POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Engineering and Management

Master Degree Thesis

Analisi comparativa e valutazione di sistemi LLM nella generazione di diagrammi delle classi UML

Supervisors
Prof. Riccardo Coppola
Doct. Giacomo Garaccione

Candidato
Diego Maria Calabrese

Academic Year 2024-2025

Sommario

La diffusione dei Large Language Models (LLM), modelli linguistici avanzati basati su architetture Transformer, ha rivoluzionato diversi settori, tra cui l'ingegneria del software e i sistemi informativi. In questi settori si è infatti palesata la possibilità, tra le tante, di utilizzare i LLM per automatizzare la modellazione concettuale. In particolare, la generazione automatica di diagrammi UML delle classi a partire da requisiti testuali rappresenta un'opportunità ancora poco esplorata. Questa tesi si propone di colmare questa lacuna attraverso un'analisi comparativa delle prestazioni di quattro LLM di ultima generazione (GPT-4o, Gemini 2.5 Pro, Qwen 3, DeepSeek v3) nella generazione di diagrammi UML a partire dalla descrizione del sistema scritta in linguaggio naturale. Questo argomento risulta interessante sia per il mondo industriale, dove automatizzare la modellazione concettuale potrebbe rendere più efficienti le prime fasi dello sviluppo software, sia in ambito didattico, dove strumenti basati su intelligenza artificiale possono aiutare gli studenti a capire e applicare meglio i concetti della modellazione. Tuttavia, passare da una descrizione testuale a un diagramma UML formale non è un processo semplice. Richiede infatti la capacità di interpretare correttamente il testo, individuare le informazioni rilevanti e rispettare regole sintattiche ben precise, aspetti che ad oggi mettono in difficoltà anche i modelli più evoluti. Il lavoro di tesi ha previsto la costruzione di un dataset di 15 esercizi realistici di modellazione concettuale, redatti in italiano, con livelli di difficoltà differenti. Ogni esercizio è stato classificato secondo criteri di complessità sintattica, ambiguità semantica e presenza di vincoli impliciti. I modelli sono stati guidati tramite prompt standardizzati, costruiti con tecniche di prompt engineering (few-shot, role prompting, output-constrained) per produrre output in formato JSON compatibile con un UML Modeler, uno strumento accademico per la modellazione UML. La valutazione è stata condotta tramite una griglia a sei criteri (conformità UML, validità JSON, validità semantica, completezza, chiarezza grafica, comprensibilità), assegnando un punteggio da 0 a 2 per ciascun aspetto. I risultati ottenuti mostrano che tutti i modelli sono in grado di generare file JSON formalmente validi nella maggior parte dei casi, ma emergono differenze significative in termini di completezza e interpretazione semantica. GPT-4o e Gemini si sono dimostrati i modelli più accurati nella traduzione dalle tracce testuali ai modelli, mentre Qwen e DeepSeek hanno introdotto errori più frequenti nella rappresentazione delle relazioni e nell'uso delle generalizzazioni. Le difficoltà maggiori per tutti i modelli sono emerse nei casi ad alta ambiguità semantica, dove era necessario inserire relazioni o entità non esplicitamente descritte. Nonostante ciò i modelli hanno mostrato buone prestazioni in contesti a bassa complessità e in esercizi con vincoli ben strutturati, evidenziando il ruolo cruciale di progettazione del prompt e chiarezza della traccia. Lo studio conferma il potenziale dei LLM come strumenti di supporto alla modellazione UML, soprattutto in ambito didattico. Ne evidenzia tuttavia i limiti attuali, proponendo come sviluppi futuri l'adozione di prompt modulari e l'integrazione dei LLM in ambienti educativi dove il modello possa aiutare lo studente nella costruzione del diagramma.

Indice

Capitolo 1

Introduzione

1.1 Contesto e motivazioni

Negli ultimi anni, i Large Language Models (LLM) hanno acquisito un ruolo centrale nell'evoluzione dell'intelligenza artificiale, grazie alla loro capacità di elaborare e generare linguaggio naturale con un livello di precisione e flessibilità crescente. L'impiego di tali modelli si è rapidamente esteso a numerosi ambiti applicativi, tra cui l'ingegneria del software, l'analisi dei dati e l'automazione dei processi decisionali.

In particolare, l'attenzione della comunità scientifica si è progressivamente orientata verso l'esplorazione del potenziale dei LLM in attività tradizionalmente affidate alla modellazione concettuale manuale, come la produzione di diagrammi UML a partire da specifiche testuali. Tale approccio risulta di interesse sia in ambito industriale, in quanto consente di semplificare la documentazione e l'evoluzione dei sistemi software, sia in ambito didattico, dove può costituire uno strumento di supporto per la verifica automatica degli elaborati prodotti dagli studenti.

Tuttavia, l'integrazione dei LLM nei processi di modellazione solleva numerose questioni di carattere metodologico e qualitativo. In particolare, si rende necessario analizzare in modo sistematico le capacità effettive dei modelli nel produrre rappresentazioni sintatticamente corrette, semanticamente coerenti e funzionalmente utili, a partire da requisiti espressi in linguaggio naturale.

1.2 Obiettivi della tesi

Il presente lavoro si propone di valutare in modo comparativo l'efficacia di diversi LLM nella generazione automatica di diagrammi delle classi UML, con riferimento a un insieme strutturato di esercizi di modellazione concettuale. In particolare, gli obiettivi specifici della tesi sono i seguenti:

- analizzare la qualità dei diagrammi UML generati da quattro modelli LLM di ultima generazione, sulla base di sei criteri di valutazione afferenti alle dimensioni sintattica, semantica e pragmatica;

- esaminare le variazioni di performance dei modelli al variare della complessità concettuale degli esercizi proposti;
- valutare l'influenza delle strategie di prompting sulla qualità dell'output prodotto, con riferimento a diverse formulazioni testuali della consegna;
- identificare e classificare le principali tipologie di errore ricorrenti nei diagrammi generati;
- formulare raccomandazioni operative per l'impiego dei LLM in contesti educativi e professionali legati alla modellazione del software.

1.3 Struttura del lavoro

La tesi è articolata nei seguenti capitoli:

- **Capitolo 2 - Background** – Presenta il quadro teorico di riferimento, includendo una panoramica sui LLM, sulle principali tecniche di prompting, sulle problematiche della modellazione UML e sugli strumenti esistenti per la valutazione automatica della qualità dei modelli.
- **Capitolo 3 - Metodologia** – Descrive l'impostazione metodologica adottata, specificando il dataset di esercizi utilizzato, i modelli analizzati, i criteri di valutazione, le modalità di prompting e la procedura di raccolta dei dati.
- **Capitolo 4 - Risultati sperimentali** – Riporta e discute i risultati ottenuti, proponendo un'analisi quantitativa e qualitativa delle performance dei modelli, articolata in funzione dei criteri di valutazione e della difficoltà degli esercizi.
- **Capitolo 5 - Conclusioni** – Fornisce le considerazioni conclusive, evidenziando i principali contributi dello studio, le sue limitazioni e le possibili direzioni di ricerca futura.

Capitolo 2

Background

2.1 Large Language Models

I Large Language Models (LLM) sono modelli di intelligenza artificiale di grandi dimensioni, basati sull'architettura Transformer [18] introdotta da Vaswani et al. nel 2017: tale tecnologia ha rivoluzionato in modo radicale il Natural Language Processing (NLP), campo dell'intelligenza artificiale che si occupa dell'interazione tra individuo e computer. L'architettura Transformer sostituisce le reti neurali ricorrenti con un meccanismo di attenzione chiamato "Scaled Dot-Product Attention", che consente al modello di ponderare l'importanza di ciascun "token", cioè parola, rispetto agli altri nella "sequenza" o frase. Questa proprietà migliora non solo la capacità del modello di catturare dipendenze a lungo raggio, ma anche di elaborare le sequenze in parallelo.

Grazie alla loro scalabilità, i LLM possono essere addestrati su grandi quantità di dati testuali, diventando capaci di affrontare una molteplicità di compiti, anche complessi, senza essere specializzati su uno specifico dominio.

Tra le caratteristiche che rendono i LLM particolarmente versatili si evidenziano:

- In-context learning (ICL): i LLM sono in grado di apprendere nuovi compiti direttamente dal contenuto fornito nel prompt, ovvero un'istruzione o input, senza modificare i propri pesi. Attraverso pochi esempi o una semplice descrizione in linguaggio naturale, il modello può generalizzare e generare output coerenti, anche in modalità zero-shot o few-shot [2].

- Instruction tuning: mediante una fase di fine-tuning, un pre-addestramento fornito grazie all'ausilio di un data set di dimensioni inferiori, supervisionato su un insieme di compiti descritti da istruzioni testuali, il modello migliora la propria capacità di comprendere e seguire indicazioni esplicite, risultando più efficace anche su task non visti durante l'addestramento [20]. E' così possibile ampliare non solo il dominio di task eseguibili, ma anche migliorarne la loro qualità.

- Chain-of-thought prompting (CoT): questa tecnica consiste nel fornire esempi di ragionamento passo-passo all'interno del prompt, favorendo lo sviluppo di una

sequenza di passaggi logici intermedi necessari per giungere a risposte più strutturate e corrette. Tale approccio ha dimostrato di migliorare significativamente le prestazioni dei LLM in compiti che richiedono ragionamento logico o matematico [21].

Recenti ricerche hanno esplorato l'uso di modelli generativi per la creazione automatica di diagrammi UML a partire da descrizioni testuali, dimostrando che i LLM sono in grado di produrre diagrammi formalmente validi e comprensibili, pur presentando limiti in termini di completezza o coerenza semantica rispetto ai requisiti originari [5]. Tali risultati aprono la strada allo sviluppo di tecniche di prompting specializzate, finalizzate a migliorare l'accuratezza e l'utilità dei modelli generati nel contesto della modellazione concettuale.

2.2 UML Class Diagram

2.2.1 Definizione

Nel contesto dell'ingegneria del software, il linguaggio UML (Unified Modeling Language) rappresenta uno standard industriale per la modellazione visiva dei sistemi. Introdotto e mantenuto dall'Object Management Group (OMG), UML consente di visualizzare, costruire e documentare i componenti di un sistema software, facilitando la comunicazione tra analisti, progettisti e sviluppatori [13].

Tra i diversi tipi di diagrammi previsti dal linguaggio UML, il diagramma delle classi è uno degli strumenti più rilevanti nella progettazione orientata agli oggetti. Esso descrive la struttura statica del sistema attraverso la rappresentazione delle classi, con relativi attributi, operazioni, e relazioni tra di esse [6].

I principali elementi rappresentati in un diagramma delle classi sono:

- **Classi**: entità che aggregano dati (attributi) e comportamenti (metodi). Ogni classe è rappresentata da un rettangolo suddiviso in tre compartimenti: nome, attributi e operazioni.
- **Relazioni**: collegamenti tra classi, tra cui associazioni, aggregazioni, composizioni ed ereditarietà. Ogni tipo implica un diverso livello di dipendenza e semantica.
- **Molteplicità**: indica il numero di istanze ammesse nella relazione tra due classi.
- **Visibilità**: specifica l'accessibilità di attributi e metodi (es. + pubblico, - privato, # protetto, ~ package).

