without requiring predefined templates or manual rule definition, greatly reducing the workload for instructors.

2. Second, their probabilistic and data-driven nature allows them to adapt to a wide range of student inputs, providing more flexible and personalized feedback that can help students at different levels of understanding.

3. Third, Gen-AI can better handle domain-specific or unexpected terminology, as it draws from vast and varied training data, minimizing issues related to rigid syntax matching or limited semantic interpretation.

4. In conclusion, Gen-AI can produce plausible but factually incorrect or entirely fabricated outputs, a phenomenon known as hallucination. This can be especially problematic in educational contexts.

Additionally, these models are not constrained by specific formats or tools, which enhances interoperability between different educational platforms and modeling environments. Finally, Gen-AI systems can also simulate realistic and complex scenarios that would be difficult to design manually, offering richer and more engaging learning experiences for students.

Chapter 3

Methodology

3.1 Dataset construction

In order to proceed with the research study of the effectiveness in using Gen-AI tools for the evaluation and the generation of UML diagram variants, the data set applied must first be provided. By using academic search tools such as Google Scholar and extending the search including previous years material, it was possible to identify exercises with solutions that included prompts in either Italian or English. For consistency reasons, only Italian-written exercises were analysed. Moreover, given that UML modeling language is continually evolving and may adopt different conventions depending on the country or time period (e.g., different symbolisms), exercises that significantly deviated from the standard adopted in the remainder of the dataset were excluded from consideration. However, many of these exercises were sourced from the websites of other Italian universities and authored by different instructors. Hence, they all pertain to the field of computer engineering and are relevant to the context of this research, especially in an academic perspective. As mentioned before, the main limit of evaluating modeling exercises lies in their inherent subjectivity. Therefore, an approach was adopted to ensure maximum objectivity, which, through the application of multiple criteria and scoring methods, allowed the classification of a total of 32 exercises into five difficulty levels, namely very easy, easy, medium, difficult and very difficult. Each criterion represents the difficulty level of an exercise with respect to a specific aspect. Furthermore, each criterion follows specific rules that allow for the assignment of a score ranging from 1 to 5, corresponding to increasing levels of difficulty in each category analyzed. The sum of the scores obtained for each criterion is compared with the final table, which definitively assigns the total difficulty classification of the exercise.

3.1.1 Number of Classes

The number of classes present in the UML diagram serves as an indicator of the size and conceptual scope of the modeled system. Larger diagrams typically require a more complex understanding of entities and their interrelations. All classes are counted, including superclasses and subclasses involved in generalization and specialization hierarchies.

Number of Classes	Score
1–4	1
5–8	2
9–12	3
13–16	4
More than 16	5

Table 3.1. Score based on number of classes

3.1.2 Number of Relationships

This criterion captures the number of relationships and the diversity of relationship types used in the diagram.

Total Relationships	Score
0–4	1
5–8	2
9–12	3
13–16	4
More than 16	5

Table 3.2. Score based on number of relationships

3.1.3 Variety of Relationship Types

This criterion evaluates the diversity of relationship types represented in the UML diagram, regardless of their quantity, such as association, aggregation, generalization, specialization, and different forms of hierarchy. Each distinct UML relationship type, as defined by standard UML semantics, is counted once, regardless of how many instances of that type appear in the diagram. Both structural and dependency-based relationships are considered valid. Abstracted or indirect relationships are only considered if they are explicitly modeled.

Types of Relationships Used	Score
1 type (e.g., only association)	1
2 types	2
3 types	3
4 types	4
5 or more types	5

Table 3.3. Score based on variety of relationship types

3.1.4 Cardinalities Constraints

This criterion considers the presence of cardinalities, role names, and other annotations in the relationships. All explicitly stated multiplicities are taken into account and the presence of role names and annotations adds semantic clarity, increasing the score. Notational completeness (e.g., both ends of an association labeled and constrained) is valued.

Constraints Present	**Score**
No constraints or only fixed values (e.g., 1)	1
Basic cardinalities (e.g., 0..1, 1..*) without roles	2
Cardinalities with 1–2 role names	3
Cardinalities with roles and custom labels	4
Complex intervals (e.g., n..m), named roles, notes or constraints	5

Table 3.4. Score based on structural constraints

3.1.5 Average Number of Attributes per Class

This score is computed by dividing the total number of attributes by the number of classes, rounded up. Operations, or behavioral elements are counted as attributes only in few cases depending on the meaning. Repeated attributes, such as same name and type, across different classes are counted individually.

Average Attributes per Class	**Score**
1	1
2	2
3	3
3–4	4
More than 4	5

Table 3.5. Score based on attribute density

3.1.6 Inheritance Structures

Generalization or specialization hierarchies increase complexity based on depth, annotations, and combinations. Generalization allows one class, named 'child' to inherit attributes and behaviors from another, named 'parent', and its use contributes significantly to the conceptual complexity of the model. At the most basic level, if the diagram does not include any inheritance relationships, it is considered structurally simple in this regard and receives the lowest score. This would typically involve entirely independent classes with no superclass-subclass connections.

Inheritance Complexity	Score
No inheritance	1
One simple generalization	2
Two or more generalizations with abstract/intermediate classes	3
Use of disjoint/complete constraints	4
Multiple inheritance or multi-level specializations	5

Table 3.6. Score based on inheritance complexity

3.1.7 Clarity of the Exercise Text

This criterion reflects how easy or difficult it is to interpret the task based on the formulation of the exercise.For clarity, the presence of structured formatting (e.g., numbered tasks) lowers the score, while open-ended questions or business case style prompts increase complexity. This criterion has a crucial role in the evaluation of s UML class diagram difficulty evaluation, as indeed it represent the most human-like behaviour the LLM has to replicate, extrapolating implicit information hidden in the text.

Textual Clarity	Score
Fully guided and structured instructions	1
Simple language with bullet points	2
Descriptive but clear text	3
Partially implicit or open-ended statements	4
Vague or not specified scenarios requiring interpretation	5

Table 3.7. Score based on clarity of exercise description

3.1.8 Difficulty Classification

The total score is the sum of the six criterion scores. The final classification is based on the following scale:

Total Score	Difficulty Level
6–10	Very Easy
11–14	Easy
15–18	Medium
19–22	Difficult
23–30	Very Difficult

Table 3.8. Difficulty levels based on total score

3.2 Dataset Presentation

Once all the exercises had been collected, they were systematically organized into a dedicated folder. For each exercise, a Word file containing the exercise number and text was included, along with an image file representing the corresponding UML class diagram solution. For the sake of consistency, it was decided that all exercises would be presented in Italian, regardless of whether the human-made solution included class and attribute names in English. The worksheet was then shared on Drive

Using the criteria and rules mentioned above, it was possible to categorize all 32 exercises in the dataset. The assignment of points to each criterion was done manually by comparing the solutions of the UML diagrams linked with the tables previously presented, while the clustering process was carried out using excel and some elementary formulas for calculating the sum of the scores. Each row of the excel sheet has been filled with the ID number of the exercise and each point associated with a criterion.

For example:

#	Cls	Rel	Types	Constr	Attr	Inh	Txt	Score	Diff
1	2	2	3	2	1	3	2	15	Medium

Table 3.9. Example of a UML exercise evaluation row (compact format)

Figure 3.4 presents a pie chart that shows the distribution of difficulty levels across the 32 UML exercise data set.

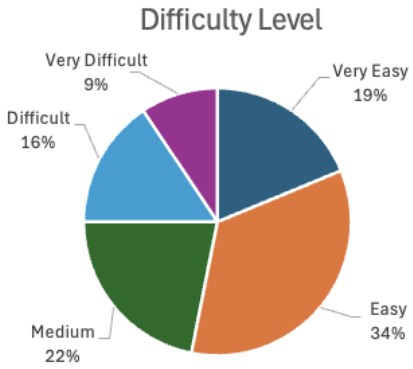

Figure 3.1. Distribution of UML Exercises by Difficulty Level

The largest portion of the pie chart is occupied by the "Easy" category, which represents 34% of the exercises. This indicates that more than one third of the UML exercises in the dataset are considered relatively straightforward in terms of structural complexity and conceptual modeling. These exercises typically involve a limited number of classes and relationships and may feature basic or moderate use of UML features.

The second most frequent difficulty level is "Medium", representing 22% of the exercises. These exercises tend to strike a balance between simplicity and complexity, often introducing a wider variety of relationships and perhaps some limited use of inheritance or structural constraints.

The "Very Easy" category constitutes 19% of the dataset. The exercises in this group are minimal in complexity, typically including very few classes and relationships, little to no inheritance, and basic cardinalities.

The "Difficult" level includes the 16% of the exercises. These are more intricate tasks, often involving multiple inheritance structures, disjoint constraints, or greater attribute density. Interpreting these exercises may require a deeper understanding of UML semantics and modeling strategies.

Finally, the "Very Difficult" category makes up the smallest portion of the dataset, with 9% of exercises falling under this classification. These tasks usually feature a combination of high-class count, diverse and nested relationships, advanced cardinality constraints, multiple inheritance, and textual complexity. They challenge not only technical UML knowledge, but also interpretation and abstraction skills.

Overall, the chart provides a clear overview of the range and distribution of UML modeling complexity across the dataset, with a notable skew toward the simpler (*Easy* and *Very Easy*) categories. Hence, the result obtained from clustering is sufficiently homogeneous: it might reflect a Gaussian distribution, with particular emphasis on easy and medium cases, which are the most common categories, and with few cases of very difficult and very easy categories, which are borderline cases.

3.3 Introduction to the Prompt-Based Generation

In this work, a prompt-driven approach was adopted to generate UML class diagrams from textual descriptions, with the goal of analyzing the correctness of the results obtained from the Gen-Ai tools and comparing them with the human-made ones. The prompt simulates a realistic instructional setting by positioning the model as a university-level software engineering professor engaged in the creation of class diagram exercises (role-based prompting). Specifically, the prompt instructs the model to interpret a natural language description of a system and return a JSON-formatted representation of a UML reference solution, which is a "easy to read and write" text format. This includes classes, attributes, and associations to maintain focus on the structural components of object-oriented design.

The complete prompt provided to the model is as follows:

```
You are a software engineering professor who is designing a UML class
    diagram modeling exercise for your students. Starting from a textual
    description of a system, you must create a reference solution in JSON
    format that is fully compatible with the Apollon UML tool (https://
    apollon.ase.in.tum.de).

The JSON output must accurately represent:
```

```
- Classes and their attributes (no methods)
- Associations (bidirectional, aggregations, compositions)
- Generalizations (inheritance hierarchies)

Follow these strict modeling guidelines:

1. Attribute rules:
   - Only simple types are allowed: string, int, float, date
   - A class cannot have another class as an attribute
   - No collections: avoid lists, sets, maps
   - Use associations instead of composite types

2. Relationships:
   - If the text suggests specializations (e.g., types of users or
     entities), model them using generalizations:
     - Use "type": "ClassBidirectional"
     - Set "name": "is-a" and leave multiplicities empty
   - If the text implies ownership or whole-part structure, use
     composition:
     - Model it as "type": "ClassAggregation"
     - Set "multiplicity": "1" on the composite (whole) side
   - If the relationship is an aggregation (e.g., "X has many Ys"), use:
     - "type": "ClassAggregation"
     - The diamond must be on the aggregating class side
   - For ordinary associations (peer-to-peer), use "ClassBidirectional"

3. The output JSON must always start with the following exact structure:

{
  "version": "3.0.0",
  "type": "ClassDiagram",
  "size": { "width": 1600, "height": 780 },
  "interactive": {
    "elements": {},
    "relationships": {}
  },
  "assessments": {},

This header must be present exactly as shown above.

Important: After this header, you must continue the JSON with a complete
    declaration of:
- "elements": { ... }
- "relationships": { ... }

These must be written outside and separately from the "interactive" block.

Do not insert any class, attribute, or relationship inside "interactive".
    That block must remain empty as required by Apollon format for
    compatibility.

4. Class formatting (elements dictionary):
   - Each class must be represented as an object with a UUIDv4 key and
     include:
```

```
     - id, name, "type": "Class", owner, bounds, attributes, and an empty
         methods list

5. Each attribute must:
   - Appear immediately after its owning class
   - Have: id, name = "+ attributeName: Type", "type": "ClassAttribute",
       owner, bounds

6. Relationship formatting (relationships dictionary):
   - Each relationship must include: id, type, name, bounds, path, source,
       target, and isManuallyLayouted = false
   - Source and target must specify:
     - element, direction, multiplicity
     - Optional role for disambiguation

7. Guidelines for semantic clarity:
   - If multiple associations exist between the same pair of classes,
       label them clearly with name and use role in source/target when
       necessary
   - When a single class plays multiple roles, model each relationship
       separately and specify their roles
   - Always include intermediate classes if they improve semantic clarity
   - Use associative classes for relationships that have their own
       attributes
   - Always assign realistic multiplicities: use 1, 0..1, 0..n, or 1..n as
       implied by the domain

8. Handling ambiguous or underspecified text:
   - If the description is vague or incomplete, assume a complete
       pedagogical model
   - Include abstract/general classes, useful associations, and
       appropriate multiplicities to enrich the diagram
   - Do not omit any class or relationship that is logically or
       semantically inferable

9. Naming conventions:
   - Use meaningful, consistent, and grammatically correct names for
       classes, attributes, and relationships as they are specified in the
       text
   - Avoid generic or ambiguous labels like "thing", "info", or "writes"
   - For clarity, use phrases like "is supervised by" instead of active
       voice ("supervises")

10. Minimum Modeling Completeness:
In every exercise, treat the goal as pedagogical. Even if the textual
    description is minimal, you must model a semantically rich and
    educationally useful class diagram. Always include:

At least one generalization if there are subtypes (e.g., types of policies
    )
At least one composition or aggregation if whole-part relations are
    implied
At least three realistic attributes for each main class
item Clarify ambiguous terms with appropriate modeling constructs (
    enumerations, abstract classes, or associative classes)
```

```
\end{itemize}

Return only the complete JSON object. Do not include any explanation,
    comment, or markdown formatting
```

The prompt enforces a number of modeling constraints to ensure consistency and adherence to UML best practices. For example, it avoids class nesting or collection types within attributes and disallows the inclusion of empty classes unless they can be reinterpreted as attributes or roles.
Furthermore, when generating UML class diagrams compatible with the Apollon JSON schema, several modeling conventions differ from standard UML practices:

- Instead of using the standard UML generalization arrow (a line with an unfilled triangular head), inheritance relationships are modeled as ClassBidirectional associations labeled with the name "is-a". The semantic meaning is encoded in the relationship name rather than through a dedicated UML construct.
- The prompt distinguishes only between ClassAggregation and ClassBidirectional relationships. Both composition and aggregation are expressed using ClassAggregation. The intended semantics are conveyed through the direction of the aggregation diamond and multiplicities, rather than using a separate ClassComposition type.
- Attributes are restricted to primitive types only (string, int, float, date). A class cannot include an attribute whose type is another class. Instead, all structural connections between classes must be modeled as explicit associations.
- Unlike XMI or other UML export formats, which organize model elements in a hierarchical structure (e.g., attributes nested within classes), the Apollon format stores all elements in a flat dictionary under "elements" and all relationships under "relationships". Attributes are defined as separate ClassAttribute objects and methods are entirely excluded.
- When multiple associations exist between the same classes, or when a class participates in multiple roles, the prompt requires the use of explicit names and roles for relationships to ensure semantic clarity and prevent ambiguity.