I diagrammi delle classi possono essere utilizzati sia nella fase di modellazione concettuale (per rappresentare entità e relazioni del dominio), sia nella fase di progettazione dettagliata (come modello di riferimento per l'implementazione del codice).

2.2.2 Qualità di un Diagramma UML

Per valutare la qualità di un diagramma delle classi generato automaticamente, è possibile ricorrere a una serie di criteri formali:

- **Completezza**: il diagramma deve riflettere tutte le informazioni rilevanti contenute nella descrizione testuale, comprese classi, attributi, metodi e relazioni.
- **Correttezza sintattica**: conformità alla notazione UML standard, nel rispetto di simboli, molteplicità, visibilità e connessioni.
- **Coerenza semantica**: aderenza alla logica del dominio applicativo, tale da garantire che le relazioni rappresentate siano significative rispetto al contesto.
- **Organizzazione strutturale**: disposizione chiara ed equilibrata degli elementi nel layout, che favorisca la leggibilità e l'interpretazione del diagramma.
- **Assenza di ambiguità**: il modello deve evitare generalizzazioni eccessive o relazioni mal definite, garantendo precisione semantica.

UML si basa su un metamodello formale che ne definisce la sintassi e ne permette la verifica automatica attraverso l'uso di vincoli, spesso espressi in Object Constraint Language (OCL) [12]. Alcuni strumenti di modellazione supportano controlli automatici di correttezza sintattica, noti anche come good-form checks.

Oltre alla correttezza formale, negli ultimi anni sono stati proposti diversi metodi per valutare anche la qualità strutturale dei diagrammi, attraverso metriche tradizionali dell'ingegneria del software, come coesione, accoppiamento e complessità [3].

In ambito educativo, infine, sono in fase di sviluppo tecniche per il confronto automatico tra i modelli realizzati dagli studenti e le soluzioni di riferimento, con l'obiettivo di supportare sia l'apprendimento che la valutazione automatizzata [17]. Questi criteri risultano essenziali per una valutazione sistematica dei diagrammi generati da Large Language Models, in quanto permettono di confrontare in modo oggettivo le soluzioni prodotte rispetto a standard consolidati.

2.3 Prompt

2.3.1 Definizione di prompt

Nel contesto dei Large Language Models (LLM), il termine prompt indica l'input testuale fornito al modello per ottenere una risposta. Il prompt può consistere in una semplice istruzione, una domanda o una descrizione di un compito con annessi esempi di input e output.

A seconda della complessità del compito i prompt possono variare da istruzioni molto brevi a strutture articolate e multilivello. Nei casi in cui si richieda un output formale e strutturato, come la generazione automatica di diagrammi UML, il design del prompt assume un'importanza critica.

2.3.2 Prompt Engineering

Il prompt engineering è l'attività di progettazione di input testuali mirati a guidare i modelli linguistici di grandi dimensioni (LLM) nella produzione di output coerenti, rilevanti e di qualità. Con l'evoluzione dei LLM, tale pratica si è consolidata come disciplina autonoma, fondendo principi del linguaggio naturale, del machine learning e dell'ingegneria del software.

Nei sistemi basati su LLM, il comportamento del modello è interamente determinato dal contenuto del prompt: strutturare correttamente questo input permette di influenzare in modo sostanziale la qualità della risposta generata. In assenza di un fine-tuning specifico sul compito, l'intero processo di condizionamento del modello avviene in input, motivo per cui è stato spesso descritto come una forma di programmazione linguistica. Tale potere espressivo ha portato alla definizione di pratiche, convenzioni e strategie note come *prompt engineering*.

2.3.3 Tecniche comuni di prompt engineering

Di seguito vengono illustrate alcune delle tecniche più comuni di prompt-engineering:

- Zero-shot prompting: si basa su un'unica istruzione senza esempi, sfruttando le capacità generali del modello.
- Instruction prompting: consiste nel fornire un'istruzione diretta e chiara, ad esempio "Genera una descrizione riassuntiva del seguente testo...".
- Few-shot prompting: utilizza uno o più esempi di input-output all'interno del prompt per mostrare al modello il formato e il contenuto attesi.
- Chain-of-thought prompting: supporta il modello nel ragionamento necessario per ottenere l'output atteso, rendendo esplicito il processo.
- Role-based prompting: ad esempio "Agisci come un docente universitario di ingegneria del software...".
- Output formatting: definisce esplicitamente il formato dell'output desiderato, come nel caso di "Fornisci la risposta in formato JSON valido secondo questo schema...".
- Self-consistency: prevede la generazione di più risposte e la selezione della più coerente tramite confronto automatico.

2.3.4 Prompting per la generazione di diagrammi UML

Negli scenari in cui si richiede la generazione automatica di diagrammi UML da parte di modelli linguistici, il prompt engineering assume un ruolo centrale nel garantire l'accuratezza, la coerenza e la struttura formale dell'output. A differenza di compiti più generici,

in cui è sufficiente produrre testo naturale, in questo contesto è essenziale guidare il modello verso la produzione di output rigorosamente strutturati — come rappresentazioni in JSON, PlantUML o linguaggi specifici compatibili con tool di modellazione.

Questa esigenza ha portato allo sviluppo di strategie di prompting specializzate:

- la definizione esplicita nel prompt del formato dell'output atteso;
- l'utilizzo di esempi di input-output che illustrino la struttura desiderata;
- la conformità semantica, ovvero l'allineamento del contenuto generato alle convenzioni UML standard;
- la robustezza, intesa come consistenza e validità sintattica dei file generati nei linguaggi target (es. validazione JSON).

Per ottimizzare la formulazione del prompt, oltre alle tecniche citate sopra, è stato seguito un processo iterativo basato sull'analisi di esercizi di training: per ciascun errore riscontrato nel diagramma generato, è stato richiesto al modello stesso di motivare la causa dell'errore e di proporre una riformulazione del prompt che ne evitasse la ricorrenza nei successivi tentativi.

2.3.5 Promptware Engineering

Con l'aumento della complessità dei compiti affidati ai LLM e la crescente importanza della qualità del prompt nel determinare l'efficacia dell'output, è emersa l'idea di trattare i prompt non come semplici stringhe di testo, ma come componenti progettuali veri e propri, da costruire, testare e mantenere con cura. Questo approccio prende il nome di Promptware Engineering [23], e si ispira ai principi e alle pratiche consolidate dell'ingegneria del software.

L'assunto alla base è che, al pari del codice, anche i prompt possono (e dovrebbero) essere progettati, testati, documentati, versionati e riutilizzati nel tempo. In altre parole, i prompt diventano "software", con caratteristiche che li rendono affidabili, scalabili e manutenibili all'interno di processi produttivi strutturati.

Nel contesto della generazione automatica di diagrammi UML, questo significa ad esempio:

- progettare prompt modulati per task specifici, come l'identificazione delle classi, l'estrazione delle associazioni o la normalizzazione degli attributi;
- definire template riutilizzabili per diverse tipologie di esercizi o domini applicativi;
- associare a ciascun prompt test in grado di verificarne la robustezza e la correttezza;
- tenere traccia delle modifiche e dell'evoluzione dei prompt nel tempo;
- integrare i prompt in processi automatizzati, come parte di sistemi di supporto alla modellazione o strumenti didattici.

Questa prospettiva consente di trattare il prompt non come un elemento accessorio, ma come una componente a pieno titolo di un sistema software basato su LLM. Il Promptware Engineering pertanto promuove una visione più disciplinata, tracciabile e scalabile della progettazione dei prompt, ponendosi all'incrocio tra la programmazione in linguaggio naturale e gli approcci classici a progettazione e sviluppo software. Il prompt pertanto non è più un input occasionale ma una vera e propria componente software soggetta a testing, refactoring e tracciabilità.

Nel medio-lungo termine, questo approccio potrebbe favorire la creazione di librerie di prompt condivise, ambienti collaborativi per la loro progettazione, e pratiche di testing automatico che garantiscano la qualità dei sistemi generativi in modo più sistematico.

2.3.6 Model-Driven Prompt Engineering

Un'evoluzione interessante del prompt engineering è rappresentata dal Model-Driven Prompt Engineering (MDPE) [22], un approccio che mira a introdurre maggiore astrazione, riusabilità e sistematicità nella progettazione dei prompt. L'idea centrale è quella di utilizzare linguaggi specifici di dominio (DSL, Domain-Specific Languages) per rappresentare i prompt in una forma strutturata e modellabile, analoga a quanto avviene nello sviluppo software basato su modelli (Model-Driven Engineering).

In questo contesto, i prompt non vengono più scritti direttamente come testo libero, ma vengono costruiti a partire da modelli ad alto livello che ne descrivono struttura, contenuto e vincoli. Ad esempio, si può definire un metamodello che stabilisce quali sezioni debba contenere un prompt (istruzioni, contesto, esempi, formato di output), con quali vincoli logici e in quale ordine, permettendo di generare automaticamente il prompt testuale finale a partire da tali specifiche.

Questo approccio offre diversi vantaggi:

- consente di generare varianti coerenti dello stesso prompt adattate a diversi contesti, semplicemente modificando il modello sottostante;
- rende il processo di progettazione dei prompt più tracciabile, grazie all'esplicitazione delle scelte progettuali in forma modellata;
- favorisce la verifica automatica di vincoli sintattici e semantici prima della generazione effettiva del prompt;
- abilita l'integrazione dei prompt in pipeline software strutturate, come parte di sistemi di generazione automatica o strumenti di supporto alla modellazione.

Il Model-Driven Prompt Engineering rappresenta quindi un tentativo di colmare il divario tra la flessibilità della scrittura in linguaggio naturale e la precisione richiesta in ambiti formali come la generazione di modelli UML. Pur trattandosi di una linea di ricerca ancora emergente, essa apre prospettive promettenti per lo sviluppo di ambienti integrati in cui i prompt possano essere progettati, documentati e mantenuti con la stessa cura riservata agli artefatti software.

La progettazione di prompt per output strutturati come JSON beneficia fortemente dell'adozione di un'impostazione modulare. Un prompt modulare suddivide le istruzioni in blocchi semanticamente coerenti e indipendenti, facilitando sia la comprensione da parte del modello che l'evoluzione iterativa del prompt stesso. In questo lavoro, l'efficacia del Prompt Qwen è stata favorita anche da una chiara strutturazione in sezioni (header, elementi, relazioni, esempi, vincoli speciali) che ne ha aumentato la robustezza e l'adattabilità.

2.4 Difficoltà nella Modellazione UML

La modellazione UML, sebbene supportata da notazioni standardizzate, è spesso soggetta a errori concettuali e ambiguità, soprattutto quando viene svolta da utenti inesperti o in contesti educativi.

Diversi studi hanno evidenziato come studenti e principianti incontrino difficoltà nel rappresentare correttamente il dominio concettuale attraverso diagrammi delle classi. Studi come quelli di Santos et al. [15], Tanaka et al. [16] e Liu et al. [11] hanno sistematizzato queste problematiche attraverso analisi empiriche su esercitazioni svolte da studenti universitari.

In particolare, vengono riscontrati frequentemente i seguenti problemi:

- Astrazione eccessiva o insufficiente: gli utenti tendono a generalizzare concetti che dovrebbero essere specifici, oppure introducono dettagli superflui, perdendo di vista la struttura logica del sistema.
- Ambiguità semantica: la descrizione dei requisiti può generare ambiguità interpretative, che si riflettono in modelli incoerenti con il dominio reale.
- Errori sintattici e strutturali: includono l'uso scorretto delle relazioni (es. confusione tra associazione e composizione), molteplicità errate o attributi fuori contesto.
- Nominalismo eccessivo: anzichè astrarre i concetti, gli studenti si limitano a "trascrivere" i nomi presenti nel testo, senza riflessione sul ruolo concettuale delle entità.

La presenza di questi errori nei modelli umani fornisce un importante termine di paragone per la valutazione dei diagrammi UML generati automaticamente da LLM. Infatti, eventuali difetti nei diagrammi generati possono essere analizzati non solo in senso assoluto, ma anche rispetto alle difficoltà tipiche degli esseri umani, offrendo un metro comparativo più realistico nel contesto educativo e sperimentale.

2.5 Tool per la Valutazione Automatica dei Diagrammi UML

La valutazione automatica dei diagrammi UML rappresenta un tema di crescente interesse, in particolare nell'ambito della didattica e della formazione ingegneristica.

L'obiettivo principale di questi strumenti è fornire un supporto alla correzione degli esercizi modellativi, riducendo il carico per i docenti e offrendo feedback immediato agli studenti.

Diversi studi hanno sviluppato tool in grado di confrontare i diagrammi prodotti dagli studenti con una soluzione di riferimento.

Un esempio significativo è il sistema proposto da Fauzan et al. [4], che utilizza la rappresentazione XMI (XML Metadata Interchange) per analizzare la struttura dei diagrammi e confrontarli con una risposta attesa.

Il confronto si basa su elementi formali come classi, attributi, metodi e relazioni, producendo un report dettagliato che evidenzia discrepanze e mancanze.

Un approccio simile è adottato da Romero et al. [14], che presentano un ambiente educativo integrato per la valutazione automatica dei diagrammi UML.

Il sistema è progettato per essere utilizzato in contesti accademici, permettendo di gestire su larga scala le esercitazioni degli studenti.

Altri contributi, come quello di Jain et al. [8], offrono una panoramica comparativa dei principali strumenti disponibili, evidenziandone punti di forza e limiti.

Tuttavia, nonostante i progressi, permangono alcune limitazioni comuni a molti di questi strumenti:

- Dipendenza da una soluzione attesa: il sistema richiede che l'insegnante fornisca un diagramma corretto di riferimento, da cui derivare i criteri di valutazione.
- Assenza di comprensione semantica: la valutazione si limita al confronto sintattico e strutturale, senza interpretare il significato del modello rispetto al dominio applicativo.
- Incapacità di trattare input in linguaggio naturale: la maggior parte dei tool non consente di partire da descrizioni testuali, rendendoli inadatti a pipeline che coinvolgono LLM.

Questi limiti evidenziano la necessità di strumenti più flessibili e semantici, capaci di integrarsi con sistemi di generazione automatica basati su modelli linguistici e in grado di valutare anche la coerenza concettuale delle soluzioni proposte.

2.6 Ricerche sulla Generazione Automatica UML

La possibilità di utilizzare modelli linguistici per generare diagrammi UML a partire da descrizioni testuali ha attirato crescente attenzione nella comunità scientifica, in particolare per il suo potenziale impatto nei contesti educativi, industriali e di prototipazione rapida.

Uno dei primi contributi significativi in questa direzione è stato fornito da Ferrari et al. [5], che hanno esplorato le potenzialità di ChatGPT nella generazione di diagrammi UML a partire da requisiti in linguaggio naturale. I risultati dello studio mostrano che, pur essendo in grado di produrre strutture formalmente corrette, i modelli linguistici

tendono a generare diagrammi incompleti o parzialmente allineati con le aspettative semantiche.

L'indagine evidenzia inoltre come la qualità del prompt influenzi in maniera determinante la validità del risultato.

Un altro ambito di ricerca riguarda la generazione automatica e la correzione di esercizi basati su diagrammi, come nel caso dello studio di Gomez-Abajo et al. [7].

Gli autori propongono un'infrastruttura per l'automatizzazione di esercizi UML all'interno della piattaforma Moodle, dimostrando che è possibile integrare tecniche di generazione guidata anche in contesti didattici standardizzati.

Parallelamente, studi come quello di Fauzan et al. [4] e Romero et al. [14] hanno analizzato strumenti per la valutazione automatica di diagrammi UML. Sebbene non focalizzati direttamente sulla generazione da LLM, questi lavori pongono le basi per la definizione di metriche e criteri di confronto applicabili ai modelli prodotti automaticamente, contribuendo alla costruzione di benchmark affidabili.

Nel complesso, queste ricerche confermano che l'applicazione dei LLM alla generazione di diagrammi UML è promettente, ma presenta ancora criticità. In particolare, emergono tre sfide principali:

- l'assenza di standard condivisi per la valutazione dei diagrammi generati;
- la difficoltà di garantire un'interpretazione corretta delle descrizioni testuali complesse;
- la necessità di prompt progettati con attenzione per ottenere output formalmente e semanticamente coerenti.

Queste considerazioni motivano l'approccio adottato in questa tesi, volto a esplorare in modo sistematico le condizioni e i limiti entro cui i LLM riescono a produrre diagrammi UML validi a partire da input testuali descrittivi.

2.7 Limiti Attuali delle Soluzioni

Nonostante i risultati incoraggianti, le soluzioni attuali per la generazione automatica di diagrammi UML tramite LLM presentano numerose criticità che ne limitano l'affidabilità e l'applicabilità su larga scala.

Un primo limite riguarda l'assenza di standard condivisi per la valutazione delle soluzioni generate. In letteratura si osserva una forte eterogeneità nei metodi di valutazione adottati, spesso basati su griglie costruite ad hoc o su confronti qualitativi poco formalizzati.

Questo rende difficile comparare i risultati ottenuti nei diversi studi e ostacola la costruzione di benchmark robusti e ripetibili.

In secondo luogo, molti studi si focalizzano su pochi esempi o su domini ristretti, con dataset poco diversificati.

La scarsa varietà di input testuali limita la possibilità di generalizzare le conclusioni e non consente di comprendere appieno la sensibilità dei modelli a variazioni linguistiche, ambiguità o incompletezze nei requisiti.

Un ulteriore problema è rappresentato dalla mancanza di allineamento semantico tra il contenuto testuale e l'output generato. I modelli, pur rispettando formalmente la sintassi UML, tendono a introdurre classi o relazioni non pertinenti, oppure a omettere elementi rilevanti presenti nella descrizione.

Questo fenomeno riflette la difficoltà dei LLM nel compiere un vero "grounding" concettuale, ovvero nel comprendere in profondità il dominio descritto e tradurlo in una rappresentazione strutturata coerente.

Infine, le soluzioni esistenti spesso non distinguono tra componenti core del modello e varianti opzionali o contestuali. L'assenza di questa distinzione riduce la precisione del modello generato e complica la sua valutazione.

Queste limitazioni evidenziano la necessità di studi più sistematici, in grado di:

- definire protocolli sperimentali ripetibili e scalabili;
- costruire dataset ampi, variabili e ben annotati;
- elaborare metriche di valutazione standardizzate e fondate teoricamente;
- progettare prompt e strategie di validazione che favoriscano la coerenza semantica e l'espressività strutturale.

La presente tesi si inserisce in questo panorama, proponendo un contributo metodologico che mira a colmare parte di queste lacune attraverso una sperimentazione mirata e documentata.

Capitolo 3

Metodologia

3.1 Obiettivi della sperimentazione

L'obiettivo principale di questo studio è la valutazione delle capacità dei diversi modelli linguistici di grandi dimensioni (LLM) nella generazione di diagrammi delle classi UML a partire da un prompt standardizzato e ripetibile e un dataset di esercizi strutturati che simulano scenari tipici della modellazione concettuale. In particolare, il nostro caso d'interesse si è focalizzato sulla generazione di output in formato JSON, compatibile con lo strumento Apollon e UML Modeler. Il formato JSON, richiesto esplicitamente dal prompt come singolo output, ha permesso di facilitare il caricamento dei file ottenuti in ambienti didattici o di valutazione automatica. Lo studio analizzerà inoltre la qualità dei diagrammi secondo dimensioni sintattiche, semantiche e pragmatiche.

L'analisi condotta è volta a valutare le seguenti criticità:

- Correttezza sintattica e validità formale dell'output generato. Si intende quindi il rispetto delle regole formali del linguaggio UML, che riguardano nello specifico classi, attributi e relazioni, e della sintassi del formato json, in termini di formattazione.
- Aderenza semantica tra quanto descritto nel testo e quanto rappresentato nel modello. Il modello deve essere in grado di rappresentare tutte le entità dichiarate nel testo fornitogli, le loro proprietà e relazioni che le inter-connettono.
- Completezza strutturale, intesa come presenza di tutti gli elementi rilevanti. Un output infatti, può risultare formalmente e semanticamente corretto, ma incompleto di elementi importanti, quali per esempio attributi rilevanti o associazioni altrettanto importanti.
- Chiarezza e leggibilità, ovvero nomi appropriati alla struttura del diagramma. Questa voce valuta la comprensibilità dell'output generato e la capacità del modello di generare degli "outcomes" non ambigui.

Oltre alla misurazione quantitativa, l'analisi ha anche l'importante scopo di identificare eventuali pattern ricorrenti di errore, ambiguità nella comprensione da parte dei

modelli o limiti attuali nella generazione automatica di diagrammi concettuali. Si intende infine discutere in che misura gli output ottenuti possano essere utilizzati come supporto alla didattica della modellazione o come base per sistemi di correzione automatica.

Apollon

Apollon è uno strumento web-based per la modellazione visuale, sviluppato presso la Technische Universität München (TUM), concepito principalmente per supportare l'insegnamento della modellazione concettuale. Il tool consente la creazione interattiva di diagrammi UML (principalmente diagrammi delle classi), offrendo un'interfaccia intuitiva e una rappresentazione conforme alle specifiche standard UML.

Una caratteristica distintiva di Apollon è sicuramente il supporto che permette di avere per la valutazione automatica dei diagrammi prodotti dagli studenti. Ciò avviene tramite un confronto semantico e strutturale con una soluzione di riferimento, altrettanto "human-made", ma spesso generata dal professore, consentendo in questo modo feedback e paragoni immediati e facilitando l'apprendimento iterativo. Il sistema supporta anche l'esportazione e l'importazione di diagrammi in formato JSON, permettendo l'integrazione con strumenti esterni e la generazione automatica di esercizi e soluzioni.