These modeling constraints are specifically designed to maximize compatibility with the Apollon UML editor, facilitate automated parsing, and support the generation of pedagogically meaningful class diagrams in educational settings.

3.3.1 Iterative Prompt Refinement

To reach the final version of the prompt, an iterative trial-and-error approach was adopted. Starting from a basic prompt, multiple refinements were performed, each time comparing the generated JSON output with the reference training example, which had been exported from the UML modeler after manually creating the diagram. This evolution was driven by the need to improve the precision, structural validity, and semantic alignment of the

model output. Each version addressed limitations encountered in the previous one, culminating in a final prompt capable of producing consistent and pedagogically meaningful class diagrams.

1. First versions of the prompt were loosely defined instructions that asked the model to generate a UML class diagram in JSON format, based on a textual description of a system. They lacked a formal structure and provided no details on the format expected by Apollon or the rules of UML modeling. While it could produce a basic diagram, it did not define how to represent class attributes or associations. It offered no guidance on generalizations, compositions, or multiplicities and it was unclear whether the model should include only classes, or also relationships and structure. The following prompt is the first attempt:

```
You are a software engineering professor who is designing a UML class
     diagram modeling exercise for your students. Starting from a
    textual description of a system, you must create a reference
    solution in JSON format that is fully compatible with the Apollon
    UML tool (https://apollon.ase.in.tum.de).

The JSON output must accurately represent:
- Classes and their attributes (no methods)
- Associations (bidirectional, aggregations, compositions)
- Generalizations (inheritance hierarchies)

Follow these strict modeling guidelines:

1. Attribute rules:
   - Only simple types are allowed: string, int, float, date
   - A class cannot have another class as an attribute
   - No collections: avoid lists, sets, maps
   - Use associations instead of composite types

2. Relationships:
-       If the text suggests specializations (e.g., types of users or
     entities), model them using generalizations:
o       Use "type": "ClassBidirectional"
o       Set  "name": "is-a" and leave multiplicities empty
-       If the text implies ownership or whole-part structure, use
    composition:
o       Model it as "type": "ClassAggregation"
o       Set "multiplicity": "1" on the composite (whole) side
-       If the relationship is an aggregation (e.g., "X has many Ys")
    , use:
o       "type": "ClassAggregation"
o       The diamond must be on the aggregating class side
-       For ordinary associations (peer-to-peer), use "
    ClassBidirectional"

3. Output JSON structure must be ALWAYS:
-       "version": "3.0.0"
-       "type": "ClassDiagram"
-       "size": { "width": 1600, "height": 780 }
-       "interactive" must be  "interactive": {
```

```
"elements": {},
"relationships": {}
}
-       "assessments"` must be an empty object: `"assessments": {}`

Once you presented point 3 as it is written, you can proceed with:
4. Class formatting (elements dictionary):
   - Each class must be represented as an object with a UUIDv4 key
     and include:
     - `id`, `name`, `"type": "Class"`, `owner`, `bounds`, `
       attributes`, and empty `methods` list

5. Each attribute must:
     - Appear immediately after its owning class
     - Have: `id`, `name = "+ attributeName: Type"`, `"type": "
       ClassAttribute"`, `owner`, `bounds`

6. Relationship formatting (relationships dictionary):
   - Each relationship must include: `id`, `type`, `name`, `bounds`,
     `path`, `source`, `target`, `   isManuallyLayouted   = false `
   - Source and target must specify:
     - `element`, `direction`, `multiplicity`
     - Optional `role` for disambiguation

7. Guidelines for semantic clarity:
   - If multiple associations exist between the same pair of classes
     (e.g., `Thesis` and `File`), label them clearly with `"name"`
     and use `"role"` in source/target when necessary
   - When a single class plays multiple roles (e.g., `Professor` as `
     supervisor` and `co-supervisor`), model each relationship
     separately and specify their roles
   - Always include intermediate classes (e.g., `StaffMember`, `
     AssociationRecord`) if they improve semantic clarity
   - Use associative classes for relationships that have their own
     attributes (e.g., `Enrollment`, `Period`, `Assignment`)
   - Always assign realistic multiplicities: use `1`, `0..1`, `0..n`,
      or `1..n` as implied by the domain

8. Handling ambiguous or underspecified text:
   - If the description is vague or incomplete, assume a complete
     pedagogical model
   - Include abstract/general classes, useful associations, and
     appropriate multiplicities to enrich the diagram
   - Do not omit any class or relationship that is logically or
     semantically inferable

9. Naming conventions:
   - Use meaningful, consistent, and grammatically correct names for
     classes, attributes, and relationships
   - Avoid generic or ambiguous labels like `"thing"`, `"info"` or `"
     writes"`
   - For clarity, use phrases like `"is supervised by"` instead of
     active voice (`"supervises"`)
```

```
Return only the complete JSON object. Do not include any explanation,
    comment, or markdown formatting.
```

As a result, the outputs created only some parts of the entire diagram as shown in the figure below.

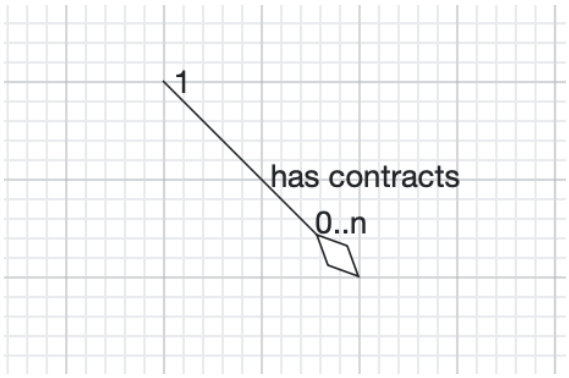

Figure 3.2. UML Modeler output based on Prompt first version

2. From the previous prompt attributes were mistakenly expected to be defined as separate objects of type "ClassAttribute" and placed externally in the "elements" dictionary, independent of their owning class. Even when classes were correctly recognized, their attributes were not rendered in the expected way, causing the diagram to appear incomplete or empty in the Apollon interface.

   The corrected and final prompt explicitly addresses this issue by enforcing that attributes be embedded directly within each class under the "attributes" key, using the following textual instruction:

   "Each class must be declared as an object with [...] 'attributes': a list of strings in the format '+ attributeName: Type'" This change aligns perfectly with the Apollon specification and ensures that class definitions are self-contained and rendered properly. By internalizing the attributes within their respective class declarations, the prompt avoids fragmentation and enables consistent output that can be visualized and validated within the Apollon environment.

   However, the diagrams created based on the outut generated from this prompt were basically the same of the previous ones.

3. After providing the correct JSON output of the training exercise as a reference, the prompt was refined to ensure that its usage could produce different outputs while still strictly adhering to the rules applied in the training example. The prompt produced was the following:

```
You are a software engineering professor designing a UML class
    diagram modeling exercise for your students. Based on a textual
    description of a system, you must generate a complete JSON
    representation that is fully and strictly compatible with the
    Apollon UML tool (https://apollon.ase.in.tum.de), matching the
    structure of the reference file provided.

The JSON output must:

1. Start with this exact header:
{
  "version": "3.0.0",
  "type": "ClassDiagram",
  "size": { "width": 1180, "height": 880 },
  "interactive": {
    "elements": {},
    "relationships": {}
  },

2. Then define all class elements under the `"elements"` dictionary.
    Each class must be declared as a separate object with:
   - "id": unique UUIDv4
   - "name": the name of the class
   - "type": "Class"
   - "owner": null
   - "bounds": with x, y, width and height (e.g., width: 160, height:
       130)
   - "attributes": a list of UUID references to attribute objects
   - "methods": an empty list

3. Every attribute must be declared as a separate object in `"
    elements"`:
   - "id": unique UUIDv4
   - "name": formatted as "+ attributeName: Type" (e.g., "+ title:
      string")
   - "type": "ClassAttribute"
   - "owner": UUID of the class it belongs to
   - "bounds": x, y, width, height

4. Define all relationships in the `"relationships"` dictionary. Each
     relationship must include:
   - "id": unique UUIDv4
   - "name": the label of the association (e.g., "is supervised by")
   - "type": one of: "ClassBidirectional", "ClassAggregation", "
      ClassComposition"
   - "bounds" and "path": use at least two points, values can be
      placeholders
   - "source" and "target": include:
       - "element": class UUID
       - "direction": one of "Left", "Right", "Up", "Down", "
          Bottomleft"
       - "multiplicity": e.g., "1", "*", "0..n", or "" if omitted
       - "role": optional but useful for disambiguation
   - "isManuallyLayouted": false
```

```
5. Conclude the JSON with:
  "assessments": {}
}

Additional modeling rules:

- Include at least three meaningful classes and realistic
    relationships derived from the exercise.
- Each class must have at least from 2 to 3 attributes (as separate
    objects).
- Use generalization (ClassBidirectional with name "is-a") if the
    domain includes subtypes.
- Use aggregation or composition if whole-part structures are implied
    .
- Always specify multiplicities and meaningful names for associations
    .
- Ensure UUIDs are unique and consistent for each entity.

Only output a complete and valid JSON object, strictly following the
    format described above. Do not include comments, formatting, or
    explanations.

Begin generation from the following exercise description:
```

The result obtained from this new version allowed for the correct visualization of the classes and their corresponding attributes, as shown in the figure below, but not of the respective associations.

4. Finally, the last refinement focused specifically on the handling of relationships, placing particular emphasis on the 'bounds', which define the classes involved, and the 'path', which represents the trajectory of the association arrow. This final adjustment led to the development of the latest version of the prompt previously presented.

Prompt Engineering Techniques

The final version of the prompt, the result of a progressive refinement strategy, systematically incorporated several advanced prompt engineering techniques, each of which contributed to guiding the LLM toward structurally valid, semantically coherent, and Apollon or UML Modeler compatible outputs.

- **Role prompting:** The prompt explicitly assigns the role of a "software engineering professor specialized in UML modeling" to the LLM. This technique encourages disciplined and domain-aware generation behavior, aligning the model's style with academic expectations.
- **Few Shots Technique:** The prompt includes complete input–output examples (text + JSON), fully compatible with Apollon. These serve as implicit demonstrations of the expected behavior, reinforcing the correct structure and helping the model align textual descriptions with diagrammatic outputs.

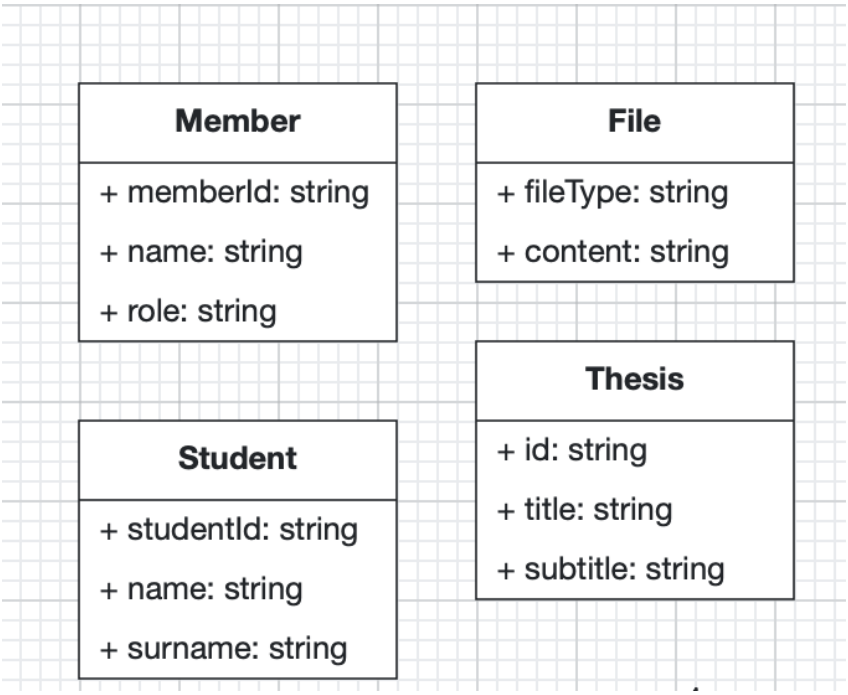

Figure 3.3. UML Modeler output based on Prompt third version

- **Binding instructions:** Each structural and syntactic constraint (e.g., the placement of "elements" and "relationships", the presence of "owner", "bounds", and "path") is expressed through repetitive and unambiguous directives. This repetition increases the likelihood that the model retains these patterns even in longer prompts.
- **Semantic specificity:** The prompt provides precise modeling guidelines for advanced UML constructs such as compositions, aggregations, generalizations, and associative classes. It explicitly dictates the use of specific relationship types ("ClassComposition", "ClassAggregation", "ClassBidirectional") and instructs the model to label inheritance relations as "is-a".
- **Linguistic constraints:** Clear recommendations are given to avoid ambiguous or generic labels (e.g., "thing", "info"), and to prefer passive and descriptive naming for relationships (e.g., "is supervised by"), in order to enhance the semantic readability and clarity of the diagram.

3.3.2 Reference Exercise for Output Calibration

In addition to the prompt, it was also necessary to leverage a training input-output pair to further refine the behavior of the model.

Hence, both the text and the JSON output of the training exercise were used not only to guide the model generate the final correct prompt, but also to establish the proper

setting for the model in order to generate correct output for the following exercises.

Specifically, the reference solution to the exercise description, or *input*, as explained before, had to be manually recreated in Apollon as a UML class diagram, allowing the corresponding .json file, or *output*, to be downloaded.

This automated technique falls under the *few-shot* and *role-based* prompting paradigms discussed earlier. In the few-shot setting, the LLM is not trained or fine-tuned on the task in a traditional supervised learning sense, but rather guided through a small number of illustrative examples that define the desired input-output behavior. These examples serve as implicit instructions, enabling the model to infer the task structure and generalize to new, unseen instances with minimal supervision. In addition to this, the role-based aspect of the prompt involves assigning the model a specific identity or persona, such as 'You are a UML parser assistant' or 'You are a conceptual modeling expert', which helps condition its responses within a well-defined context and set of expectations. This combination of minimal examples and contextualized instruction allows the LLM to simulate task understanding and align its generation with domain-specific requirements, without the need for additional architectural changes or fine-tuning. This approach has proven particularly effective for structured output generation, where maintaining consistency, terminology, and formal constraints is essential. Furthermore, the iterative refinement of the prompt using a set of training exercises analyzing model errors, adjusting the prompt accordingly, and re-evaluating the output reflects a methodology aligned with emerging promptware engineering practices. By leveraging test cases, the prompt was progressively adapted to better handle ambiguity and semantic coverage, but also comply with the syntactic structure required by the Apollon format. This engineering-oriented approach ensured that the prompt not only guided the model effectively, but also remained robust and reusable across a diverse set of modeling tasks.

The following figure shows the original 'human-made' diagram, which was manually replicated in Apollon.