Grazie alla sua accessibilità online, Apollon è ampiamente impiegato in contesti accademici e sperimentazioni didattiche che coinvolgono la generazione e la valutazione automatica di modelli UML, rivelandosi uno strumento particolarmente adatto per studi sul prompt engineering e sulla generazione automatica di modelli tramite LLM([9], [19]).

3.2 Dataset di esercizi UML

Il dataset utilizzato per la sperimentazione è composto da 15 esercizi di modellazione concettuale, ciascuno dei quali presenta un diagramma delle classi UML a partire da una descrizione testuale del sistema. Le tracce, formulate in italiano, presentano una lunghezza media compresa tra 80 e 150 parole e descrivono scenari applicativi eterogenei con un linguaggio naturale tecnico ma non artificiale. Il dataset è stato costruito a partire da esercizi realmente utilizzati in contesti didattici universitari, da materiali ufficiali resi disponibili dai corsi universitari di Ingegneria o Informatica in vari atenei italiani, come l'Università degli Studi di Milano e l'Università di Trieste ed il Politecnico di Torino stesso, reperibili pubblicamente nelle sezioni didattiche dei loro siti.

Gli esercizi coprono tre principali categorie:

- *Domini amministrativi e gestionali*, come compagnie assicurative, sistemi di prenotazione, biblioteche e gestione eventi.
- *Domini sanitari ed educativi*, tra cui strutture ospedaliere, scuole e contesti universitari.
- *Domini tecnici e applicazioni generiche*, come sistemi di monitoraggio ambientale, parcheggi automatizzati, sistemi di voto elettronico e gestione della grande distribuzione.

La selezione è stata effettuata con l'obiettivo di rappresentare diverse tipologie di difficoltà concettuale. Più della metà delle tracce presenta ambiguità semantiche o elementi impliciti (es. relazioni non esplicitate, stati ricorrenti, cardinalità deducibili dal contesto). In circa un terzo dei casi sono presenti gerarchie tra entità, con generalizzazioni esplicite o implicite, mentre composizioni e aggregazioni compaiono in modo significativo in circa due terzi degli esercizi.

Le tracce sono state normalizzate nello stile e nel lessico, evitando formulazioni eccessivamente guidate o strutturate. E' importante segnalare che in nessun caso è stato fornito al modello un esempio esplicito di diagramma corretto. Questa scelta ha permesso di valutare esclusivamente la capacità del sistema di comprendere il dominio e produrre un output autonomo a partire da istruzioni testuali non strutturate, senza che lo stesso potesse essere influenzato da un diagramma di riferimento per la sua generazione.

3.2.1 Criteri di difficoltà

Al fine di comprendere precisamente il comportamento del modello in riferimento alle diverse tipologie di esercizio, è stata condotta un' analisi, con successiva categorizzazione, sulla difficoltà di ogni esercizio compreso nel dataset. Per caratterizzare il livello di difficoltà degli esercizi, è stata adottata una griglia basata su tre dimensioni. Questo approccio è ispirato al lavoro di Gómez-Abajo et al., che propone una griglia di classificazione a tre livelli basata su struttura, ambiguità e vincoli impliciti [7]. Ogni esercizio viene quindi valutato su una scala da 1, inteso come valore basso, a 3, punteggio alto, per ciascun criterio. La Tabella 3.1 ne riassume i criteri.

Criterio	**Descrizione**	**Fonte**
Struttura e sintassi	Quantità e varietà di elementi nel diagramma UML (classi, attributi, associazioni, gerarchie, uso di costrutti avanzati come composizione o ereditarietà)	Almusawi et al., 2021 [1]
Ambiguità e astrazione	Presenza nella traccia testuale di concetti vaghi, impliciti o astratti (es. eventi, stati, entità non fisiche, relazioni sottintese)	Umar et al., 2021 [17]; Gomez et al.,2023 [7]
Vincoli logici impliciti	Necessità di inferire vincoli semantici non esplicitati (cardinalità, obbligatorietà, esclusività, dipendenze tra entità)	Almusawi et al., 2021 [1]
Leggibilità testuale	Complessità linguistica dell'enunciato, stimata tramite l'indice Gulpease per valutare l'accessibilità del testo	GULP, 1988

Tabella 3.1. Criteri adottati per la classificazione della difficoltà degli esercizi UML.

I vari criteri valutano quindi differenti proprietà dei singoli esercizi. Nello specifico:

- Struttura e sintassi: valuta la complessità visibile del modello UML, includendo il numero e la varietà di elementi (classi, attributi, relazioni, costrutti come composizione o ereditarietà). Un modello più strutturato richiede una gestione concettuale e sintattica più articolata.
- Ambiguità e astrazione: misura quanto la traccia testuale richieda interpretazioni collegate a concetti non esplicitamente definiti (es. eventi, stati, ruoli impliciti). Questo introduce un'incertezza cognitiva rilevante, legata alla necessità di inferire significati da elementi astratti o vagamente definiti.
- Vincoli logici impliciti: Rileva la presenza di vincoli semantici non dichiarati esplicitamente, come cardinalità non esplicitate, obbligatorietà condizionali, esclusività, dipendenze logiche.

Il punteggio per ogni criterio è stato assegnato sulla base delle caratteristiche qualitative della traccia testuale e del diagramma UML soluzione di riferimento. Una volta assegnato il singolo punteggio al singolo criterio, la somma dei punteggi parziali costituisce il cosìddetto Indice di Difficoltà Composito (IDC), con un valore minimo di 3 punti e massimo di 9.

Sulla base del valore dell'IDC, gli esercizi sono stati classificati nei seguenti livelli di difficoltà complessiva:

- **Facile**: IDC compreso tra 3 e 4
- **Medio**: IDC compreso tra 5 e 6
- **Difficile**: IDC compreso tra 7 e 9

Questa classificazione permette di stimare il livello di impegno cognitivo richiesto per interpretare e modellare ciascun esercizio, ed è coerente con approcci già consolidati in ambito educativo e di valutazione automatizzata della modellazione UML. In particolare, Almusawi et.al. (2021) hanno studiato la complessità strutturale degli esercizi [1], Umar et.al. (2021) hanno approfondito il ruolo dell'ambiguità semantica [17], Gómez-Abajo et.al. (2023) hanno proposto una griglia a tre dimensioni per classificare la difficoltà degli esercizi UML [7].

Oltre ai tre criteri principali di valutazione della difficoltà concettuale (struttura, ambiguità e vincoli), è stato introdotto un ulteriore indicatore di supporto: l'indice di leggibilità Gulpease, calcolato automaticamente per ciascun enunciato. Questo indice, sviluppato dal GULP (Gruppo Universitario Linguistico Pedagogico) nel 1988, è specificamente progettato per la lingua italiana e consente di stimare in modo oggettivo la complessità sintattica e lessicale di un testo. È particolarmente adatto in contesti didattici, poiché permette di valutare quanto la formulazione linguistica di un esercizio possa incidere sul carico cognitivo percepito da chi lo legge.

L'indice Gulpease si calcola tramite la seguente formula:

$$\text{Indice Gulpease} = 89 + \frac{300 \times F - 10 \times L}{P}$$

dove:

- F è il numero di frasi nel testo,
- L è il numero di lettere,
- P è il numero di parole.

Il risultato è un numero compreso tra 0 e 100: valori più alti indicano una maggiore leggibilità. In genere, testi con un indice superiore a 60 sono comprensibili almeno da studenti con licenza media, mentre valori inferiori a 40 segnalano una lettura complessa anche per chi ha una formazione universitaria.

3.3 Modelli LLM utilizzati

Per la sperimentazione sono stati selezionati quattro modelli di linguaggio di grandi dimensioni (LLM), scelti per rappresentare una varietà di approcci architetturali e livelli di accessibilità. Questa selezione include sia modelli proprietari di punta sia soluzioni open-source emergenti, al fine di valutare le capacità di LLM diversi nella generazione di diagrammi UML a partire da descrizioni testuali. In particolare:

- **Modelli emergenti e accessibili**: Qwen 3 e DeepSeek v3 sono stati rilasciati tra dicembre 2024 e gennaio 2025 come soluzioni *low-cost*, *open-source* e senza limiti d'uso, caratteristiche che ne favoriscono l'adozione in contesti di ricerca accademica.
- **Benchmark consolidati**: l'inclusione di ChatGPT e Gemini consente di confrontare l'efficacia dei nuovi modelli emergenti rispetto a modelli già affermati.
- **Utilizzo operativo per compiti accademici**: come nello studio di riferimento, i modelli saranno impiegati su task di *parafrasi* di abstract e di *question answering* su articoli scientifici, per misurarne la capacità operativa.

La selezione dei modelli analizzati (*ChatGPT*, *Gemini*, *Qwen 3* e *DeepSeek v3*) è eterogenea, in quanto ognuno di essi presenta caratteristiche architetturali diverse, strategie di training e modalità di interazione differenti. In particolare:

ChatGPT (GPT-4o) – OpenAI

- Accesso: tramite l'interfaccia web di ChatGPT (Plus) e API.
- Caratteristiche: supporta formati strutturati (es. JSON) e mantiene buone capacità di generalizzazione. È attualmente uno dei modelli con le migliori prestazioni su benchmark strutturati.

Gemini 2.5 Pro – Google DeepMind

- Accesso: disponibile tramite Google AI Studio e API Vertex AI.
- Caratteristiche: finestra di contesto fino a 1 milione di token, prestazioni competitive nella comprensione di contesto e generazione strutturata.

Qwen 3 – Alibaba Cloud

- Accesso: disponibile su Hugging Face o API.
- Caratteristiche: modello multilingue con buone capacità di ragionamento e struttura, pensato per adattarsi a vari task.

DeepSeek – DeepSeek AI

- Accesso: disponibile via API (compatibile con OpenAI).
- Caratteristiche: modello ottimizzato per generazione di codice e output formalizzati, con forte orientamento alla produttività.

La selezione di questi modelli consente di confrontare le prestazioni tra soluzioni proprietarie e open-source, nonché di valutare le capacità dei LLM nella generazione di diagrammi UML a partire da descrizioni testuali in linguaggio naturale.

3.4 Criteri di valutazione dei diagrammi UML generati

Dopo aver definito il dataset di esercizi e i modelli linguistici impiegati, è stato necessario individuare una metodologia analitica e rigorosa per valutare la qualità dei diagrammi UML generati. È stato adottato un approccio ispirato al framework teorico proposto da Lindland, Sindre e Solvberg [10], che definisce la qualità dei modelli concettuali in tre dimensioni principali: sintattica, semantica e pragmatica. Questo modello è stato ripreso anche in lavori recenti sulla valutazione automatica in ambito didattico, come in [7, 17], dove viene utilizzato per costruire rubriche computabili e replicabili.