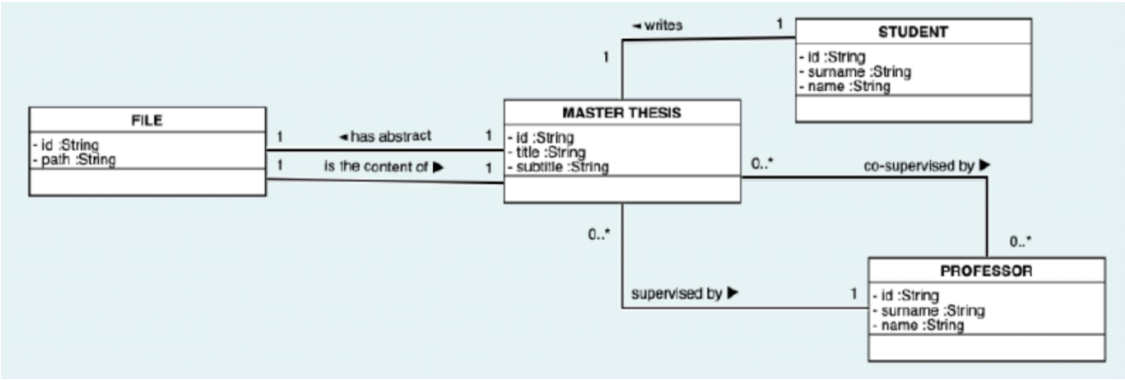

Figure 3.4. Human made solution of the training exercise

The manual replication of the figure above on Apollon, enabled us to download the respcctive JSON file.

```
{
    "version": "3.0.0",
    "type": "ClassDiagram",
    "size": {
        "width": 1180,
        "height": 880
    },
    "interactive": {
        "elements": {},
        "relationships": {}
    },
    "elements": {
        "be41f224-93b5-410e-94a7-1
            c5884e99866": {
            "id": "be41f224-93b5-410
                e-94a7-1c5884e99866
                ",
            "name": "File",
            "type": "Class",
            "owner": null,
            "bounds": {
                "x": -570,
                "y": 200,
                "width": 160,
                "height": 100    },
            "attributes": [
                "4eb437ff-2433-444c
                    -80f9-665
                    a4d5f1c37",
                "7d43b52d-c07e-4e14-
                    a557-9
                    a5748ba9d9d"
            ],
            "methods": []
        },
        "4eb437ff-2433-444c-80f9-665
            a4d5f1c37": {
            "id": "4eb437ff-2433-444
                c-80f9-665a4d5f1c37
                ",
            "name": "+ id: string",
            "type": "ClassAttribute
                ",
            "owner": "be41f224-93b5
                -410e-94a7-1
                c5884e99866",
            "bounds": {
                "x": -569.5,
                "y": 240.5,
                "width": 159,
                "height": 30   }
        },
        "7d43b52d-c07e-4e14-a557-9
            a5748ba9d9d": {
            "id": "7d43b52d-c07e-4
                e14-a557-9
                a5748ba9d9d",
            "name": "+ path: string
                ",
            "type": "ClassAttribute
                ",
            "owner": "be41f224-93b5
                -410e-94a7-1
                c5884e99866",
            "bounds": {
                "x": -569.5,
                "y": 270.5,
                "width": 159,
                "height": 30  }
        },
        "15c4d455-ad8c-4e83-ade5-
            fc8db4af22db": {
            "id": "15c4d455-ad8c-4
                e83-ade5-
                fc8db4af22db",
            "name": "Student",
            "type": "Class",
            "owner": null,
            "bounds": {
                "x": 30,
                "y": 20,
                "width": 160,
                "height": 130   },
            "attributes": [
                "472916a3-e32e-4c04
                    -8235-
                    b13d00197ee5",
                "c50cdb0a-00b6-4f15-
                    b888-
                    a5a4ca761d2f",
                "76164a21-8600-47dc
                    -905b-
                    e741894f12ad"
            ],
            "methods": []
        },
        "472916a3-e32e-4c04-8235-
            b13d00197ee5": {
            "id": "472916a3-e32e-4
                c04-8235-
                b13d00197ee5",
            "name": "+ id: String",
            "type": "ClassAttribute
                ",
            "owner": "15c4d455-ad8c
                -4e83-ade5-
                fc8db4af22db",
```

```
            "bounds": {
                "x": 30.5,
                "y": 60.5,
                "width": 159,
                "height": 30  }
        },
        "c50cdb0a-00b6-4f15-b888-
            a5a4ca761d2f": {
            "id": "c50cdb0a-00b6-4
                f15-b888-
                a5a4ca761d2f",
            "name": "+ surname:
                String",
            "type": "ClassAttribute
                ",
            "owner": "15c4d455-ad8c
                -4e83-ade5-
                fc8db4af22db",
            "bounds": {
                "x": 30.5,
                "y": 90.5,
                "width": 159,
                "height": 30  }
        },
        "76164a21-8600-47dc-905b-
            e741894f12ad": {
            "id": "76164a21-8600-47
                dc-905b-e741894f12ad
                ",
            "name": "+ name: String
                ",
            "type": "ClassAttribute
                ",
            "owner": "15c4d455-ad8c
                -4e83-ade5-
                fc8db4af22db",
            "bounds": {
                "x": 30.5,
                "y": 120.5,
                "width": 159,
                "height": 30 }
        },
        "8ed19bb7-4815-4230-9020-74
            df970fbd6b": {
            "id": "8ed19bb7
                -4815-4230-9020-74
                df970fbd6b",
            "name": "Professor",
            "type": "Class",
            "owner": null,
            "bounds": {
                "x": 110,
                "y": 290,
                "width": 160,
                "height": 130
            },
            "attributes": [
                "fa24fdb6-7a6a-4cca-
                    b3e7-
                    d8bf2090b05f",
                "ad1339f1-81aa-4ddf-
                    aaf8-
                    d1c66d811619",
                "c0e15e73-6cf2-4990-
                    bd75-4
                    d5358b28433"
            ],
            "methods": []
        },
        "fa24fdb6-7a6a-4cca-b3e7-
            d8bf2090b05f": {
            "id": "fa24fdb6-7a6a-4
                cca-b3e7-
                d8bf2090b05f",
            "name": "+ id: String",
            "type": "ClassAttribute
                ",
            "owner": "8ed19bb7
                -4815-4230-9020-74
                df970fbd6b",
            "bounds": {
                "x": 110.5,
                "y": 330.5,
                "width": 159,
                "height": 30   }
        },
        "ad1339f1-81aa-4ddf-aaf8-
            d1c66d811619": {
            "id": "ad1339f1-81aa-4
                ddf-aaf8-
                d1c66d811619",
            "name": "+ surname:
                String",
            "type": "ClassAttribute
                ",
            "owner": "8ed19bb7
                -4815-4230-9020-74
                df970fbd6b",
            "bounds": {
                "x": 110.5,
                "y": 360.5,
                "width": 159,
                "height": 30    }
        },
        "c0e15e73-6cf2-4990-bd75-4
            d5358b28433": {
            "id": "c0e15e73-6cf2
                -4990-bd75-4
                d5358b28433",
```

```
            "name": "+ name: String
                ",
            "type": "ClassAttribute
                ",
            "owner": "8ed19bb7
                -4815-4230-9020-74
                df970fbd6b",
            "bounds": {
                "x": 110.5,
                "y": 390.5,
                "width": 159,
                "height": 30   }
        },
        "f653690d-6622-41cb-b798-4
            f0be27a5a30": {
            "id": "f653690d-6622-41
                cb-b798-4f0be27a5a30
                ",
            "name": "Master's Thesis
                ",
            "type": "Class",
            "owner": null,
            "bounds": {
                "x": -280,
                "y": 160,
                "width": 160,
                "height": 130
            },
            "attributes": [
                "8cffe93f-c617-41e7-
                    b734-730
                    ca9c80563",
                "c71a98db-a336-4460-
                    b9c4-73
                    a46ee9eef5",
                "19bb6890-2b13-4eb2
                    -9066-72
                    fddffb23f9"
            ],
            "methods": []
        },
        "8cffe93f-c617-41e7-b734-730
            ca9c80563": {
            "id": "8cffe93f-c617-41
                e7-b734-730ca9c80563
                ",
            "name": "+ id: string",
            "type": "ClassAttribute
                ",
            "owner": "f653690d
                -6622-41cb-b798-4
                f0be27a5a30",
            "bounds": {
                "x": -279.5,
                "y": 200.5,
                "width": 159,
                "height": 30   }
        },
        "c71a98db-a336-4460-b9c4-73
            a46ee9eef5": {
            "id": "c71a98db-a336
                -4460-b9c4-73
                a46ee9eef5",
            "name": "+ title: string
                ",
            "type": "ClassAttribute
                ",
            "owner": "f653690d
                -6622-41cb-b798-4
                f0be27a5a30",
            "bounds": {
                "x": -279.5,
                "y": 230.5,
                "width": 159,
                "height": 30
            }
        },
        "19bb6890-2b13-4eb2-9066-72
            fddffb23f9": {
            "id": "19bb6890-2b13-4
                eb2-9066-72
                fddffb23f9",
            "name": "+ subtitle:
                string",
            "type": "ClassAttribute
                ",
            "owner": "f653690d
                -6622-41cb-b798-4
                f0be27a5a30",
            "bounds": {
                "x": -279.5,
                "y": 260.5,
                "width": 159,
                "height": 30     }
    }
  },
  "relationships": {
        "ded41504-d840-43c6-bd29
            -603621176c86": {
            "id": "ded41504-d840-43
                c6-bd29-603621176c86
                ",
            "name": "writes",
            "type": "
                ClassBidirectional",
            "owner": null,
            "bounds": {
                "x": -213.90625,
                "y": 75,
                "width": 243.90625,
```

```
                "height": 95.65625
            },
            "path": [
                {
                    "x": 243.90625,
                    "y": 10
                },
                {
                    "x": 13.90625,
                    "y": 10
                },
                {
                    "x": 13.90625,
                    "y": 85
                }
            ],
            "source": {
                "direction": "Left",
                "element": "15c4d455
                    -ad8c-4e83-ade5-
                    fc8db4af22db",
                "multiplicity": "1",
                "role": ""
            },
            "target": {
                "direction": "Up",
                "element": "f653690d
                    -6622-41cb-b798
                    -4f0be27a5a30",
                "multiplicity": "",
                "role": "1"
            },
            "isManuallyLayouted":
                false
        },
        "7656c7a2-ee37-473b-b20b-
            a567a43a4380": {
            "id": "7656c7a2-ee37-473
                b-b20b-a567a43a4380
                ",
            "name": "is co-
                supervised by",
            "type": "
                ClassBidirectional",
            "owner": null,
            "bounds": {
                "x": -120,
                "y": 215,
                "width": 320.640625,
                "height": 85.65625
            },
            "path": [
                {
                    "x": 0,
                    "y": 10
                },
                {
                    "x": 310,
                    "y": 10
                },
                {
                    "x": 310,
                    "y": 75
                }
            ],
            "source": {
                "direction": "Right
                    ",
                "element": "f653690d
                    -6622-41cb-b798
                    -4f0be27a5a30",
                "multiplicity": "*",
                "role": ""
            },
            "target": {
                "direction": "Up",
                "element": "8ed19bb7
                    -4815-4230-
                    9020-74df970fbd6b",
                "multiplicity": "*",
                "role": ""
            },
            "isManuallyLayouted":
                false
        },
        "10adda4a-07d2-464e-8df6-
            c9251ccb2e59": {
            "id": "10adda4a-07d2-464
                e-8df6-c9251ccb2e59
                ",
            "name": "is supervised
                by",
            "type": "
                ClassBidirectional",
            "owner": null,
            "bounds": {
                "x": -205,
                "y": 290,
                "width": 315,
                "height": 104.65625
            },
            "path": [
                {
                    "x": 5,
                    "y": 0
                },
                {
                    "x": 5,
                    "y": 65
                },
```

```
            {
                "x": 315,
                "y": 65
            }
        ],
        "source": {
            "direction": "Down",
            "element": "f653690d
                -6622-41cb-b798
                -4f0be27a5a30",
            "multiplicity": "*",
            "role": ""
        },
        "target": {
            "direction": "Left",
            "element": "8ed19bb7
                -4815-4230-9020
            -74df970fbd6b",
            "multiplicity": "1",
            "role": ""
        },
        "isManuallyLayouted":
            false
    },
    "7903b4af-b394-415c-b042-
        cb933e51ed00": {
        "id": "7903b4af-b394-415
            c-b042-cb933e51ed00
            ",
        "name": "has abrastract
            ",
        "type": "
            ClassBidirectional",
        "owner": null,
        "bounds": {
            "x": -495,
            "y": 140,
            "width": 215,
            "height": 124.65625
        },
        "path": [
            {
                "x": 215,
                "y": 85
            },
            {
                "x": 175,
                "y": 85
            },
            {
                "x": 175,
                "y": 0
            },
            {
                "x": 5,
                "y": 0
            },
            {
                "x": 5,
                "y": 60
            }
        ],
        "source": {
            "direction": "Left",
            "element": "f653690d
                -6622-41cb-b798
                -4f0be27a5a30",
            "multiplicity": "1",
            "role": ""
        },
        "target": {
            "direction": "Up",
            "element": "be41f224
                -93b5-410e-94a7
                -1c5884e99866",
            "multiplicity": "1",
            "role": ""
        },
        "isManuallyLayouted":
            false
    },
    "6a3dabfe-db4d-4e2b-b366-9
        ad223d24f8c": {
        "id": "6a3dabfe-db4d-4
            e2b-b366-9
            ad223d24f8c",
        "name": "is the content
            of",
        "type": "
            ClassBidirectional",
        "owner": null,
        "bounds": {
            "x": -495,
            "y": 290,
            "width": 268.90625,
            "height": 50
        },
        "path": [
            {
                "x": 5,
                "y": 10
            },
            {
                "x": 5,
                "y": 50
            },
            {
                "x": 255,
                "y": 50
            },
```

```
            {
                "x": 255,
                "y": 0
            }
        ],
        "source": {
            "direction": "Down",
            "element": "be41f224
                -93b5-410e-94a7
                -1c5884e99866",
            "multiplicity": "1",
            "role": ""
        },
        "target": {
            "direction": "
                Bottomleft",
            "element": "f653690d
                -6622-41cb-b798
                -4f0be27a5a30",
            "multiplicity": "1",
            "role": ""
        },
        "isManuallyLayouted":
            false
    }
  },
  "assessments": {}
}
```


Below is the Italian-language text used to train the LLM for the prompt setting:

Un'università sta riprogrammando il suo sistema informativo affinché possa gestire anche le tesi magistrali. Ciascuna tesi si distingue dall'altra grazie ad un ID, ad un titolo e ad un sottotitolo. Ogni tesi è associata a due file: il primo contiene l'abstract e il secondo contiene l'intero contenuto della tesi. Ogni tesi è scritta da un singolo studente, ha un relatore e può avere uno o più correlatori. Per semplicità si assume che sia il relatore che i correlatori siano membri dell'università.

3.3.3 Quality Assessment of Produced ChatGPT Diagrams

To assess the quality and correctness of the diagrams obtained importing the file output in JSON format by ChatGPT on UML Modeler, many approaches had been taken into consideration. The correctness of the diagrams was evaluated avoiding any comparison between the human-made diagrams since the analogy was properly researched after this section. Evaluating correctness based on a reference diagram would have introduced circularity into the experimental design. Hence, alternative strategies were investigated:

- The first method relied on a parsing-based comparison between two sets of structured representations: one extracted from the original textual prompt and the other derived from the Apollon-generated JSON output. However, this approach was not successful due to a recurring issue of structural mismatch between the textual input and the generated JSON file. Specifically, the original Italian descriptions often omitted many attributes that were either implied or left entirely unspecified. In contrast, the JSON output generated by ChatGPT typically included additional attributes, since the prompting strategy explicitly instructed the model to enrich the diagrams with semantically meaningful attributes, even when these were not explicitly mentioned in the text. As a result, sentence-level parsing of the two sources (text and JSON) produced outputs of significantly different lengths and granularity. This imbalance introduced inconsistencies in the comparative analysis, making the parsing approach unsuitable for systematic or objective correctness evaluation.

Similar difficulties have been reported in recent research questioning the reliability of parsing-based comparisons for LLM-generated content [8].

- Another method we tried was using sentence-BERT to compare the meaning of short sentences. The idea was to take statements written from the original exercise text and compare them to similar statements created by analyzing the generated UML diagram. Using sentence-BERT, we turned each sentence into a vector (a type of number-based representation) and then measured how similar the two sets of sentences were. Although this method gave us some useful insights, it did not work very well in practice because it was very sensitive to how the sentences were written: if two sentences meant the same thing but used different words, the model sometimes failed to recognize them as similar. Due to these problems, we decided that this method was not reliable enough for a consistent evaluation. Rubric-based evaluation was adopted as a more efficient and transparent method to assess the correctness of the diagram. Inspired by recent literature in the field (e.g., Toward Automated UML Diagram Assessment: Comparing LLM-Generated Scores with Teaching Assistants), this rubric allowed for structured, criterion-based scoring without relying on a fixed reference model, ensuring a more objective and scalable evaluation process.

3.4 Rubric-based Evaluation of Diagram Correctness

As a result, a rubric-based evaluation was adopted as a more efficient and transparent method to assess the correctness of the diagram. Inspired by recent literature in the field [9], this rubric allowed structured criterion-based scoring without relying on a fixed reference model, ensuring a more objective and scalable evaluation process.

The rubric was composed of seven carefully defined criteria, each capturing a fundamental dimension of structural model quality. These ranged from the presence and completeness of classes and relationships, to semantic accuracy, naming conventions, and overall consistency. A penalty mechanism was also included to address major conceptual omissions or semantic contradictions.

The explicit weighting of each dimension (20/20/20/20/10/10/-10) allowed for fine-grained differentiation between diagram quality levels, while the normalization procedure ensured comparability across exercises.

The choice of criteria was grounded in conceptual modeling literature and teaching practice. For example, "Completeness of Classes" and "Attributes" reflect the capacity of the model to faithfully translate domain knowledge into structural entities. "Semantic Correctness of Relationships" was introduced as a distinct criterion to emphasize not only the presence of associations but their correct directionality, multiplicity, and UML type (e.g., association vs. generalization). Naming clarity, while less critical structurally, was included for its relevance in pedagogical and collaborative modeling contexts.

Each diagram's raw score was normalized to a [0, 1] range and mapped to a categorical tier:

- Under 80%: Under Expectation

- 81–85%: Minimum Standard
- 86–90%: Acceptable
- 91–95%: Proficient
- 96–100%: Above Expectation

This mapping was used not only for interpretability but also to identify broader trends in model quality. For instance, it allowed recognition of common mistakes among mid-tier models, as well as structural patterns that distinguish excellent diagrams.

Case Example Exercise 1

To illustrate the application of the rubric, Exercise number 1 has been taken as a descriptive example. Hence, the Italian text of the exercise and the Diagram generated by LLM are below provided.

Figure 3.5. Italian Text of Exercise 1

```
Esercizio 1 Testo:

Descriviamo una compagnia di assicurazione.
Siamo interessati a descrivere il dominio del problema,
con particolare riferimento agli aspetti di modellazione statica.
Produciamo un diagramma delle classi.
La compagnia di assicurazioni stipula diversi tipi di
polizze (RC auto, vita, rischi diversi).
La compagnia ha diversi clienti,
ciascuno dei quali può sottoscrivere più contratti.
```

While, the UML Class Diagram used for the analysis is the following one:

Finally, Table 3.10 reports the scores obtained by the AI-generated diagram for Exercise 1. The results indicate a high-quality overall product, with some minor omissions penalized accordingly.

This example was labeled as "Proficient" thanks to its total point of 92 over 100 and exhibited strength in all areas except for a minor structural omission, possibly an implicit class or intermediary relationship.

The general distribution of the scores for the 32 exercises is shown in Figure 3.7 and in Figure 3.11.

The histogram reveals the presence of a few lower-scoring diagrams, which were typically penalized for either missing core entities or misinterpreting relationship types and directions. The prevalence of exercises falls under the higher scores. This emphasizes the tendency of the AI-tool to properly create UML diagrams, given an input exercise.

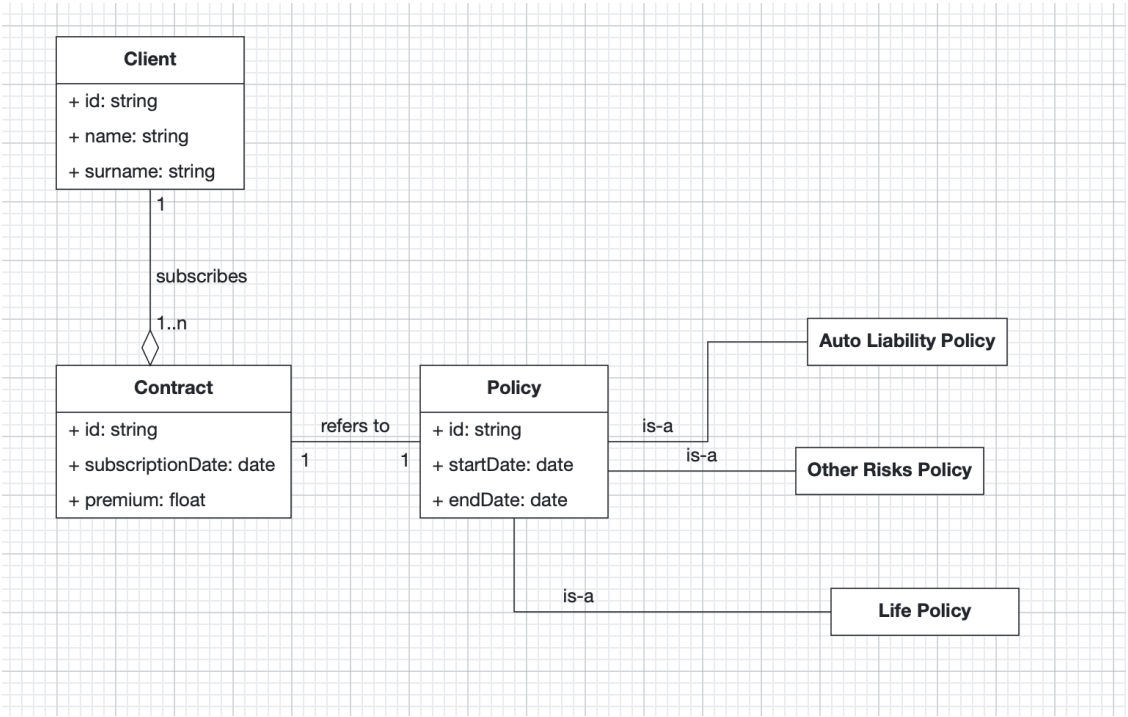

Figure 3.6. LLM generated Diagram of Exercise 1

Table 3.10. Rubric-based Evaluation for Exercise 1

Evaluation Criterion	**Score**	**Max Points**
Completeness of Classes	17	20
Completeness of Attributes	20	20
Completeness of Relationships	17	20
Semantic Correctness of Relationships	20	20
Clarity and Consistency of Naming	10	10
Overall Consistency	10	10
Penalties for Major Omissions or Ambiguities	-2	-10
Total Correctness	**92**	**100**

In addition to the first analysis, a bar chart, shown below in Figure 3.8 was made: it plots the average performance across the six main evaluation dimensions and it allows us to note that the best performance of the tool was regarding the "Completeness of Attributes". Note that the criteria "Clarity and Consistency of Naming" and "Overall Consistency" had a maximum score of 10 points, while all others had a maximum of 20.

Furthermore, it appears that:

Completeness of relationships and Classes showed slightly more variation with respect to Completeness of Attributes and a moderately lower mean, likely reflecting the model's occasional tendency to omit or oversimplify structural connections. Naming and overall

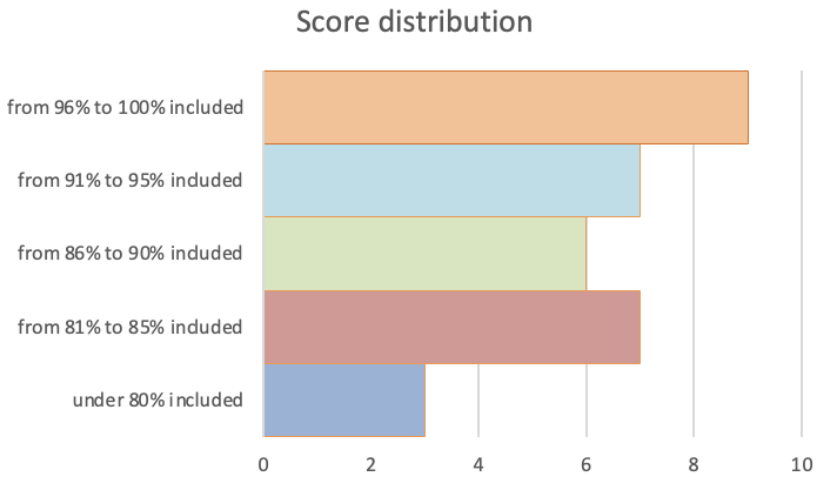

Figure 3.7. Score Distribution of Total Correctness of Exercises

Table 3.11. Performance Evaluation per Score Range

Performance Category	**Score**	**Total Exercise**
Under Expectation Performance	under 80% included	3
Minimum Standard Performance	from 81% to 85% included	7
Acceptable Performance	from 86% to 90% included	6
Proficient Performance	from 91% to 95% included	7
Above Expectation Performance	from 96% to 100% included	9

consistency, while lower in absolute value, remain proportionally strong when normalized by their respective weights. This suggests a general clarity in language use and structural cohesion.

Moreover, penalty reductions are typically associated with major omissions, such as missing core entities, or structural ambiguities, such as contradictory multiplicities or incorrect generalizations. Errors in applying the proper multiplicity were the most frequent. Only a subset of the exercises was affected, reinforcing the observation that the model generally performs reliably under the adopted prompting approach.

Both graphs confirm that the generative model consistently meets fundamental structural expectations. Its most common weaknesses relate to omissions in relationship modeling or oversimplified class structures. The rubric appears balanced in its ability to reward strong diagrams while penalizing serious conceptual flaws. Finally, the rubric ensured that the comparison was not biased by stylistic differences, focusing instead on structural adequacy. It proved especially effective in mitigating linguistic mismatches (e.g., between Italian and English class names), as the evaluation emphasized presence and correctness over surface form.

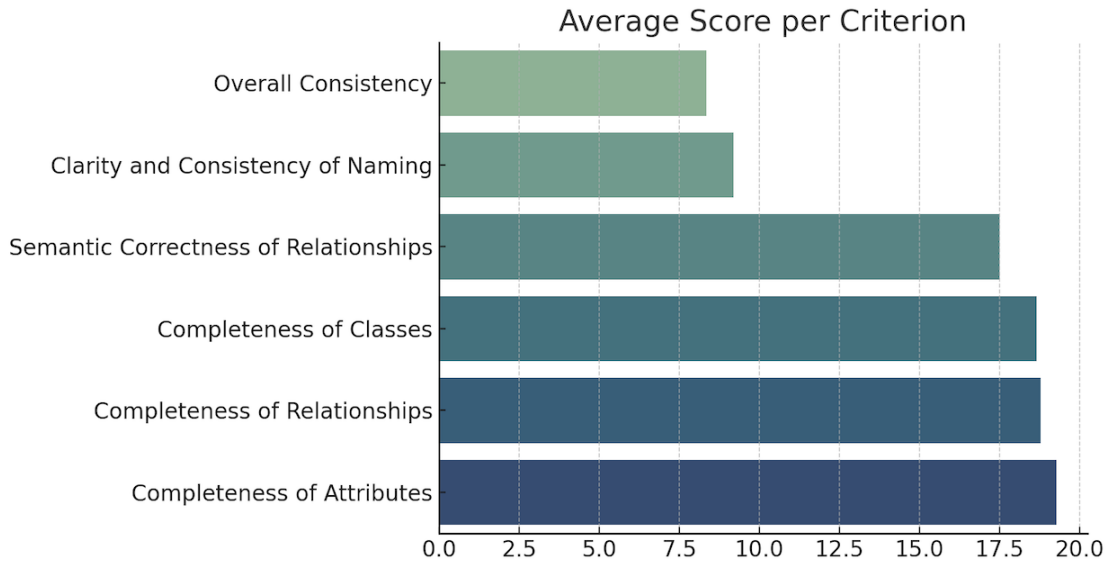

Figure 3.8. Average Score per Criterion Bar Chart

3.4.1 Human Error

The rubric-based evaluation demonstrated a high degree of reliability and interpretability in assessing UML diagram correctness. It provided a replicable framework that could be extended in future research to include human-AI comparisons, prompt refinement analysis, or integration with automated validators. Most importantly, it enabled a level of semantic scrutiny that purely syntactic approaches often fail to capture.

However, it is important to emphasize that the analysis just presented arises from the inability to employ fully automated and systematic tools to verify and evaluate the correctness of the diagrams due to the critic limits presented before. Consequently, although the rubric adopts a numerical scoring system, a certain degree of subjectivity is inevitably introduced in the evaluation process. To clarify, since each rubric category defines a score range rather than a fixed value, the actual score assigned within that range ultimately depends on personal judgment. Therefore, the analysis can be regarded, in essence, as qualitative. Moreover, human evaluation is inherently susceptible to various cognitive and contextual biases that may affect the consistency and reliability of the assessment. For instance, evaluators may unconsciously anchor their judgment to the complexity or visual appeal of a diagram, rather than strictly adhering to the predefined criteria. Confirmation bias may also arise, especially when assessors expect a certain quality level based on previous performance or prior expectations. Additionally, fatigue and time pressure can lead to variability in scoring, particularly in large-scale assessments. These factors highlight the challenges of maintaining objectivity in manual evaluation and reinforce the need for more standardized and replicable approaches to UML diagram assessment.

3.5 Rubric-based Evaluation of Diagrams Similitude

In order to quantitatively assess the degree of similarity of LLM based class diagrams and their human-made counterparts, a structured evaluation rubric was implemented and applied to the set of exercises used in the previous analysis. The comparison focused on analyzing each pair of diagrams, one created manually by a domain expert and the other generated by a Large Language Model thanks to the last version of the prompt presented, along with with a set of different categories that covers all the possible dimensions of UML modeling language.

The analysis was conducted through an excel sheet in which rows were filled by each exercise and the columns presented each dimenion evaluated. Specifically, the table records the total number of classes, attributes, and relationships found in each diagram, along with any observed syntactic or semantic errors. In particular, syntactic errors include, for instance, incomplete or malformed elements such as classes with no attributes, missing multiplicities, or inconsistently named associations, while semantic errors involve more subtle inconsistencies such as misrepresenting a relationship as an attribute, omitting generalizations, or introducing redundant or irrelevant elements not implied by the original specification.

The rubric considers five components:

1. Completeness of Classes: calculated as the ratio of correctly matched classes over the total number of classes in the reference diagram, with a 50% weight over the final result.

2. Completeness of Attributes: reflects the proportion of expected attributes captured in the generated diagram, with a 30% weight over the final result.