Sulla base di tale riferimento, è stata costruita una griglia valutativa strutturata in sei criteri, ciascuno dei quali è stato associato a un punteggio numerico crescente da 0 a 2, secondo la seguente scala:

- 0 = criterio non soddisfatto
- 1 = criterio parzialmente soddisfatto
- 2 = criterio pienamente soddisfatto

L'utilizzo di una scala numerica con valori da 0 a 2 consente una valutazione più graduale e sfumata rispetto ai metodi binari, ed è compatibile con i sistemi di valutazione automatica proposti da Fauzan et al. (2021) e Gómez-Abajo et al. (2023) [4, 7].

Le descrizioni sintetiche dei criteri riportate in tabella sono state sviluppate in modo più esteso per chiarire meglio i parametri di valutazione adottati. Di seguito si riportano le definizioni dettagliate associate a ciascun criterio.

Tabella 3.2. Griglia di valutazione numerica per i diagrammi UML generati

Dimensione	**Criterio**	**Descrizione sintetica**
Qualità Sintattica	Conformità UML	Rispetto della sintassi UML (notazione, visibilità, molteplicità, correttezza simboli)
	Struttura JSON	Validità del file JSON secondo le specifiche della piattaforma Apollon
Qualità Semantica	Validità	Rappresentazione corretta dei concetti espressi nel requisito testuale
	Completezza	Presenza di tutti i concetti rilevanti contenuti o impliciti nella traccia
Qualità Pragmatica	Chiarezza	Organizzazione visiva ordinata e leggibile del diagramma
	Comprensibilità	Facilità di interpretazione da parte di un lettore esperto
Totale punteggio massimo		**12**

Si consideri, ad esempio, un diagramma che, pur rispettando pienamente la sintassi del linguaggio UML, omette la rappresentazione di una classe importante esplicitamente menzionata nella descrizione testuale. In tale circostanza, il diagramma potrebbe ottenere un punteggio massimo nel criterio di 'Conformità UML', ma essere valutato con il punteggio minimo nel criterio di 'Completezza', evidenziando la distinzione tra correttezza formale e copertura semantica.

Infatti, nello specifico, i vari criteri analizzano:

- **Conformità UML** Verifica che il diagramma rispetti le convenzioni sintattiche del linguaggio UML: nomi delle classi posizionati correttamente, uso appropriato di attributi, presenza di molteplicità dove necessario, tipo corretto delle relazioni (associazioni, composizioni, aggregazioni, generalizzazioni) e connessioni formalmente valide tra gli elementi. Errori in questa categoria includono relazioni non etichettate, molteplicità mancanti, o uso improprio dei simboli UML.

Questo criterio riflette l'importanza della correttezza formale dei modelli, un aspetto evidenziato anche da Fauzan et al. [4], dove la valutazione sintattica viene automatizzata attraverso l'utilizzo dello standard XMI (XML Metadata Interchange) per verificare la conformità ai linguaggi di modellazione UML. XMI è uno standard definito dall'Object Management Group (OMG) per la serializzazione dei modelli UML in formato XML. Tale rappresentazione consente la validazione automatica della sintassi del modello, il trasferimento tra diversi strumenti di modellazione, e l'integrazione con sistemi di valutazione automatica. XMI viene utilizzato per verificare automaticamente la correttezza formale dei diagrammi generati, assicurando la piena conformità alle specifiche UML. Sebbene il nostro approccio sia manuale, il principio guida resta lo stesso: un modello formalmente scorretto è inutilizzabile anche se semanticamente accurato.

- **Struttura JSON** Controlla che il file prodotto sia un JSON valido e compatibile con il formato richiesto dalla piattaforma Apollon o UML Modeler. Ciò implica una formattazione corretta, quindi la presenza di tutte le sezioni obbligatorie (es. "elements", "relations", "size"), il rispetto della struttura ad albero e l'assenza di errori della formattazione stessa. Un punteggio basso viene attribuito in caso di errori di sintassi JSON, errori di formattazione (inserimento degli elementi quali classi o attributi nella prima dichiarazione di "elements" che deve essere dichiaratamente vuota) elementi mancanti, oppure altri errori che comportano un'impossibilità della visualizzazione del file durante l'importazione su Apollon o UML Modeler (una piattaforma didattica sviluppata internamente dal gruppo SoftEng e usata per i suoi corsi).

  Infatti, la validità del JSON è stata testata caricando ciascun file nella piattaforma Apollon e osservandone l'output. Questo criterio è fondamentale per garantire l'eseguibilità del diagramma nel contesto sperimentale, come analogamente avviene nel framework Moodle [7].

- **Validità** Valuta se gli elementi presenti nel diagramma riflettono correttamente il contenuto semantico della descrizione testuale. Questo include la correttezza dei nomi, la rappresentazione coerente delle relazioni tra entità, e la coerenza tra concetti descritti nel testo e quelli modellati. Sono penalizzate l'introduzione di entità arbitrarie o non dichiarate esplicitamente nel testo, relazioni inesatte o interpretazioni errate dei vincoli logici del dominio.

  La validità semantica può essere valutata confrontando i modelli prodotti con una soluzione di riferimento, considerando anche sinonimi e variazioni ammissibili. Sebbene nella nostra analisi il confronto sia avvenuto manualmente, il principio di riferimento resta quello della corrispondenza sostanziale tra contenuto testuale e modellazione concettuale.

- **Chiarezza** Considera l'organizzazione e il layout visivo, quindi la leggibilità del diagramma. Un modello chiaro presenta un layout equilibrato, con relazioni ben distribuite, assenza di incroci inutili di associazioni e un uso armonico dello spazio.

I punteggi più bassi vengono attribuiti in caso di disordine grafico, eccessiva densità locale di elementi, etichette sovrapposte o posizionamento caotico.

La chiarezza visiva, pur presentando una componente soggettiva, è riconosciuta come un criterio rilevante anche da Almusawi et al. [1], che la include tra i parametri valutativi sotto la voce di "Visual Clarity". Essa contribuisce a facilitare la revisione manuale dei diagrammi e ne migliora la comprensibilità complessiva, anche in assenza di errori semantici.

- **Comprensibilità** Misura quanto il diagramma può essere compreso da un lettore esperto nel dominio, senza ambiguità interpretative. Un diagramma è comprensibile quando i concetti sono espressi in modo intuitivo, i nomi sono coerenti con il lessico del dominio e le relazioni risultano semanticamente esplicite. Viene penalizzata la presenza di nomi generici o ambigui, relazioni non etichettate, o elementi non chiari.

  Romero et al. [14] sottolineano che la comprensibilità è una qualità indispensabile per l'adozione di strumenti di correzione automatica: anche modelli formalmente corretti possono risultare inutilizzabili se l'utente non riesce a coglierne il significato. Per questo, il criterio è stato valutato da due correttori indipendenti secondo criteri condivisi.

Il punteggio complessivo ottenuto su scala 0–12 consente di classificare la qualità del diagramma secondo la seguente interpretazione:

- 10–12 punti: *diagramma eccellente*
- 7–9 punti: *diagramma buono, con minime imperfezioni*
- 4–6 punti: *diagramma sufficiente ma migliorabile*
- 0–3 punti: *diagramma inadeguato*

Le soglie sono state definite in modo da riflettere il grado di completezza e correttezza osservato nei modelli. La classificazione è stata effettuata manualmente secondo la griglia, senza l'uso di metriche automatiche, indi per cui è necessario sottolineare che, nella valutazione, si inserisce un grado di soggettività non trascurabile, seppur l'obiettivo primario rimanga pur sempre quello di adottare un approccio quantitativo e oggettivo.

Sebbene il criterio relativo alla struttura del file JSON non sia presente nel modello teorico originale preso d'ispirazione, è stato introdotto in questo contesto per verificare la compatibilità tecnica con il tool Apollon, utilizzato nella sperimentazione.

Tale sistema consente non solo un'analisi comparativa e sistematica, ma soprattutto replicabile della performance dei diversi LLM, offrendo metriche sia qualitative sia quantitative. L'approccio adottato, oltre a garantire consistenza nella valutazione, è pensato per essere esteso anche in contesti educativi strutturati, favorendo la riproducibilità e l'integrazione con sistemi di correzione assistita [4, 14].

3.5 Strategia di Prompting e Specifica del Prompt

La qualità e la validità dei diagrammi UML generati dai Large Language Models (LLM) dipendono in modo cruciale dalla formulazione del prompt, ovvero l'input di partenza che viene fornito agli stessi LLM. Per questo motivo, una parte significativa della metodologia è stata dedicata alla progettazione, validazione e progressivo raffinamento della strategia di prompting, con l'obiettivo di ridurre gli errori sintattici, garantire la piena compatibilità con lo strumento Apollon e massimizzare la completezza e la chiarezza dei modelli generati.

Prompt iniziale: caratteristiche e limiti

Il lavoro è stato inizialmente condotto utilizzando un prompt iniziale già molto dettagliato, denominato *Prompt 1* qui di seguito presentato:

Listing 3.1. Prompt finale utilizzato

```
PROMPT 1:

Sei un professore di ingegneria del software esperto in
   modellazione UML e nel formato JSON utilizzato da Apollon (
   https://apollon.ase.in.tum.de).

Il tuo compito è generare un diagramma UML delle classi in formato
    JSON pienamente compatibile con Apollon, a partire dalla
   descrizione testuale di un sistema.

Devi rispettare scrupolosamente le seguenti istruzioni.

La struttura JSON iniziale deve essere esattamente la seguente:
{
  "version": "3.0.0",
  "type": "ClassDiagram",
  "size": { "width": 1600, "height": 780 },
  "interactive": { "elements": {}, "relationships": {} },
  "assessments": {},
  ...
}
Questo header deve essere presente esattamente come mostrato sopra
   .

Importante: dopo questo header, il JSON deve continuare con una
   dichiarazione completa di:
- "elements": { ... }
- "relationships": { ... }

Queste sezioni devono essere scritte fuori e separatamente dal
   blocco "interactive".
```

```
Non inserire **nessuna classe, attributo o relazione** all'interno
    di "interactive".
Quel blocco deve restare **vuoto**, come richiesto dal formato
   Apollon per garantire la compatibilità.
Gli oggetti definiti in "elements" e "relationships" vanno scritti
    fuori dal blocco "interactive".

Ogni classe (con type "Class") deve includere:
un campo "id" univoco;
un campo "name";
una lista "attributes" (con riferimenti agli ID degli attributi);
una lista "methods" (anche vuota se non previsti dal testo);
un campo "bounds" con coordinate non sovrapposte e dimensioni
   coerenti.

Ogni classe deve essere rappresentata come oggetto all'interno del
    blocco "elements",
identificato tramite una chiave univoca (preferibilmente in
   formato UUIDv4).

Il contenuto dell'oggetto classe deve includere i seguenti campi:
- "id"
- "name"
- "type": "Class"
- "owner"
- "bounds"
- "attributes"
- "methods": [] (anche se vuoto)


Ogni attributo e metodo deve essere definito come oggetto
   indipendente con:
type: "ClassAttribute" o "ClassMethod";
owner: ID della classe a cui appartiene;
bounds corretti (posizionati sotto la classe).

Ogni attributo deve essere posizionato immediatamente dopo la
   classe a cui appartiene,
e deve contenere i seguenti campi:
- "id"
- "name": nel formato "+ nomeAttributo : Tipo" (es. "+ nome :
   string")
- "type": "ClassAttribute"
- "owner": ID della classe
- "bounds": coordinate coerenti, posizionate sotto la classe


Gli attributi devono essere di tipo semplice: string, int, float o
    date.
```

```
Non è ammesso che un attributo sia una classe (no composite types)
    .
Non sono permesse collezioni (liste, set, mappe): eventuali
    molteplicità devono essere modellate tramite relazioni
    esplicite.

Le relazioni (come "ClassBidirectional", "ClassComposition", ecc.)
     devono includere:
"source" e "target" con ID delle classi, molteplicità e direzione;
"path" con coordinate per il tracciato;
"bounds" compatibili con la disposizione grafica;
"isManuallyLayouted": false.
Ogni relazione (inserita nel blocco "relationships") deve
    includere i seguenti campi:
- "id"
- "type"
- "name"
- "bounds"
- "path"
- "source": con "element", "direction", "multiplicity", e
    opzionalmente "role"
- "target": con "element", "direction", "multiplicity", e
    opzionalmente "role"
- "isManuallyLayouted": false


Se il testo implica una relazione di specializzazione tra entità (
    es. tipi di utente o entità), modellala come ClassBidirectional
     con name: "is-a" e molteplicità vuote.

Se il testo suggerisce un legame di proprietà o parte-tutto (es. "
    una biblioteca ha molti libri"), usa ClassComposition o
    ClassAggregation.

Nelle aggregazioni, la classe contenitore (aggregante) deve avere
    il rombo (cioè essere il source se appropriato).

Le molteplicità devono essere realistiche: 1, 0..1, 0..n, 1..n, in
     base al dominio.



Devi sempre includere nomi, tipi, molteplicità e direzioni delle
    relazioni secondo quanto espresso nel testo.

Ogni classe deve essere correttamente collegata alle altre se
    previsto dalla descrizione (per esempio: composizione tra Conto
     e VoceOrdine, associazione tra Prenotazione e Cliente, ecc.).
```

```
Se una classe svolge ruoli diversi in più relazioni verso la
   stessa altra classe, modella ogni relazione separatamente e
   assegna nomi distintivi.
Quando necessario, usa il campo role in source e target per
   disambiguare i ruoli.
Se una relazione implica attributi propri (es. data, quantità),
   modellala tramite una classe associativa con i relativi
   attributi.

Se nel testo sono indicati dei metodi, devi inserirli come oggetti
    separati con type "ClassMethod".

In particolare, includi:
- almeno una generalizzazione se il testo implica sottotipi
- almeno una composizione o aggregazione se ci sono relazioni
   parte-tutto
- almeno 3 attributi realistici per ogni classe principale
- classi astratte, enumerazioni o classi associative se utili a
   chiarire concetti ambigui

Assicurati che i 'path' di ogni relazione abbiano larghezza > 0 e
   rientrino nelle dimensioni indicate in    size    , altrimenti
   regola le coordinate.

Se logicamente implicate inserisci classi astratte, associazioni
   mancanti, generalizzazioni o enumerazioni se logicamente
   implicate utili a chiarire il sistema.
Il tuo output deve essere un JSON valido, completo e funzionante
   su Apollon, senza commenti o testo aggiuntivo.

Usa nomi chiari, coerenti e grammaticalmente corretti per classi,
   attributi e relazioni.

Evita etichette generiche come "thing", "elemento", "info", "
   scrive".
Quando utile alla chiarezza, preferisci la forma passiva per i
   nomi delle relazioni
(es. "è guidato da" invece di "guida").


Alla fine del prompt troverai la descrizione testuale del sistema
   da modellare.

Esempi da seguire:

Esempio 1     testo:
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.

Esempio 1     JSON:
(vedi file: diagram (1).json     fornito come riferimento)

Esempio 2     testo:
Descrivere con un diagramma a stati il comportamento di un
   orologio digitale dotato di display per la visualizzazione e di
    due bottoni per l'impostazione dell'orario.
Il primo bottone definisce la modalità di visualizzazione (normale
   , impostazione ora, impostazione minuti),
mentre il secondo viene utilizzato in modalità impostazione per
   incrementare l'ora o i minuti correnti.
Dalla modalità di visualizzazione normale (nella quale viene
   visualizzato l'orario corrente)
premendo una prima volta il bottone che seleziona la modalità di
   visualizzazione si passa alla modalità di impostazione dell'ora
   ,
premendo una seconda volta si passa all'impostazione dei minuti,
premendolo una terza volta si torna alla modalità di
   visualizzazione normale.

Esempio 2     JSON:
(vedi file: diagram (2).json     fornito come riferimento)

Ecco a te la descrizione testuale del sistema da modellare:
```

Questo prompt, sebbene concepito per LLM generalisti, presentava un livello di specificità elevato, includendo:

- la struttura esatta dell'header JSON (versione, tipo, dimensioni);
- l'obbligo di posizionare i blocchi elements e relationships all'esterno della sezione `interactive`;

- la definizione formale di ciascun oggetto (classi, attributi, metodi, relazioni), con i campi obbligatori da includere;
- alcuni vincoli su nomenclatura, sintassi, tipo degli attributi e strutture relazionali corrette come composizioni, aggregazioni e generalizzazioni;
- esempi completi di training (testo + JSON) per facilitare l'inferenza di struttura da parte del modello.

Pur non essendo generico, il prompt iniziale ha mostrato alcune debolezze operative: modelli LLM diversi da ChatGPT (come Qwen, Gemini e DeepSeek) faticavano a produrre output compatibili con Apollon. I problemi principali risiedevano nell'omissione di blocchi fondamentali, nell'inserimento errato di elementi all'interno di `interactive`, o nella violazione di vincoli su campi come `owner`, `bounds` o `path`. Tali criticità, pur non dovute a errori concettuali nella formulazione del prompt, hanno evidenziato la necessità di una maggiore modularità, ridondanza e chiarezza espositiva per facilitare la comprensione da parte del modello.

3.5.1 Evoluzione incrementale e progettazione del prompt finale

In risposta agli errori rilevati, è stato avviato un processo iterativo di raffinamento del prompt, in cui la strategia di prompting è stata progressivamente migliorata. Questa fase si è basata su una serie di esercizi di training, selezionati tra tracce autentiche provenienti da corsi universitari italiani di Ingegneria del Software e Modellazione concettuale. Dopo ogni errore di generazione, il prompt veniva aggiornato per esplicitare il vincolo violato: spesso era lo stesso modello, attraverso il proprio output parzialmente corretto, a fornire indicazioni implicite su come correggere il prompt per evitare il ripetersi dell'errore.

Questo processo ha portato allo sviluppo di un *prompt finale*, qui di seguito riportato:

Listing 3.2. Prompt finale utilizzato

```
PROMPT FINALE

Sei un professore di ingegneria del software esperto in
   modellazione UML e nel formato JSON utilizzato da Apollon (
   https://apollon.ase.in.tum.de).
Il tuo compito è generare un diagramma UML delle classi in formato
    JSON pienamente compatibile con Apollon, a partire dalla
   descrizione testuale di un sistema.
Devi rispettare scrupolosamente le seguenti istruzioni.

La struttura JSON iniziale deve essere esattamente la seguente:
{
  "version": "3.0.0",
  "type": "ClassDiagram",
  "size": { "width": 1600, "height": 780 },
  "interactive": { "elements": {}, "relationships": {} },
  "assessments": {},
```

```
  ...
}
Questo header deve essere presente esattamente come mostrato sopra
   .
Il blocco "elements" e "relationships" devono trovarsi fuori da "
   interactive" e non devono contenere altri oggetti nidificati al
    loro interno.
Ogni attributo e metodo deve essere definito come oggetto
   indipendente all'interno del blocco "elements" con i seguenti
   campi:

"id": UUID unico (es. "a1b2c3d4-e5f6-7890-g1h2-i3j4k5l6m7n8 ),
"name": nel formato "+ nome : tipo" per gli attributi o "nome(
   parametri): tipo" per i metodi,
"type": "ClassAttribute" o "ClassMethod ,
"owner": ID della classe a cui appartiene
"bounds": coordinate positive, sotto la classe

Importante: tutti i campi come "id", "name", "type", "owner", "
   bounds" devono essere presenti dove richiesti.

Importante: dopo questo header, il JSON deve continuare con una
   dichiarazione completa di:
- "elements": { ... }
- "relationships": { ... }

Queste sezioni devono essere scritte fuori e separatamente dal
   blocco "interactive".
Non inserire **nessuna classe, attributo o relazione** all'interno
    di "interactive".
Quel blocco deve restare **vuoto**, come richiesto dal formato
   Apollon per garantire la compatibilità.
Gli oggetti definiti in "elements" e "relationships" vanno scritti
    fuori dal blocco "interactive".

Ogni classe (con type "Class") deve includere:
un campo "id" univoco;
un campo "name";
una lista "attributes" (con riferimenti agli ID degli attributi);
una lista "methods" (anche vuota se non previsti dal testo);
un campo "bounds" con coordinate non sovrapposte e dimensioni
   coerenti.

Ogni classe deve essere rappresentata come oggetto all'interno del
    blocco "elements",
identificato tramite una chiave univoca (preferibilmente in
   formato UUIDv4).

Il contenuto dell'oggetto classe deve includere i seguenti campi:
```

```
- "id"
- "name"
- "type": "Class"
- "owner"
- "bounds"
- "attributes"
- "methods": [] (anche se vuoto)


Ogni attributo e metodo deve essere definito come oggetto
   indipendente con:
type: "ClassAttribute" o "ClassMethod";
owner: ID della classe a cui appartiene;
bounds corretti (posizionati sotto la classe).

Ogni attributo deve essere posizionato immediatamente dopo la
   classe a cui appartiene,
e deve contenere i seguenti campi:
- "id"
- "name": nel formato "+ nomeAttributo : Tipo" (es. "+ nome :
   string")
- "type": "ClassAttribute"
- "owner": ID della classe
- "bounds": coordinate coerenti, posizionate sotto la classe


Gli attributi devono essere di tipo semplice: string, int, float o
    date.
Non è ammesso che un attributo sia una classe (no composite types)
   .
Non sono permesse collezioni (liste, set, mappe): eventuali
   molteplicità devono essere modellate tramite relazioni
   esplicite.

Le relazioni (come "ClassBidirectional", "ClassComposition", ecc.)
    devono includere:
"source" e "target" con ID delle classi, molteplicità e direzione;
"path" con coordinate per il tracciato;
"bounds" compatibili con la disposizione grafica;
"isManuallyLayouted": false.
Ogni relazione (inserita nel blocco "relationships") deve
   includere i seguenti campi:
- "id"
- "type"
- "name"
- "bounds"
- "path"
- "source": con "element", "direction", "multiplicity", e
   opzionalmente "role"
```

```
- "target": con "element", "direction", "multiplicity", e
    opzionalmente "role"
- "isManuallyLayouted": false

Le relazioni devono essere definite nel blocco "relationships" e
    includere:

"source" e "target" con "element", "direction", "multiplicity", e
    opzionalmente " r o l e
"path": array di punti {x, y} che riflette graficamente la
    posizione delle classi
"isManuallyLayouted": true
"bounds": dimensioni e posizione della relazione nello spazio del
    diagramma

La direzione della relazione deve riflettere semanticamente il
    ruolo svolto dalle classi:

* Usa "Left", "Right", "Up", "Down" in base alla posizione
    reciproca delle classi.
* La molteplicità deve essere sempre specificata coerentemente:
     * "1"     esattamente uno
     * "0..1"     zero o uno
     * "1..n"     uno o più
     * "*"     molti (usa solo se necessario)


Se il testo implica una relazione di specializzazione tra entità (
    es. tipi di utente o entità), modellala come ClassBidirectional
     con name: "is-a" e molteplicità vuote.

Gli ID devono essere stringhe casualiche, preferibilmente nel
    formato UUIDv4 (es. "a1b2c3d4-e5f6-7890-g1h2-i3j4k5l6m7n8"), ma
     anche formati semplici come "rel1", "attr1" sono accettabili
    se non si ha ambiguità.

I metodi senza implementazione devono comunque avere un "name"
    valido (es. "toString(): String"), mai vuoto.

Se il testo suggerisce un legame di proprietà o parte-tutto (es. "
    una biblioteca ha molti libri"), usa ClassComposition o
    ClassAggregation.

Nelle aggregazioni, la classe contenitore (aggregante) deve avere
    il rombo (cioè essere il source se appropriato).

Le molteplicità devono essere realistiche: 1, 0..1, 0..n, 1..n, in
     base al dominio.
```

```
Tutti gli elementi devono avere "bounds" con coordinate positive ,
    mai negative, e mai sovrapposti . Evita coordinate come x =
   -600. Usa valori compresi tra 0 e 1600 per x, e tra 0 e 780 per
    y.

* Per relazioni di generalizzazione (ereditarietà) usa "
   ClassBidirectional" con "name": "is-a" e molteplicità vuote.
* Per relazioni di composizione (parte-tutto) usa "
   ClassComposition" con rombo rivolto verso la classe contenitore
   .
* Per aggregazioni usa "ClassAggregation".


Devi sempre includere nomi, tipi, molteplicità e direzioni delle
   relazioni secondo quanto espresso nel testo.

Ogni classe deve essere correttamente collegata alle altre se
   previsto dalla descrizione (per esempio: composizione tra Conto
    e VoceOrdine, associazione tra Prenotazione e Cliente, ecc.).
Se una classe svolge ruoli diversi in più relazioni verso la
   stessa altra classe, modella ogni relazione separatamente e
   assegna nomi distintivi.
Quando necessario, usa il campo role in source e target per
   disambiguare i ruoli.
Se una relazione implica attributi propri (es. data, quantità),
   modellala tramite una classe associativa con i relativi
   attributi.


Se nel testo sono indicati dei metodi, devi inserirli come oggetti
    separati con type "ClassMethod".

In particolare, includi:
- almeno una generalizzazione se il testo implica sottotipi
- almeno una composizione o aggregazione se ci sono relazioni
   parte-tutto
- almeno 3 attributi realistici per ogni classe principale
- classi astratte, enumerazioni o classi associative se utili a
   chiarire concetti ambigui

Assicurati che i    path    di ogni relazione abbiano larghezza >
   0 e rientrino nelle dimensioni indicate in    size   ,
   altrimenti regola le coordinate.

Se logicamente implicate inserisci classi astratte, associazioni
   mancanti, generalizzazioni o enumerazioni se logicamente
   implicate utili a chiarire il sistema.
Il tuo output deve essere un JSON valido, completo e funzionante
   su Apollon, senza commenti o testo aggiuntivo.
```

```
Usa nomi chiari, coerenti e grammaticalmente corretti per classi,
   attributi e relazioni.

Evita etichette generiche come "thing", "elemento", "info", "
   scrive".
Quando utile alla chiarezza, preferisci la forma passiva per i
   nomi delle relazioni
(es. "è guidato da" invece di "guida").

Tutti i campi obbligatori nei file esempio devono essere presenti.
    In particolare:

* "owner" in attributi/metodi
* "type" corretto
* "bounds" in tutti gli elementi
* "path" in tutte le relazioni



Alla fine del prompt troverai la descrizione testuale del sistema
   da modellare.

Esempi da seguire:

Esempio 1, testo:
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.

Esempio 1, JSON:
(vedi file: diagram (1).txt     fornito come riferimento)

Esempio 2     testo:
```

```
Descrivere con un diagramma a stati il comportamento di un
   orologio digitale dotato di display per la visualizzazione e di
    due bottoni per l'impostazione dell'orario.
Il primo bottone definisce la modalità di visualizzazione (normale
   , impostazione ora, impostazione minuti),
mentre il secondo viene utilizzato in modalità impostazione per
   incrementare l'ora o i minuti correnti.
Dalla modalità di visualizzazione normale (nella quale viene
   visualizzato l'orario corrente)
premendo una prima volta il bottone che seleziona la modalità di
   visualizzazione si passa alla modalità di impostazione dell'ora
   ,
premendo una seconda volta si passa all'impostazione dei minuti,
premendolo una terza volta si torna alla modalità di
   visualizzazione normale.

Esempio 2     JSON:
(vedi file: diagram (2).txt     fornito come riferimento)

Ecco a te la descrizione testuale del sistema da modellare:
```

Il prompt finale è risultato notevolmente più articolato, caratterizzato da:

- una struttura gerarchica chiaramente segmentata (classi, attributi, metodi, relazioni);
- una maggiore ridondanza sintattica e semantica nei comandi, per ridurre l'ambiguità;
- l'inclusione sistematica di istruzioni grafiche su `bounds`, `path` e `isManuallyLayouted`;
- l'esplicitazione di trattamenti per casi speciali (classi astratte, generalizzazioni, enumerazioni, classi associative);
- linee guida stilistiche per l'uso coerente e grammaticale dei nomi (es. forma passiva nelle relazioni);
- istruzioni chiare per disambiguare ruoli semantici e identificare quando inserire molteplicità o proprietà.

Questo prompt è stato progettato per essere il più possibile deterministico, validabile e compatibile con Apollon, fungendo da interfaccia dettagliata tra testo e output grafico.

3.5.2 Osservazioni conclusive

L'evoluzione del prompt non si è limitata a un raffinamento sintattico, ma ha rappresentato un elemento chiave del processo metodologico. Infatti, la capacità dei LLM di generare diagrammi UML validi è stata fortemente influenzata dalla precisione, chiarezza e struttura del prompt utilizzato. Il processo iterativo di correzione, supportato dagli

stessi modelli attraverso feedback impliciti (come l'indicazione dell'assenza di campi richiesti o l'errata collocazione di oggetti), ha mostrato come il prompting possa essere affinato in sinergia tra progettazione umana e risposta automatica. Questo approccio ha permesso di costruire una strategia robusta e adattabile, capace di massimizzare l'affidabilità del risultato a prescindere dal modello impiegato.

Tecniche di Prompt Engineering applicate nel Prompt

Il *Prompt*, risultato del processo iterativo di raffinamento, ha integrato in modo sistematico diverse tecniche avanzate di prompt engineering:

- **Role prompting**: il prompt inizia assegnando esplicitamente al modello il ruolo di "professore di ingegneria del software esperto in modellazione UML", al fine di stimolare una produzione più specializzata e disciplinata dell'output.

- **Istruzioni vincolanti e modulari**: ogni vincolo strutturale e sintattico (es. posizione di "elements" e "relationships", obbligatorietà della presenza di "owner", "bounds" e "path") è riportato in forma diretta, indipendente e ripetitiva, così da favorire la memorizzazione da parte del modello anche in presenza di contesti lunghi.

- **Validazione implicita tramite esempi**: il prompt include due esempi completi (testo + JSON), pienamente compatibili con Apollon, che servono come guida implicita, quindi training, al comportamento atteso, facilitando l'allineamento tra descrizione testuale e struttura del diagramma.

- **Specificità semantica**: sono presenti istruzioni per la corretta modellazione di concetti avanzati (composizioni, aggregazioni, generalizzazioni, classi associative), con raccomandazioni chiare sull'uso delle relazioni "ClassComposition", "ClassAggregation" e "ClassBidirectional" e relazioni di gerarchia o generalizzazioni, denominate con nomenclatura "is-a".

- **Stilizzazione linguistica**: il prompt fornisce linee guida esplicite per evitare nomi generici o ambigui, preferendo strutture nominali esplicite e relazioni denominate in forma passiva per migliorare la leggibilità semantica del diagramma.

- **Controllo del layout grafico**: è richiesto l'uso di coordinate positive, coerenti e non sovrapposte per tutti gli elementi (classi, attributi, metodi, relazioni), con attenzione al corretto posizionamento dei "path" delle relazioni.

- **Inclusione obbligatoria di elementi strutturali**: ogni prompt impone la presenza di almeno una generalizzazione, una composizione o aggregazione, e tre attributi per ciascuna classe principale, forzando l'emersione di pattern UML ricorrenti.

Tali tecniche, combinate in modo coerente, hanno contribuito in modo decisivo alla generazione di output deterministici, strutturalmente validi e semanticamente completi, anche da parte di modelli precedentemente non in grado di soddisfare i vincoli del task.

Aspetto	**Prompt Iniziale**	**Prompt Finale**
Ruolo assegnato al modello	Assistente esperto in UML	Professore esperto in ingegneria del software e formato Apollon
Struttura dell'output JSON	Specificata, ma non sempre rispettata	Specificata con precisione, obbligatoria e ripetuta
Separazione elementi	A volte ambigua (interactive mal usato)	Modularità chiara: "elements" e "relationships" fuori da "interactive"
Esempi few-shot	Presenti ma parziali	Due esempi completi (testo + JSON), validati su Apollon
Vincoli su attributi e metodi	Parziali o impliciti	Rigorosi: owner, bounds, type sempre obbligatori
Indicazioni semantiche avanzate	Solo accennate (generalizzazioni, composizioni)	Dettagliate: vincoli espliciti su is-a, aggregazioni, ruoli
Gestione delle ambiguità	Lasciano spazio a interpretazioni	Linee guida esplicite (naming, layout, molteplicità)
Controllo del layout grafico	Non trattato	Coordinate, dimensioni e path obbligatori e coerenti
Stilizzazione linguistica	Nessuna indicazione	Vietati nomi generici, preferita forma passiva per relazioni
Validazione implicita	Non sempre efficace	Compatibilità garantita con Apollon

Tabella 3.3. Confronto tra Prompt Iniziale (ChatGPT) e Prompt Finale (Qwen)

3.6 Formato dell'output: JSON

Il formato scelto per la rappresentazione dei diagrammi è stato quello JSON compatibile con il tool Apollon (https://apollon.ase.in.tum.de), una piattaforma accademica per la generazione e validazione visuale di diagrammi UML. Tale formato consente:

- un parsing automatizzato del contenuto generato;
- la visualizzazione diretta del diagramma senza post-processing;
- il confronto strutturale con soluzioni di riferimento.