3. Completeness of Relationships: measures how many of the reference associations, aggregations, or generalizations are successfully represented, with a 20% weight over the final result.

4. Syntactic Error Penalty: negative adjustment based on the number of syntax-level modeling mistakes, with a maximum of -20%, found multiplying each error per 0,01.

5. Semantic Error Penalty: more severe penalty reflecting conceptual modeling issues, with a maximum of -50%, found multiplying each error per 0,1.

The decision to assign different weights to the various dimensions of the similarity rubric, prioritizing the correctness of class identification over that of attributes and relationships, was directly influenced by the design of the prompt itself. Specifically, the prompt imposed strict modeling constraints on attributes and associations, such as requiring a minimum of three attributes per class and the inclusion of at least one composition or aggregation when logically implied. As a result, these dimensions were more guided and less informative to assess the autonomous reasoning capabilities of the model. Therefore, their weight in the final similarity score was intentionally reduced, whereas class-level modeling, which remained less constrained, was considered more significant and received greater emphasis in the evaluation.

#	Cls	Attr	Rel	Syn.Err	Sem.Err	Score (%)	Diff
1	6/7	10/8	5/7	0	1	84.6	High

Table 3.12. Example of a UML diagram similarity evaluation row based on real data

Here presented one row example of the table:

In addition, it is important to note that association classes were excluded from the similarity computation. This decision was motivated by the fact that their modeling is often implicit and highly variable, making them difficult to evaluate consistently. Similarly, methods were not taken into account, as the prompt explicitly instructed the model to omit them. The scoring framework was thus aligned with the modeling scope defined by the prompt to ensure coherence between generation and evaluation.

Furthermore, in cases where the LLM-generated diagrams exceeded the size of the corresponding human-made reference solutions, whether in terms of number of classes, attributes, or relationships, a normalization constraint was applied to ensure mathematical coherence within the evaluation framework. Specifically, the count of generated elements was set equal to the total number of elements in the reference diagram. This decision was necessary to preserve the internal consistency of the formulas, which were based on ratios and therefore could not exceed 1. Any such surplus was not ignored, but rather treated as a semantic modeling error. Each excess element (a class, an attribute, or a relationship) was interpreted as a divergence from the intended conceptual model. The total number of such mismatches was recorded in the semantic error penalty defined in the rubric. In this way, the evaluation procedure remained aligned with the principle that correct modeling is not only about inclusion, but also about relevance and adherence to specification.

The total number of such events is reported in the table below.

Notably, they occurred frequently and manifested in several different forms.

Table 3.13. Overgeneration of Elements in LLM-based Diagrams (Pre-Refinement)

Problem Explanation	**Number of Exercise**	**% over TOT Exercise**
LLM-based diagram presents more attributes than the human-made diagram	28	88%
LLM-based diagram presents more classes than the human-made diagram	8	25%
LLM-based diagram presents more relationships than the human-made diagram	6	19%

The similarity scores were grouped into four categories to support a qualitative interpretation of the results and highlight differences in model performance. The thresholds were defined to reflect meaningful changes in the structural and semantic alignment between the LLM-generated diagrams and the human-made references. In particular, the

lower bound of 59% was selected as the upper limit of the Low Similarity range, as values below this threshold typically indicate the presence of major structural mismatches, missing or misclassified entities, or completely incoherent diagram topology. The distinction between 59% and 60% is therefore not arbitrary, but reflects a turning point between fundamentally flawed diagrams and those that begin to show partial correctness.

- Low Similarity: score between 0% and 59% included which reflects critical differences and incoherent structures.
- Moderate Similarity: score between 60% and 74% included which remarks that some elements are correct, but others are not present.
- High Similarity: score between 75% and 89% included which remarks similar diagrams with no critical differences.
- Very High Similarity: score between 90% and 100% included which remarks that diagrams are almost identical, wit minor difference.

Results reveal a heterogeneous distribution of similarity levels, as can be seen in the table below:

Table 3.14. Similarity Categories in LLM-Generated Diagrams (Baseline Prompt)

Category	**Interval (included)**	**Results**
Low Similarity	0–59%	22
Moderate Similarity	60–74%	4
High Similarity	75–89%	5
Very High Similarity	90–100%	0

The distribution of similarity scores across the four defined categories reveals a clear divergence of the LLM-generated diagrams compared to the human-made reference ones. As shown in the table, 22 out of 32 exercises (68.75%) fall within the Low Similarity range (0–59%), indicating a substantial deviation from the expected structure. Only 4 exercises fall into the Moderate Similarity band (60–74%), while 5 exercises reach the High Similarity range (75–89%). In particular, no diagram achieved a score in the Very High Similarity range (90–100%), underscoring the consistent difficulty encountered by the model in generating near-structural matches.

Consequently, common differences have been analyzed: one of the recurring divergence patterns involved the modeling strategy, that is, the LLM would often represent a domain concept as a class instead of an attribute or the reverse. **While such modeling choices are not incorrect, they do affect structural similarity and can lead to penalization under the rubric**. Indeed, **this raises the broader question of how to differentiate between genuine modeling errors and plausible modeling variants, an issue that becomes particularly relevant when multiple representations are logically valid**.

Beyond individual cases, the general analysis supports several general findings:

- No syntactic errors were identified in any of the 32 evaluated exercises. This outcome demonstrates the LLM's consistent ability to respect the fundamental structural rules of UML class diagrams. According to the adopted evaluation framework, syntactic correctness covers a range of conditions, including the presence of valid and unique class and attribute names, the specification of appropriate types for all attributes, and the use of valid multiplicities in associations. LLM is particularly effective at adhering to the formal syntax of the UML notation when guided by a well-structured prompt. From this perspective, the syntactic reliability of the generated diagrams can be considered high and stable.

- Semantic errors were frequent and impactful, accounting for most of the penalty weight in lower scoring diagrams. In most cases, these errors were not due to subtle semantic mismatches, but rather to over-generation, that is, the presence of additional classes or attributes not included in the reference diagram. Only in a smaller subset of exercises were the semantic penalties associated with more specific mismatches, such as attributes with incorrect types, missing associations between classes that were expected to be connected, or discrepancies in the naming, multiplicity, or type of the associations themselves. In several cases, missing associations were simply a consequence of missing classes that should have been linked.

- Generalizations included in human-made reference diagrams were often replaced by a string attribute named "type" in the superclass. This behavior can be explained by the absence of explicit instructions for modeling enumerations in the prompt, leading the LLM to simulate subtype distinctions through attributes rather than inheritance structures.

Although the rubric-based approach provides a practical and repeatable method to assess similarity, it does come with certain limitations. The scoring process, despite being numerically grounded, still requires human judgment to identify what constitutes a matching class or relationship, introducing a degree of subjectivity. Moreover, the weighting scheme, although empirically reasonable, reflects implicit assumptions about the relative importance of different modeling dimensions.

3.6 Final Refinement of the strategy

The previous analysis highlights a key limitation in the model's generative capabilities under the initial prompt: although some outputs partially reflect the intended structure, the vast majority lack sufficient alignment in classes, relationships, and attributes to be considered similar diagrams with respect to the reference ones. Moreover, a recurrent issue was the inclusion of redundant or unnecessary attributes, often added solely to fulfill the prompt's minimum count requirement. These superfluous attributes not only affected the semantic clarity of the diagrams, but also introduced inefficiencies by increasing the memory footprint of the JSON representation without contributing meaningful information. This observation reinforces the need for refining the prompting strategies

to produce more concise and computationally efficient UML models. Hence, the next phase was related to the study of changes to be adopted, which ended with the choice of two modifications. The first attempt dealt exclusively with the prompt. Since, as can be seen from the results of similarity analysis, the number of some elements of the LLM-based diagrams often exceeded the number of the reference ones, it was necessary to adopt a prompt modification technique that allowed more attachment to the model, thus omitting the various impositions of minimum number of attributes or aggregations and compositions, although these were not evaluated in either the correct or similarity analysis. Crucially, the new prompt removed the fixed rule on the number of attributes per class and instead introduced a qualitative guideline: each main class should contain at least two realistic attributes, as implied by the domain. This change allowed the models to avoid redundant or invented data, leading to more realistic outputs. For example, in the same Library case, new prompt-based outputs tended to include only attributes such as "+ name: string" and "+ address: string", omitting speculative or extraneous fields. Beyond attribute handling, the new prompt also encouraged a deeper understanding of domain logic by requiring explicit representation of inheritance, composition, or role-based relationships when inferred from the text. It formalized the distinction between different types of associations, such as Class Bidirectional, Class Aggregation and Class Composition and specified how generalizations should be structured, ensuring that the specialized class appears as the source and the generalized one as the target, with empty multiplicities. Here below the new updated prompt.

```
You are a software engineering professor who is designing a UML class
    diagram modeling exercise for your students. Starting from a textual
    description of a system, you must create a reference solution in JSON
    format that is fully compatible with the Apollon UML tool (https://
    apollon.ase.in.tum.de).


Follow these strict modeling guidelines:


1. The output JSON must always start with the following exact structure:

{
  "version": "3.0.0",
  "type": "ClassDiagram",
  "size": { "width": 1600, "height": 780 },
  "interactive": {
    "elements": {},
    "relationships": {}
  },
  "assessments": {},

This header must be present exactly as shown above.

Important: After this header, you must continue the JSON with a complete
    declaration of:
- "elements": { ... }
- "relationships": { ... }
```

```
These must be written outside and separately from the "interactive" block.

Do not insert any class, attribute, or relationship inside "interactive".
    That block must remain empty as required by Apollon format for
    compatibility.

2. Class formatting (elements dictionary):
   - Each class must be represented as an object with a UUIDv4 key and
       include:
   - id, name, "type": "Class", owner, bounds, attributes, and a methods
       list (which may be empty if no methods are required)

3. Each attribute and method must:
   - Appear immediately after its owning class as an independent object
       within the "elements" block!!!
   - Have: unique id, name = "+ attributeName: Type", "type": either "
       ClassAttribute" or "ClassMethod" , owner = ID of the class to which
        it belongs, bounds.
   - Only simple types are allowed: string, int, float, date
   - A class cannot have another class as an attribute
   - No collections: avoid lists, sets, maps
   - Use associations instead of composite types

4. Relationship formatting (relationships dictionary):
   - Each relationship must include: id (UUIDv4 format), type, name,
       bounds, path, source, target, and isManuallyLayouted = false
   - Source and target must specify:
     - element, direction (use "Left", "Right", "Up", or "Down" based on
         the relative positions), multiplicity
     - Optional role for disambiguation

5. General rules:
   - If the text suggests specializations (e.g., types of users or
       entities), model them using generalization:
 use: "type": "ClassBidirectional" , "name": "is-a".
The specialized class must be the "source", the generalized one the "
    target"
Leave multiplicities empty on both sides

Each class MUST BE CONNECTED with a relationship to at least one other
    class.

- If a general class has subtypes with distinct behaviors, model them
    using UML generalization.
- If the text includes exceptions or differentiated cases (e.g., all
    except X , only s o m e ), explicitly model those differences
    using subclasses, roles, or structural distinctions.
- Do not encode semantic rules as implicit assumptions or comments: they
    must be clearly represented in the m o d e l s structure (e.g., through
    dedicated classes, specializations, or constraints).


   - If the text implies ownership or whole-part structure model it as "
       type": "ClassAggregation" or "type": "ClassComposition"
     - Set "multiplicity": "1" on the composite (whole) side
```

```
In aggregations, the container class must be the one with the diamond
    symbol (i.e., the source if appropriate).
   - For ordinary associations (peer-to-peer), use "ClassBidirectional"

   - If multiple associations exist between the same pair of classes,
       label them clearly with name and use role in source/target when
       necessary
   - When a single class plays multiple roles, model each relationship
       separately and specify their roles
   - Always include intermediate classes if they improve semantic clarity
   - If a relationship implies its own attributes (e.g., date, quantity),
       model it as an associative class with its corresponding attributes.
   - Always assign realistic multiplicities: use 1, 0..1, 0..n, or 1..n as
        implied by the domain

If the description is vague or incomplete, assume a complete pedagogical
    model
 Include abstract/general classes, useful associations, and appropriate
     multiplicities to enrich the diagram

 Use meaningful, consistent, and grammatically correct names for classes,
     attributes, and relationships as they are specified in the text
 Avoid generic or ambiguous labels like "thing", "info", or "writes"
 For clarity, use phrases like "is supervised by" instead of active voice
     ("supervises")

Each attribute and method must appear immediately after its owning class
    as an independent object within the "elements" block!!!

10. Minimum Modeling Completeness:
In every exercise, treat the goal as pedagogical. Even if the textual
    description is minimal, you must model a semantically rich and
    educationally useful class diagram. Always include:

At least one generalization if subtypes are implied in the text (e.g.,
    types of policies)
At least one composition or aggregation if whole-part relationships are
    implied
At least 2 realistic attributes for each main class


Return only the complete JSON object. Do not include any explanation,
    comment, or markdown formatting.
All required fields in the example files must be present. In particular:
- "owner" in attributes/methods
- correct "type"
- "bounds" in all elements
- "path" in all relationships

The output must be a valid, complete, and working JSON file for Apollon,
    with no comments or additional text.

here attached you found the textual description of the system and the file
     json as an example of what I expect.
```

```
The text of the next exercise to do is:
```

After the just described modification made to the prompt, it was necessary to run both analyses, correctness and similarity, again to check for any respective changes in the results. The analysis was conducted with the same approach used with the results obtained from the use of the first prompt, through thus a rubric that took into consideration both semantic and syntactic correctness and the ability of the model to correctly represent classes, attributes and relationships declared in the text.

3.6.1 Correctness Analysis after First and Second Refinement of the Prompt

As can be seen in Table 3.15, the results obtained are satisfactory, but more importantly, they have improved over the previous results.

Table 3.15. Updated Performance Evaluation per Score Range

Performance Category	**Updated Score**	**Total Exercise**
Under Expectation Performance	under 80% included	1
Minimum Standard Performance	from 81% to 85% included	3
Acceptable Performance	from 86% to 90% included	6
Proficient Performance	from 91% to 95% included	10
Above Expectation Performance	from 96% to 100% included	12

Comparative analysis of performance distributions before and after the adoption of the revised prompt reveals a substantial improvement in the quality of the UML diagrams generated by the model.

Table 3.16. Performance comparison before and after prompt refinement

Performance Category	**Before (n)**	**After (n)**	**Abs. Change**	**% Change**
Under Expectation (<80%)	3	1	–2	–66.7%
Minimum Standard (81–85%)	7	3	–4	–57.1%
Acceptable (86–90%)	6	6	0	0.0%
Proficient (91–95%)	7	10	+3	+42.9%
Above Expectation (96–100%)	9	12	+3	+33.3%
Total	32	32		

It is noticeable from the Table 3.16 that:

- the number of exercises classified as Under Expectation Performance dropped from 3 to 1, indicating a 66% reduction in critical failures.

- Similarly, exercises that fell below the minimum standard threshold (from 81% to 85% accuracy) decreased from 10 to just 4, reflecting a 60% reduction in outputs that did not meet acceptable quality levels. This shift is significant, as it suggests that the updated prompt not only increased the average accuracy but also reduced the occurrence of unstable or low-quality generations.

- The proportion of high-quality outputs, such as those rated as Proficient or Above Expectation, increased from 16 to 22 exercises, which corresponds to a gain of 18.8% points (from 50% to 68.8%). This upward shift in performance concentration indicates greater robustness and consistency in the generative process. In particular, the number of Proficient exercises alone increased by 43%, while the top-tier Above Expectation category rose from 9 to 12 instances. Importantly, this improvement was achieved not through overfitting or redundant modeling, but through enhanced semantic alignment and structural adherence to UML conventions, as enforced by the revised prompt.

In conclusion, the updated prompt strategy not only reduced the generation of inadequate models but also significantly raised the proportion of outputs that reached or exceeded expected standards. This demonstrates its effectiveness in guiding the model toward producing diagrams that are both valid and structurally correct.

Following the first changes to the prompt, the second attempt involved the use of not one but two training exercises. Specifically, the first remained unchanged and a second exercise was added, which has following text 3.6.1 and json solutions.

Esercizio Training 2

Prestiti bancari
Sviluppare un'applicazione orientata agli oggetti per gestire i prestiti che una banca concede ai propri clienti.
La banca è caratterizzata da un nome e da un insieme di clienti. I clienti sono caratterizzati da nome, cognome, codice fiscale, stipendio.
Il prestito concesso al cliente, considerato intestatario del prestito, è caratterizzato da un ammontare, una rata, una data inizio, una data fine.
Per i clienti e per i prestiti si vuole stampare un prospetto riassuntivo con tutti i dati che li caratterizzano in un formato di tipo stringa a piacere.
Per la banca deve essere possibile aggiungere, modificare, eliminare e ricercare un cliente.
Inoltre, la banca deve poter aggiungere un prestito.
La banca deve poter eseguire delle ricerche sui prestiti concessi ad un cliente dato il codice fiscale.
La banca vuole anche sapere, dato il codice fiscale di un cliente, l'ammontare totale dei prestiti concessi.

Table 3.17. Italian Text of Exercise 2

The json file could be exported once the reference graph (Figure) was reproduced on UML Modeler.

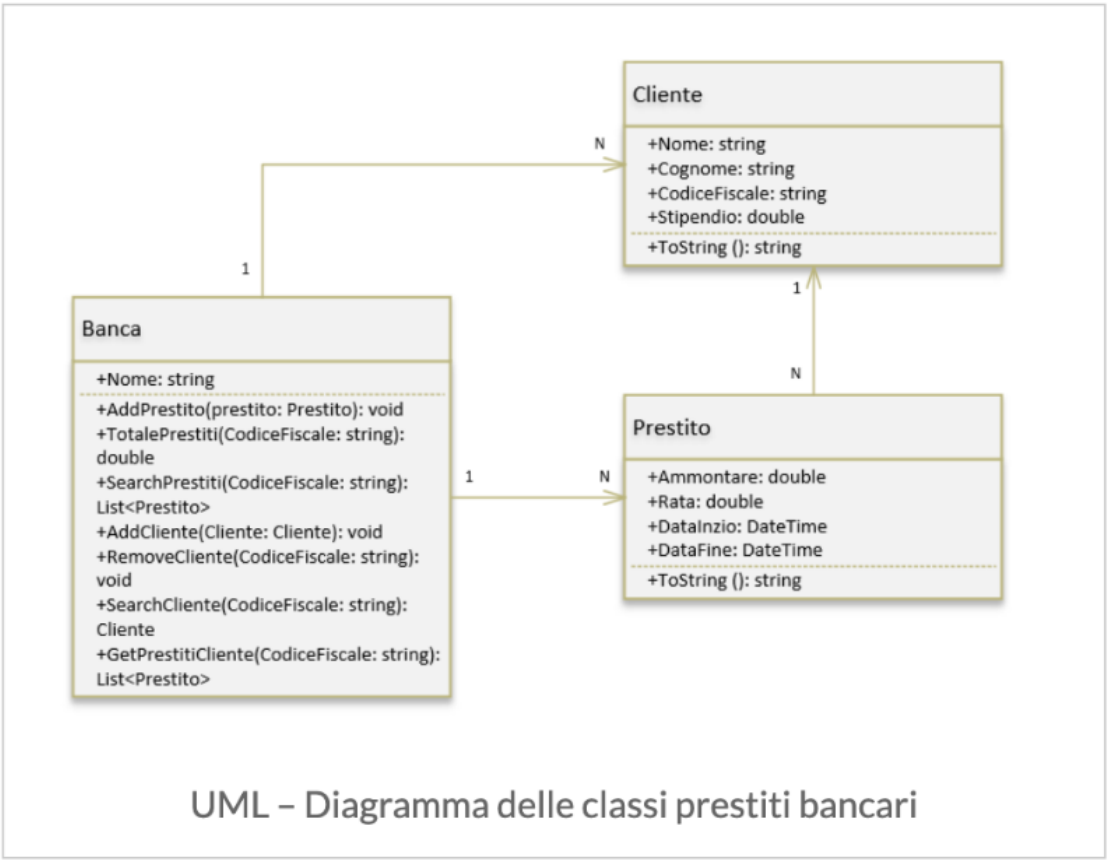

Figure 3.9. UML Diagram of Training Exercise 2

```
\textbf{Json Esercizio Training 2}
    \\[0.5em]
{
    "version": "3.0.0",
    "type": "ClassDiagram",
    "size": {
        "width": 1360,
        "height": 740
    },
    "interactive": {
        "elements": {},
        "relationships": {}
    },
    "elements": {
        "79cd04af-371c-463c-a0c7-4
            b1c2158e8d6": {
            "id": "79cd04af-371c-463
                c-a0c7-4b1c2158e8d6
                ",
            "name": "Banca",
            "type": "Class",
            "owner": null,
            "bounds": {
                "x": -660,
                "y": -210,
                "width": 410,
                "height": 280
            },
            "attributes": [
                "645d472e-06c3-47f8
                    -8647-3
                    c41c80d41c1"
            ],
            "methods": [
                "3bd89617-cb6a-4c80-
                    a0f7-
                    cd1aa22f851c",
                "f9e70230-d514-4c92
                    -8d26-
                    b04801ba4479",
```

```
            "14f67049-9bef-485a-
                bcb6-
                b33fe62d843f",
            "be7a5e8a-eead-4a71
                -9bcb-9648213
                af1f5",
            "95d49f3d-df72-4c5f
                -9a2e-097
                a19113c4f",
            "ff11c279-4c40-4bb5-
                bcc5-7519
                bd8af6b5",
            "d0fb0ae6-2772-4f40-
                ab86-7480
                da79338b"
        ]
    },
    "645d472e-06c3-47f8-8647-3
        c41c80d41c1": {
        "id": "645d472e-06c3-47
            f8-8647-3c41c80d41c1
            ",
        "name": "+ Nome: String
            ",
        "type": "ClassAttribute
            ",
        "owner": "79cd04af-371c
            -463c-a0c7-4
            b1c2158e8d6",
        "bounds": {
            "x": -659.5,
            "y": -169.5,
            "width": 409,
            "height": 30
        }
    },
    "3bd89617-cb6a-4c80-a0f7-
        cd1aa22f851c": {
        "id": "3bd89617-cb6a-4
            c80-a0f7-
            cd1aa22f851c",
        "name": "+ AddPrestito(
            prestito:Prestito):
            void)",
        "type": "ClassMethod",
        "owner": "79cd04af-371c
            -463c-a0c7-4
            b1c2158e8d6",
        "bounds": {
            "x": -659.5,
            "y": -139.5,
            "width": 409,
            "height": 30
        }
    },
    "f9e70230-d514-4c92-8d26-
        b04801ba4479": {
        "id": "f9e70230-d514-4
            c92-8d26-
            b04801ba4479",
        "name": "+
            TotalePrestiti(
            CodiceFiscale:
            string):double",
        "type": "ClassMethod",
        "owner": "79cd04af-371c
            -463c-a0c7-4
            b1c2158e8d6",
        "bounds": {
            "x": -659.5,
            "y": -109.5,
            "width": 409,
            "height": 30
        }
    },
    "14f67049-9bef-485a-bcb6-
        b33fe62d843f": {
        "id": "14f67049-9bef-485
            a-bcb6-b33fe62d843f
            ",
        "name": "+SearchPrestiti
            (CodiceFiscale:
            string):List<
            Prestito>",
        "type": "ClassMethod",
        "owner": "79cd04af-371c
            -463c-a0c7-4
            b1c2158e8d6",
        "bounds": {
            "x": -659.5,
            "y": -79.5,
            "width": 409,
            "height": 30
        }
    },
    "be7a5e8a-eead-4a71-9bcb
        -9648213af1f5": {
        "id": "be7a5e8a-eead-4
            a71-9bcb-9648213
            af1f5",
        "name": "+AddCliente(
            Cliente:Cliente):
            void",
        "type": "ClassMethod",
        "owner": "79cd04af-371c
            -463c-a0c7-4
            b1c2158e8d6",
        "bounds": {
            "x": -659.5,
            "y": -49.5,
```

```
            "width": 409,
            "height": 30
        }
    },
    "95d49f3d-df72-4c5f-9a2e-097
        a19113c4f": {
        "id": "95d49f3d-df72-4
            c5f-9a2e-097
            a19113c4f",
        "name": "+RemoveCliente(
            CodiceFiscale:string
            ):void",
        "type": "ClassMethod",
        "owner": "79cd04af-371c
            -463c-a0c7-4
            b1c2158e8d6",
        "bounds": {
            "x": -659.5,
            "y": -19.5,
            "width": 409,
            "height": 30
        }
    },
    "ff11c279-4c40-4bb5-bcc5
        -7519bd8af6b5": {
        "id": "ff11c279-4c40-4
            bb5-bcc5-7519
            bd8af6b5",
        "name": "+SearchCliente(
            CodiceFiscale:string
            ):Cliente",
        "type": "ClassMethod",
        "owner": "79cd04af-371c
            -463c-a0c7-4
            b1c2158e8d6",
        "bounds": {
            "x": -659.5,
            "y": 10.5,
            "width": 409,
            "height": 30
        }
    },
    "d0fb0ae6-2772-4f40-ab86
        -7480da79338b": {
        "id": "d0fb0ae6-2772-4
            f40-ab86-7480
            da79338b",
        "name": "+
            GetPrestitiCliente(
            CodiceFiscale:string
            ):List<Prestito>",
        "type": "ClassMethod",
        "owner": "79cd04af-371c
            -463c-a0c7-4
            b1c2158e8d6",
        "bounds": {
            "x": -659.5,
            "y": 40.5,
            "width": 409,
            "height": 30
        }
    },
    "1e7637f9-f2f4-4a75-a994-
        a15537dbe815": {
        "id": "1e7637f9-f2f4-4
            a75-a994-
            a15537dbe815",
        "name": "Cliente",
        "type": "Class",
        "owner": null,
        "bounds": {
            "x": -150,
            "y": -350,
            "width": 180,
            "height": 190
        },
        "attributes": [
            "7af59ce3-4ab4-451e
                -8562-
                e3a7b637343a",
            "f5de780c-0adc-4c76-
                b0cb-430
                a1c41ca99",
            "b420a1c1-eb5d-412b
                -8ebe-04
                fef7151c02",
            "392f66b3-a15b-4c27
                -8160-70
                e8acb65c45"
        ],
        "methods": [
            "d896f6cd-4bd2-4a78-
                b397-
                a59f7e315b22"
        ]
    },
    "7af59ce3-4ab4-451e-8562-
        e3a7b637343a": {
        "id": "7af59ce3-4ab4-451
            e-8562-e3a7b637343a
            ",
        "name": "+ Nome: String
            ",
        "type": "ClassAttribute
            ",
        "owner": "1e7637f9-f2f4
            -4a75-a994-
            a15537dbe815",
        "bounds": {
            "x": -149.5,
```

```
            "y": -309.5,
            "width": 179,
            "height": 30
        }
    },
    "f5de780c-0adc-4c76-b0cb-430
        a1c41ca99": {
        "id": "f5de780c-0adc-4
            c76-b0cb-430
            a1c41ca99",
        "name": "+Cognome:String
            ",
        "type": "ClassAttribute
            ",
        "owner": "1e7637f9-f2f4
            -4a75-a994-
            a15537dbe815",
        "bounds": {
            "x": -149.5,
            "y": -279.5,
            "width": 179,
            "height": 30
        }
    },
    "b420a1c1-eb5d-412b-8ebe-04
        fef7151c02": {
        "id": "b420a1c1-eb5d-412
            b-8ebe-04fef7151c02
            ",
        "name": "+CodiceFiscale:
            String",
        "type": "ClassAttribute
            ",
        "owner": "1e7637f9-f2f4
            -4a75-a994-
            a15537dbe815",
        "bounds": {
            "x": -149.5,
            "y": -249.5,
            "width": 179,
            "height": 30
        }
    },
    "392f66b3-a15b-4c27-8160-70
        e8acb65c45": {
        "id": "392f66b3-a15b-4
            c27-8160-70
            e8acb65c45",
        "name": "+Stipendio:
            double",
        "type": "ClassAttribute
            ",
        "owner": "1e7637f9-f2f4
            -4a75-a994-
            a15537dbe815",
        "bounds": {
            "x": -149.5,
            "y": -219.5,
            "width": 179,
            "height": 30
        }
    },
    "d896f6cd-4bd2-4a78-b397-
        a59f7e315b22": {
        "id": "d896f6cd-4bd2-4
            a78-b397-
            a59f7e315b22",
        "name": "+ toString():
            String",
        "type": "ClassMethod",
        "owner": "1e7637f9-f2f4
            -4a75-a994-
            a15537dbe815",
        "bounds": {
            "x": -149.5,
            "y": -189.5,
            "width": 179,
            "height": 30
        }
    },
    "46c1a08f-ac9a-4519-8ad0-
        adc1094ddb8f": {
        "id": "46c1a08f-ac9a
            -4519-8ad0-
            adc1094ddb8f",
        "name": "Prestito",
        "type": "Class",
        "owner": null,
        "bounds": {
            "x": -160,
            "y": 60,
            "width": 180,
            "height": 190
        },
        "attributes": [
            "6943aa30-2fb7-4a7d-
                aaf5-17753
                fee4491",
            "f965a80f-1497-4c17-
                bddd-2621
                d0f5e54d",
            "af2c7958-7362-4dde-
                b245-19
                e3efa75aee",
            "03cbf26b-02a0-4d45
                -876a-
                d7fc7ebb2433"
        ],
        "methods": [
```

```
            "44aaa0ef-df2e-42af-
                b7d6-38
                a0af15caa0"
        ]
    },
    "6943aa30-2fb7-4a7d-aaf5
        -17753fee4491": {
        "id": "6943aa30-2fb7-4
            a7d-aaf5-17753
            fee4491",
        "name": "+ Ammontare:
            Double",
        "type": "ClassAttribute
            ",
        "owner": "46c1a08f-ac9a
            -4519-8ad0-
            adc1094ddb8f",
        "bounds": {
            "x": -159.5,
            "y": 100.5,
            "width": 179,
            "height": 30
        }
    },
    "f965a80f-1497-4c17-bddd
        -2621d0f5e54d": {
        "id": "f965a80f-1497-4
            c17-bddd-2621
            d0f5e54d",
        "name": "+ Rata: double
            ",
        "type": "ClassAttribute
            ",
        "owner": "46c1a08f-ac9a
            -4519-8ad0-
            adc1094ddb8f",
        "bounds": {
            "x": -159.5,
            "y": 130.5,
            "width": 179,
            "height": 30
        }
    },
    "af2c7958-7362-4dde-b245-19
        e3efa75aee": {
        "id": "af2c7958-7362-4
            dde-b245-19
            e3efa75aee",
        "name": "+ DataInizio:
            DateTime",
        "type": "ClassAttribute
            ",
        "owner": "46c1a08f-ac9a
            -4519-8ad0-
            adc1094ddb8f",
            "bounds": {
                "x": -159.5,
                "y": 160.5,
                "width": 179,
                "height": 30
            }
        },
        "03cbf26b-02a0-4d45-876a-
            d7fc7ebb2433": {
            "id": "03cbf26b-02a0-4
                d45-876a-
                d7fc7ebb2433",
            "name": "+ DataFine:
                DateTime",
            "type": "ClassAttribute
                ",
            "owner": "46c1a08f-ac9a
                -4519-8ad0-
                adc1094ddb8f",
            "bounds": {
                "x": -159.5,
                "y": 190.5,
                "width": 179,
                "height": 30
            }
        },
        "44aaa0ef-df2e-42af-b7d6-38
            a0af15caa0": {
            "id": "44aaa0ef-df2e-42
                af-b7d6-38a0af15caa0
                ",
            "name": "+ toString():
                string",
            "type": "ClassMethod",
            "owner": "46c1a08f-ac9a
                -4519-8ad0-
                adc1094ddb8f",
            "bounds": {
                "x": -159.5,
                "y": 220.5,
                "width": 179,
                "height": 30
            }
        }
    },
    "relationships": {
        "94ad86ab-9b70-4eb5-8856-79
            a4ab20eb17": {
            "id": "94ad86ab-9b70-4
                eb5-8856-79
                a4ab20eb17",
            "name": "",
            "type": "
                ClassBidirectional",
            "owner": null,
```

```
            "bounds": {
                "x": -70,
                "y": -160,
                "width":
                    21.552734375,
                "height": 230.125
            },
            "path": [
                {
                    "x": 5,
                    "y": 220
                },
                {
                    "x": 5,
                    "y": 0
                }
            ],
            "source": {
                "direction": "Up",
                "element": "46c1a08f
                    -ac9a-4519-8ad0-
                    adc1094ddb8f",
                "multiplicity": "N",
                "role": ""
            },
            "target": {
                "direction": "Down",
                "element": "1e7637f9
                    -f2f4-4a75-a994-
                    a15537dbe815",
                "multiplicity": "1",
                "role": ""
            },
            "isManuallyLayouted":
                false
        },
        "34dca8ac-f297-49bc-ac31-29
            fc7dd17dbc": {
            "id": "34dca8ac-f297-49
                bc-ac31-29fc7dd17dbc
                ",
            "name": "",
            "type": "
                ClassBidirectional",
            "owner": null,
            "bounds": {
                "x": -460,
                "y": 70,
                "width": 300,
                "height": 124.125
            },
            "path": [
                {
                    "x": 300,
                    "y": 85
                },
                {
                    "x": 5,
                    "y": 85
                },
                {
                    "x": 5,
                    "y": 0
                }
            ],
            "source": {
                "direction": "Left",
                "element": "46c1a08f
                    -ac9a-4519-8ad0-
                    adc1094ddb8f",
                "multiplicity": "N",
                "role": ""
            },
            "target": {
                "direction": "Down",
                "element": "79cd04af
                    -371c-463c-a0c7
                    -4b1c2158e8d6",
                "multiplicity": "1",
                "role": ""
            },
            "isManuallyLayouted":
                false
        },
        "363b62a1-312b-4be2
            -8036-8926656b3017": {
            "id": "363b62a1-312b-4
                be2-8036-8926656
                b3017",
            "name": "",
            "type": "
                ClassBidirectional",
            "owner": null,
            "bounds": {
                "x": -460,
                "y": -265,
                "width": 310,
                "height": 65.125
            },
            "path": [
                {
                    "x": 5,
                    "y": 55
                },
                {
                    "x": 5,
                    "y": 10
                },
                {
                    "x": 310,
```

```
                "y": 10
            }
        ],
        "source": {
            "direction": "Up",
            "element": "79cd04af
                -371c-463c-a0c7
                -4b1c2158e8d6",
            "multiplicity": "1",
            "role": ""
        },
        "target": {
            "direction": "Left",
            "element": "1e7637f9
                -f2f4-4a75-a994-
                a15537dbe815",
            "multiplicity": "N",
            "role": ""
        },
        "isManuallyLayouted":
            false
    }
  },
  "assessments": {}
}
```


Similarly to what was done before, following the decision to use a second training exercise given to the model, it was necessary to perform the evaluations on the entire data set of 32 exercises, using the same evaluation rubric criteria, in order to verify potential improvements with respect to the strategies adopted before. Here below, in Table3.18 the final results of the analysis of the Correctness of the UML Diagrams generated by the LLM model.

Table 3.18. Updated Performance Evaluation – After Final Prompt Refinement

Performance Category	**Updated Score**	**Total Exercise**
Under Expectation Performance	under 80% included	0
Minimum Standard Performance	from 81% to 85% included	4
Acceptable Performance	from 86% to 90% included	6
Proficient Performance	from 91% to 95% included	9
Above Expectation Performance	from 96% to 100% included	13

By comparing these results with the previous obtained from the second refinement of the prompt

some improvements can be noticed:

- the most significant outcome of the second prompt refinement is the complete removal of Under Expectation cases. This confirms an improved baseline robustness, as all diagrams now meet at least the minimum quality threshold of 80%.
- The number of Acceptable (86–90%) outputs remained unchanged at six, indicating that the core modeling capabilities of the LLM remained stable in the mid-quality range.
- A minor rise in the Minimum Standard category (from 3 to 4) suggests that while some outputs improved to the top tier, a few others may have regressed slightly, perhaps due to variation in prompt interpretation or edge-case ambiguity in the exercises.

Table 3.19. Performance comparison after second prompt refinement

Performance Category	**Before (n)**	**After (n)**	**Abs. Change**	**% Change**
Under Expectation (<80%)	1	0	–1	–100.0%
Minimum Standard (81–85%)	3	4	+1	+33.3%
Acceptable (86–90%)	6	6	0	0.0%
Proficient (91–95%)	10	9	–1	–10.0%
Above Expectation (96–100%)	12	13	+1	+8.3%
Total	32	32		

- The number of Proficient outputs decreased marginally (–1), while Above Expectation increased by the same amount (+1), indicating that more diagrams achieving near-perfect alignment with the reference model.
- A main part of 13 exercises over the total 32 (40.6%) reached the top category Above Expectation.

Despite minimal numerical changes, the overall trend illustrated in Figure 3.10 confirms the overall effectiveness of the progressive prompt refinements.

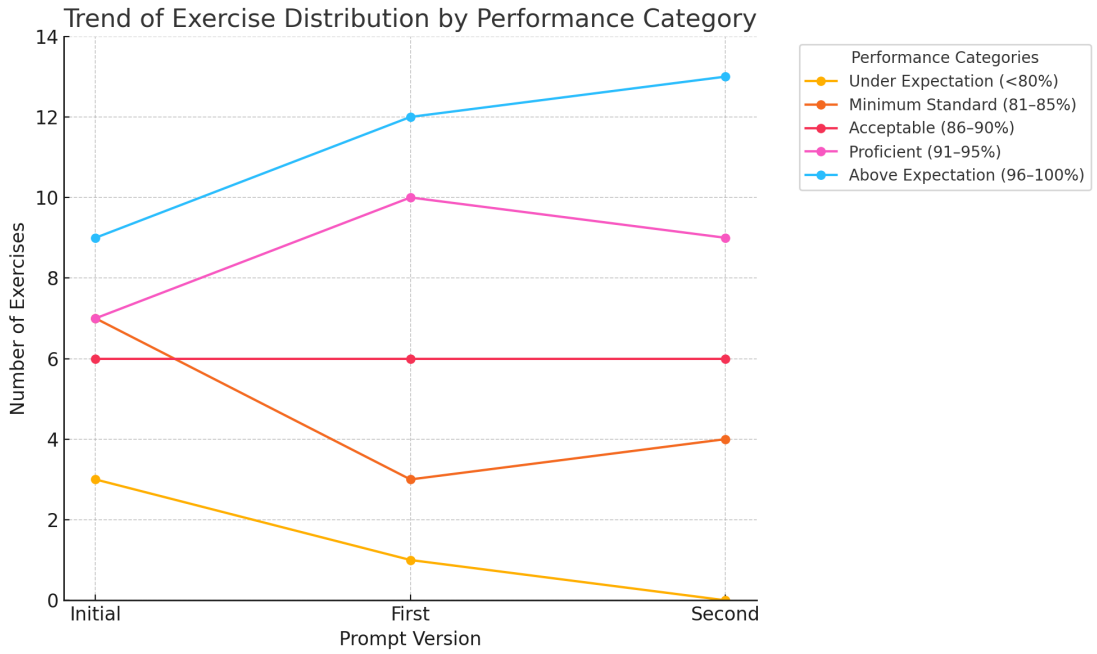

Figure 3.10. Refinement Prompt Iter Trend Results

The most striking result is the complete elimination of Under Expectation cases by the second refinement, which signals a substantial improvement in the general quality of generated UML diagrams. Additionally, there is a visible and consistent upward shift in the number of outputs classified as Above Expectation, growing from 9 with the initial prompt to 12 with the first refinement, and ultimately to 13 with the second refinement.

The Proficient category increased during the first refinement, suggesting that the initial improvements were particularly effective at providing outputs in the 91–95% range. Although there was a slight decrease in this category in the second refinement (from 10 to 9), this drop was absorbed by the growth in the Above Expectation range, indicating that some outputs were further improved rather than regressed.

The number of Acceptable results remained stable throughout the experiment, suggesting that the prompt modifications did not negatively impact mid-level performance. Meanwhile, the slight rise in Minimum Standard cases during the final iteration (from 3 to 4) appears marginal and statistically negligible when compared to the overall gain in higher tiers.

In summary, the trend demonstrates a clear and steady consolidation of performance in the upper bands, confirming that each prompt revision contributed to enhanced clarity, consistency, and modeling accuracy. The results can therefore be interpreted as a strong validation of the prompt engineering strategy adopted in this study.

3.6.2 Mitigating Human Evaluation Bias

Given that, as said in section 3.4.1., there is an intrinsic subjectivity involved in the human evaluation of UML class diagrams, especially when assessments rely on the evaluator's interpretation of quality dimensions, the possibility of human error cannot be overlooked. Although a detailed scoring rubric was adopted, defining each criterion with a maximum of 20 or 10 points, the final attribution of points involves some degree of discretion. This is particularly evident in semantically ambiguous cases or in borderline classifications where a few points can determine a shift in performance category. Factors such as the prior experience of evaluator, familiarity with UML conventions, or interpretation of textual cues can subtly influence judgment and lead to inconsistencies.

To mitigate these risks and better understand the robustness of the evaluation process, a second round of assessments was conducted. A different evaluator, specifically a student in the Management Engineering program, was asked to **independently** evaluate the entire set of UML diagrams. The second evaluator was provided with the same scoring legend and criteria but was explicitly prevented from viewing the original scores. This blind setup was essential to avoid anchoring bias, since prior evaluations could influence the second grader's judgment. By keeping the evaluators independent, the goal was to simulate a realistic peer-assessment setting and to measure the extent to which subjective variance might affect the overall results.

This second round served two primary purposes. First, it allowed for a cross-validation of the original scores by observing how consistently the criteria were applied by different individuals. Second, it enabled the study of inter-rater variability as a potential source of noise in the experimental data. While full convergence between two human evaluations is rarely achievable, comparing the two sets of scores provides a valuable indication of the

stability and reproducibility of the assessment framework.

Fortunately, the results of the correctness assessment conducted by the second evaluator were not too far compared to those obtained by the first evaluator, as shown in Table 3.20.

Table 3.20. Updated Performance Evaluation – Intermediate Prompt Refinement

Performance Category	**Updated Score**	**Total Exercise**
Under Expectation Performance	under 80% included	3
Minimum Standard Performance	from 81% to 85% included	4
Acceptable Performance	from 86% to 90% included	7
Proficient Performance	from 91% to 95% included	5
Above Expectation Performance	from 96% to 100% included	13

However, the goal of getting another independent assessment was related to the fact that we could then calculate the variance between the two sets of results and the average between the two to try to normalize the result as much as possible.

Table 3.21. Comparison between first and second evaluation per performance category

Performance Category	**I Eval.**	**II Eval.**	**Abs. Δ**	**% vs I**
Under Expectation (<80%)	0	3	+3	—
Minimum Standard (81–85%)	4	4	0	0.0%
Acceptable (86–90%)	6	7	+1	+16.7%
Proficient (91–95%)	9	5	–4	–44.4%
Above Expectation (96–100%)	13	13	0	0.0%
Total	32	32		

The comparison between the two independent evaluations shown in Table 3.22 reveals differences in the distribution of scores across performance categories, though these differences remain moderate given the total of 32 exercises. Notably, the second evaluation introduced three Under Expectation cases (below 80%), which were absent in the first. However, this shift represents less than 10% of the total sample and does not indicate a dramatic divergence in judgment. Similarly, the decrease in the number of Proficient diagrams (91–95%) from 9 to 5 was partially offset by the consistent count of Above Expectation diagrams (96–100%), which remained stable at 13 in both evaluations. Minor changes were also observed in the Acceptable category, which increased from 6 to 7 outputs.

Overall, the results show that while subjective interpretation plays a role in borderline cases, general trends were preserved between evaluators.

Furthermore, in order to adequately account for these small variations and enhance the objectivity of the evaluation process, the final analysis was based on the average score between the two assessments. This averaging strategy allows for a more balanced and normalized representation of model performance. This decision was grounded in the observation that neither evaluator consistently deviated in a systematic or exaggerated way and that both followed the same assessment criteria. Averaging serves to reduce the impact of evaluator-specific interpretation or strictness, providing a more balanced and stable representation of the actual quality of each diagram.

In this context, the mean score captures the shared perception of correctness across both evaluations while smoothing out minor individual differences. This ensures that no single evaluator disproportionately influences the final results and supports a more objective comparison across exercises and experimental conditions.

Table 3.22. Updated Performance Evaluation Mean Score (Two Examples)

Performance Category	**Updated Score**	**Total Exercise**
Under Expectation Performance	under 80% included	0
Minimum Standard Performance	from 81% to 85% included	4
Acceptable Performance	from 86% to 90% included	7
Proficient Performance	from 91% to 95% included	8
Above Expectation Performance	from 96% to 100% included	13

3.6.3 Similarity Analysis after First and Second Refinement of the Prompt

While the previous section provided a quantitative correctness evaluation of the generated UML diagrams based on predefined scoring criteria after first and second refinement of the prompt, this part of the analysis focuses on the similarity between the diagrams produced by LLMs and the official reference models. The objective is to examine how closely the generated diagrams replicate the expected structural patterns, relationships, and conceptual modeling choices.

This analysis considers both structural alignment (e.g., class composition, hierarchy, association types) and semantic adherence (e.g., appropriate naming, role consistency, use of generalizations or enumerations). Rather than relying on numeric scores alone, this section offers a comparative discussion of modeling decisions made by the LLMs and evaluates whether they converge toward expert-level UML modeling practices.

The comparison highlights recurring deviations, common patterns of approximation, and cases where the generated models exhibit high or low fidelity to the reference solutions. This approach enables a deeper understanding of the models' capabilities and limitations when tasked with structured conceptual modeling.

Hence, following the first refinement of the prompting strategy, the similarity analysis was conducted employing the same evaluation rubric and classification criteria adopted in the initial phase, ensuring consistency in the comparison. The results were grouped as before into four categories based on similarity percentage ranges: Low (0–59%), Moderate (60–74%), High (75–89%), and Very High (90–100%).

As shown in Table 3.23, the majority of the generated diagrams (18 out of 32) still fall into the Low Similarity range, given that the previous result in the same band was 22, indicating that despite the improved prompting instructions, many outputs continue to diverge significantly from the reference models in terms of structure, semantics, or both. However, a noticeable shift is observed in the upper categories: 7 diagrams exhibit Moderate Similarity rather than only 4 in the previous analysis, 3 achieved High Similarity, decreased than before, and mostly, 4 diagrams were classified as Very High Similarity, reflecting an overall improvement in the generation quality, since in the previous analysis the result in this band was equal to zero.

These findings suggest that the refinement contributed to a slightly more accurate interpretation of the modeling task in a subset of cases, particularly where the conceptual structure was simpler or more canonical. Nonetheless, the high number of low-similarity outputs indicates that the LLM still has some difficulties in representing complex or ambiguous modeling scenarios.

Table 3.23. Similarity Categories – After First Prompt Refinement

Category	**Interval (included)**	**Results**
Low Similarity	0–59%	18
Moderate Similarity	60–74%	7
High Similarity	75–89%	3
Very High Similarity	90–100%	4

A specific phase of the similarity analysis focused on the phenomenon of overgeneration, as analyzed before the prompt refinement and defined as the tendency of LLM-generated diagrams to include more elements, such as classes, attributes, or relationships, than the corresponding human-made reference diagrams. This behavior may indicate either an attempt by the model to overcompensate for ambiguity in the textual description, or an imprecise mapping between linguistic cues and modeling constructs.

Table represented in Table 3.24 reports the frequency of overgeneration after the first refinement of the prompt.

Since the primary objective of this section is to assess whether the refined prompt led to measurable improvements or regression in the LLM's modeling performance, it is useful to reintroduce the corresponding results obtained using the original (pre-refinement) version of the prompt. This enables a direct comparison between the first and the second prompt refinements:

Before the refinement:

- In 88% of the exercises (28 out of 32), the LLM produced more attributes than expected.

Table 3.24. Overgeneration Issues in LLM-Based Diagrams – First Prompt Refinement

Problem Explanation	Number of Exercise	% over TOT Exercise
LLM-based diagram presents more attributes than the human-made diagram	28	88%
LLM-based diagram presents more classes than the human-made diagram	8	25%
LLM-based diagram presents more relationships than the human-made diagram	6	19%

- In 25% of cases (8 exercises), the LLM included more classes.
- In 19% of the diagrams (6 exercises), there were additional relationships compared to the human reference.

After the refinement:

- The overgeneration of attributes decreased significantly, affecting 66% of the exercises (21 out of 32).
- The number of diagrams with extra classes increased to 34% (11 exercises).
- Similarly, relationships were overgenerated in 31% of the cases (10 exercises).

These results suggest that the refined prompt successfully reduced unnecessary attribute proliferation, likely due to more explicit constraints and guidance in the prompt formulation. However, the increase in the number of classes and relationships may indicate that the LLM, now better understanding the structural decomposition of the system, attempts to introduce additional modeling elements in an effort to improve completeness or align with implicit patterns in the training data.

This trade-off highlights the challenge of balancing precision and completeness in automated modeling and suggests that further prompt tuning should focus on improving semantic grounding, particularly in relation to class granularity and the abstraction level of associations.

Evaluation after the Second Prompt Refinement (Few-Shot Prompting)

To further enhance the reliability of the generated UML diagrams, a second refinement of the prompt was performed. This version adopted a few-shot prompting strategy, in which two example exercises (word format) with their corresponding expected solutions (JSON format) were provided as input. The intention was to offer the LLM a clearer modeling pattern to emulate, to test whether the model could better replicate the expected UML diagrams.

The introduction of a second prompt refinement, based on a few-shot approach including two solved exercises as input, produced a notable shift in the similarity distribution of the generated UML diagrams. As shown in Table 3.25, the number of diagrams falling

into the "Low Similarity" category dropped significantly from 18 to 11, indicating a substantial reduction in poor-emulating outputs.

Table 3.25. Similarity Categories – After Second Prompt Refinement

Category	**Interval (included)**	**Results**
Low Similarity	0–59%	11
Moderate Similarity	60–74%	10
High Similarity	75–89%	5
Very High Similarity	90–100%	4

At the same time, there was a clear increase in diagrams achieving Moderate Similarity (from 7 to 10) and High Similarity (from 3 to 5). The number of diagrams with Very High Similarity remained stable (4 in both cases), suggesting that the few-shot prompting helped bring more diagrams closer to the reference level, but may not have been sufficient to further increase the highest-quality matches.

Overall, these results indicate that the second refinement improved the model's consistency and alignment with the expected modeling patterns, particularly by reducing the generation of structurally divergent diagrams. The moderate-to-high similarity range now encompasses nearly 60% of the total outputs, compared to only 43% in the previous configuration, demonstrating a measurable improvement in the LLM's capacity to generalize from provided examples.

In addition to evaluating similarity scores, we re-analyzed the persistence of overgeneration errors after applying the second prompt refinement. The objective was to understand whether the introduction of few-shot prompting would reduce the frequency of diagrams including excessive elements compared to the human-designed references.

As shown in Table 3.26, the results indicate a slight improvement in this regard. The proportion of diagrams with excessive attributes decreased marginally, from 66% to 63%, suggesting a limited but consistent impact of the new examples in guiding the model toward more concise class definitions.

More notable is the reduction in the number of diagrams that overgenerated relationships (from 31% to 25%) and classes (from 34% to 31%). This suggests that the few-shot strategy helped the model better grasp the level of abstraction expected in the modeling task, leading to more selective inclusion of entities and associations.

However, the persistence of overgeneration, even if slightly reduced—highlights the d**ifficulty LLMs face in mapping vague or implicit information** in natural language to precise modeling constructs. It also confirms that prompt refinement alone, while beneficial, may not fully eliminate systematic tendencies such as attribute inflation or redundant class creation.

Table 3.26. Overgeneration Issues in LLM-Based Diagrams – After Second Prompt Refinement

Problem Explanation	**Number of Exercise**	**% over TOT Exercise**
LLM-based diagram presents more attributes than the human-made diagram	20	63%
LLM-based diagram presents more classes than the human-made diagram	10	31%
LLM-based diagram presents more relationships than the human-made diagram	8	25%

Chapter 4

Conclusion

This thesis investigated the potential of Large Language Models (LLMs) to generate and evaluate UML class diagrams starting from textual requirements. The research was focused on the dual objective of assessing both the correctness and the structural similarity of LLM-generated Assessing correctness ensures that the diagrams adhere to the formal syntactic and semantic rules of UML, reflecting valid modeling constructs such as well-formed classes, attributes, and relationships, while evaluating structural similarity allows for the measurement of how faithfully the LLM has interpreted and replicated the modeling intent expressed in the reference solutions reflecting alignment with human reasoning and abstraction choices, especiaally in terms of implicit instructions.

The experimental setup incorporated a manually curated dataset of UML modeling exercises, an iterative prompt refinement process, and a rubric-based evaluation protocol. Over the course of the experimentation, three successive prompt versions were designed, culminating in a final few-shot configuration aimed at optimizing the fidelity, consistency, and semantic alignment of the generated models.

4.1 Summary of Contributions

The main contributions of this thesis are threefold:

1. First, it demonstrates the feasibility of leveraging LLMs as generative tools for conceptual modeling in UML, bridging the gap between natural language requirements and formal design artifacts.

2. Second, it introduces a robust prompt engineering strategy that incrementally evolves from basic role-based instructions to a structured, few-shot approach, achieving measurable improvements in output quality.

3. Third, it presents an integrated rubric-based framework for evaluating both syntactic and semantic correctness, as well as structural similarity, enabling a comprehensive assessment of LLM performance in the context of software modeling tasks.

The experimental results confirmed the effectiveness of the prompt engineering strategy. The refined prompts led to significant increases in the proportion of diagrams rated as *Proficient* or *Above Expectation*, and a notable reduction in critical modeling errors, particularly syntactic violations. The transition from single-shot to few-shot prompting further improved model alignment with expected structural patterns, reducing overgeneration in attributes and relationships and raising the proportion of high-similarity diagrams.

4.2 Evaluation Insights and Key Findings

The findings underscore several key insights about the behavior and limitations of LLMs in UML modeling tasks. Notably, the correctness analysis revealed that LLMs are highly competent at adhering to syntactic constraints, consistently producing JSON files compatible with the Apollon UML editor. However, challenges remained in accurately modeling complex relationships, especially when the textual descriptions lacked explicit cues.

In terms of similarity, the progression from the initial prompt to the final two-shot configuration revealed a consistent improvement. Despite these gains, only a minority of diagrams achieved *Very High Similarity*, suggesting that fully faithful replication of human conceptual models remains a difficult goal.

The phenomenon of overgeneration was also systematically evaluated. The initial prompts often led to inflated attribute counts, a side effect of rigidly imposed completeness rules. Later refinements introduced softer constraints, which helped balance richness with precision.

These results reinforce the critical role of **prompt engineering** in steering LLM behavior. Carefully crafted prompts, with clear structure, explicit constraints, and meaningful examples, can significantly enhance the reliability and semantic depth of the outputs.

Second, the results highlight the promise of LLMs in **educational modeling scenarios**. LLMs are capable of generating well-structured class diagrams with minimal supervision, which could support students in early-stage learning activities. However, persistent issues with overgeneration underscore the importance of human oversight.

Third, the research exposes the tension between *correctness* and *similarity*. While a diagram may be structurally valid, it may still diverge from a reference solution due to differences in abstraction or modeling choices. This raises questions about what it means for a model to be "correct enough" in pedagogical contexts and **what can be defined as variants of the same diagrams**.

4.3 Limitations and Future Work

While this thesis provides a structured and replicable framework for evaluating LLM-generated UML class diagrams, several avenues remain open for future research.

- **Extension to other UML diagram types**: The current study focused exclusively on class diagrams. Future work could extend the methodology to cover additional UML diagram types such as *sequence diagrams*, *activity diagrams*, or *state*

machines, which introduce dynamic behavior and temporal interactions, thus offering a more complete testbed for evaluating LLM-based modeling capabilities.

- **Integration of automated similarity metrics**: The rubric-based approach adopted in this study proved effective but required manual interpretation. Future evaluations could benefit from automatic graph comparison techniques, such as graph matching algorithms or embedding-based similarity metrics. This could help reduce evaluation time and increase objectivity. However, it is important to address current limitations: many available tools are not equipped to handle multilingual scenarios where textual descriptions are in Italian and generated solutions are in English, as was the case in this study. This language mismatch introduces semantic ambiguity and makes it difficult for tools to accurately match concepts and roles across diagrams.
- **Use of different LLMs and fine-tuned models**: This thesis primarily evaluated the performance of a single general-purpose LLM. Future experiments could broaden the scope by including multiple LLMs, such as open-source models (e.g., LLaMA, Mistral) or domain, adapted fine-tuned variants. This would allow for a comparative analysis of architectural and training differences and their effects on diagram generation quality. It would also provide a stronger basis for generalizing the findings beyond the model used in this work.
- **Expansion of human evaluation base**: Given the inherent subjectivity in human scoring, another possible enhancement would be to increase the number of independent evaluators involved in the rubric-based assessment. Aggregating multiple evaluations could improve the robustness of the analysis and minimize individual bias, especially in borderline or ambiguous cases. A consensus-based or averaged scoring approach may also help better normalize differences in interpretation.

In summary, future work should aim to increase both the *breadth* (by diversifying LLMs and UML types) and the *depth* (by improving evaluation methodology and tooling) of the experimental setup. This would contribute to a more comprehensive and generalizable understanding of how generative models can be effectively used in conceptual modeling contexts.

Final Remarks

This thesis demonstrated that LLMs, when properly guided, can generate UML diagrams that are syntactically valid and pedagogically meaningful. While not a replacement for human expertise, they can support modeling tasks, automate generation, and provide formative feedback in academic contexts.

The findings contribute a practical methodology for evaluating UML diagrams and a theoretical framework for understanding the role of generative models in structured domains. As AI tools continue to evolve, their integration into software design workflows should be guided by robust prompt engineering, rigorous evaluation, and responsible human oversight.

Bibliography

[1] Ali Almusawi, M. Rizwan Qureshi, and Mamdouh Alenezi. Conceptual modeling education: A systematic literature review. *Computer Applications in Engineering Education*, 29(1):34–56, 2021.

[2] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on uml class diagrams. *Artificial Intelligence*, 168(1–2):70–118, 2005.

[3] Narasimha Bolloju and Felix S.K. Leung. Assisting novice analysts in developing quality conceptual models with uml. *COMMUNICATIONS OF THE ACM*, pages 108–112, 2006.

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[5] Mohammad Fauzan, Abdul Wahid, and Evi Suryani. Grading uml class diagrams automatically using graph matching. *Procedia Computer Science*, 179:399–407, 2021.

[6] Fahad Hussain, Sufyan Zafar, Noreen Ikram, and Atta Ur Rehman Wahab. Modeling practices in software development: a survey. *Journal of Systems and Software*, 172:110736, 2021.

[7] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Ishii. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

[8] Xinyu Li, Chenguang Zhao, Jie Wang, and Tao Zhang. On the limitations of structural parsing for llm-generated artifacts. *arXiv preprint arXiv:2310.04675*, 2023.

[9] Tosif Ul Rehman Nacir Bouali, Marcus Gerhold and Faizan Ahmed. Toward automated uml diagram assessment: Comparing llm-generated scores with teaching assistants. *CSEDU 2025 - 17th International Conference on Computer Supported Education*, 2025.

[10] 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. *The Tenth International Conference on Software Engineering Advances*, pages 40–41, 11 2015.

[11] Marian Petre. Uml in practice. *Proceedings of the 2013 International Conference on Software Engineering*, pages 722–731, 2013.

[12] Dorsa Sadigh, Sanjit A. Seshia, and Mona Gupta. Automating exercise generation: a step towards meeting the mooc challenge for embedded systems. In *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*, WESE '12, New

York, NY, USA, 2012. Association for Computing Machinery.

[13] Matthias Schöttle, Nishanth Thimmegowda, Omar Alam, Jörg Kienzle, and Gunter Mussbacher. Feature modelling and traceability for concern-driven software development with touchcore. In *Companion Proceedings of the 14th International Conference on Modularity*, MODULARITY Companion 2015, page 11–14, New York, NY, USA, 2015. Association for Computing Machinery.

[14] Anthony Tang, Jorge Mejia, and David Lowe. Assessing students' understanding of object-oriented modeling and design. In *Proceedings of the 2010 ICSE Workshop on Modeling in Software Engineering*, pages 33–38, 2010.

[15] 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.

[16] Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 31(3):79–85, 2014.

[17] Chengyu Zhu, Yixin Wang, Yutai Guo, Zhengbao Zhang, and Jiliang Tang. A systematic survey of prompt engineering in large language models. *Rotman Digital Working Papers*, 2024. https://rotmandigital.ca/wp-content/uploads/2024/09/A-Systematic-Survey-of-Prompt-Engineering-in-Large-Language-Models.pdf.