La compatibilità con Apollon ha costituito un criterio fondamentale nella progettazione dei prompt e nella selezione dei dati per la valutazione.

3.7 Piano di analisi dei risultati

Al termine della generazione automatica dei diagrammi UML da parte dei modelli LLM selezionati, è stata condotta un'analisi quantitativa e qualitativa dei risultati, secondo il framework metodologico descritto nei paragrafi precedenti.

In particolare, il Capitolo 4 riporterà:

- il confronto tra i punteggi medi ottenuti da ciascun modello nelle sei dimensioni di valutazione (conformità UML, struttura JSON, validità, completezza, chiarezza e comprensibilità);
- l'analisi della variabilità interna dei risultati attraverso l'uso di indicatori statistici (deviazione standard, coefficiente di variazione);
- l'indagine sull'impatto della difficoltà concettuale degli esercizi (IDC) sulle performance dei modelli;
- lo studio delle tipologie di errore più ricorrenti, con classificazione per categoria e casi studio selezionati;
- la valutazione comparativa delle prestazioni dei quattro LLM in funzione del prompt utilizzato;
- una discussione finale integrata dei risultati, utile a sintetizzare punti di forza e debolezza dei modelli.

Tutte le analisi sono condotte facendo esplicito riferimento ai criteri e alle strategie di valutazione illustrati in questo capitolo, al fine di garantire coerenza metodologica e riproducibilità dello studio.

Capitolo 4

Risultati Sperimentali

Come anticipato nel Capitolo 3, la valutazione dei diagrammi UML generati è stata condotta applicando una griglia a sei criteri, articolata secondo le tre dimensioni della qualità del modello (sintattica, semantica, pragmatica). Ogni diagramma è stato prodotto a partire da una descrizione testuale in linguaggio naturale, sottoposta a ciascun modello tramite il medesimo prompt finale.

L'obiettivo di questa fase è analizzare le prestazioni dei diversi modelli,analizzandone la qualità media dei risultati, la loro stabilità, coerenza e robustezza in presenza di variabili come la difficoltà degli esercizi.

I dati riportati derivano dalla tabella di valutazione prodotta manualmente, seguendo le linee guida definite nella Sezione 3.4 e applicate secondo la procedura descritta nella Sezione 3.6.

L'obiettivo di questo capitolo è rispondere alle seguenti domande di ricerca, attraverso un'analisi quantitativa e qualitativa delle prestazioni dei modelli:

1. In che misura i LLM sono in grado di generare automaticamente diagrammi UML formalmente corretti e semanticamente completi?

2. Come varia la qualità dei diagrammi generati al variare della difficoltà concettuale degli esercizi?

3. In che modo le strategie di prompting influenzano la qualità dell'output generato?

4. Quali sono gli errori più frequenti nei diagrammi UML generati dai modelli e come si distribuiscono in base alla difficoltà del compito?

I risultati ottenuti sono organizzati per categoria di analisi (valutazione media, difficoltà, errori, effetto del prompting, casi studio).

4.1 Descrizione generale dei risultati

4.1.1 Punteggi medi per modello

Come primo confronto generale, la Tabella 4.1 mostra il punteggio medio ottenuto da ciascun modello per ognuno dei sei criteri valutativi adottati (conformità UML, struttura JSON, validità, completezza, chiarezza, comprensibilità).

Tabella 4.1. Punteggio medio per criterio per ciascun modello

Modello	UML	JSON	Validità	Completezza	Chiarezza	Comprensibilità
ChatGPT	2.00	2.00	2.00	2.00	2.00	2.00
DeepSeek	2.00	2.00	1.80	1.67	2.00	2.00
Gemini	2.00	2.00	1.87	1.87	2.00	2.00
Qwen	1.20	1.33	1.33	1.00	1.47	1.27

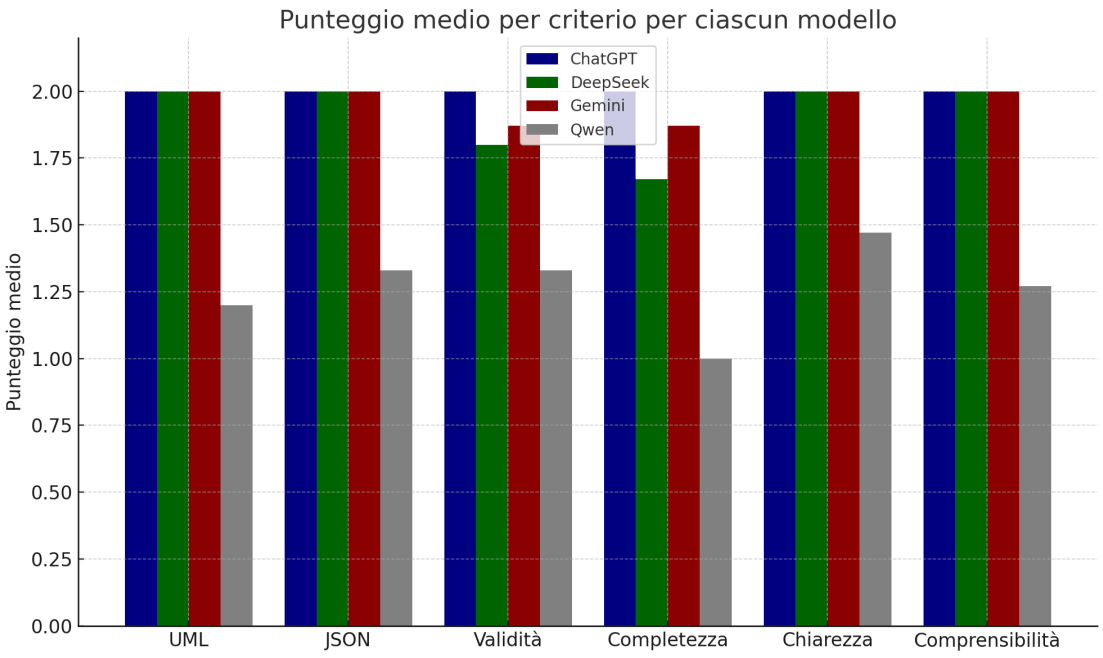


Figura 4.1. Risultati Preliminari di un Primo Confronto Generale tra LLM diversi

Dal grafico riassuntivo presente in Figura 4.1 emergono differenze significative tra i modelli analizzati. *ChatGPT* si distingue nettamente per la coerenza e l'elevata qualità dei risultati, ottenendo il punteggio massimo in tutte le dimensioni. *DeepSeek* e *Gemini* mostrano performance molto simili, con leggere flessioni nella validità e completezza, pur

mantenendo un'eccellente aderenza sintattica e chiarezza visiva. Al contrario, *Qwen* si colloca su un livello qualitativo inferiore, con punteggi più bassi e irregolari in tutte le categorie, soprattutto per quanto riguarda la completezza e la comprensibilità. Questi risultati preliminari suggeriscono che l'efficacia nella generazione dei diagrammi varia in modo marcato a seconda del modello, e richiede un'analisi più dettagliata nei paragrafi successivi.

4.1.2 Valutazione specifica dei singoli modelli

La valutazione è stata svolta sui sei criteri definiti nella Metodologia dello studio. Per ogni criterio è stato assegnato un punteggio compreso tra 0 e 2, secondo la seguente logica:

- **Conformità UML**:
  - 0 punti: presenza di errori gravi nella sintassi UML (simboli errati, uso improprio di associazioni, visibilità o relazioni errate).
  - 1 punto: sintassi parzialmente corretta, ma con errori non gravi o omissioni.
  - 2 punti: sintassi UML pienamente rispettata in termini di notazione, simboli, molteplicità e visibilità.
- **Struttura JSON**:
  - 0 punti: file non valido o non compatibile con la piattaforma Apollon.
  - 1 punto: file tecnicamente valido ma con struttura incompleta o parzialmente errata (es. mancanza di layout, coordinate incoerenti).
  - 2 punti: file pienamente conforme alle specifiche JSON richieste da Apollon, correttamente formattato e completo.
- **Validità**:
  - 0 punti: rappresentazione errata o incoerente dei concetti espressi nella traccia.
  - 1 punto: rappresentazione parzialmente corretta, ma con interpretazioni discutibili o relazioni mal posizionate.
  - 2 punti: rappresentazione pienamente coerente dei concetti espressi nel requisito testuale.
- **Completezza**:
  - 0 punti: mancano numerosi concetti rilevanti, sia espliciti che impliciti (es. classi o molteplicità esplicitate nel testo mancanti).
  - 1 punto: rappresentati solo alcuni dei concetti richiesti, oppure relazioni incomplete.

  - 2 punti: tutti i concetti chiave presenti nella traccia sono stati inclusi in modo corretto.

- **Chiarezza**:
  - 0 punti: diagramma disorganizzato, con elementi sovrapposti o distribuiti in modo caotico.
  - 1 punto: diagramma leggibile ma con problemi di allineamento, spaziatura o disposizione non ottimale.
  - 2 punti: organizzazione visiva chiara e ordinata, che facilita la lettura e l'analisi del modello.

- **Comprensibilità**:
  - 0 punti: difficile interpretazione anche per un lettore esperto, per ambiguità o incoerenze.
  - 1 punto: interpretazione possibile ma con sforzo, a causa di nomi vaghi o relazioni ambigue.
  - 2 punti: il diagramma è facilmente interpretabile da un lettore esperto, con terminologia coerente e struttura semantica ben definita.

La griglia di valutazione adottata consente di analizzare i diagrammi UML lungo tre dimensioni fondamentali: qualità sintattica, semantica e pragmatica. La struttura a sei criteri, ciascuno valutato su una scala da 0 a 2, garantisce una copertura completa degli aspetti formali, concettuali e comunicativi del modello. Tuttavia, è opportuno evidenziare alcune considerazioni critiche sulla struttura del sistema di punteggio.

Innanzitutto, la scala discreta a tre livelli semplifica l'assegnazione dei punteggi e ne facilita la standardizzazione, ma comporta una bassa granularità: situazioni qualitativamente diverse possono essere associate allo stesso valore, riducendo la capacità discriminativa della griglia. Inoltre, la distinzione tra i criteri *Validità* e *Completezza* può risultare talvolta sfumata, specie in presenza di concetti impliciti nella traccia che possono influenzare entrambe le dimensioni.

Un'ulteriore criticità riguarda il criterio relativo alla *Struttura JSON*, il quale valuta la compatibilità tecnica con la piattaforma Apollon. Sebbene tale aspetto sia rilevante per l'usabilità del modello, esso è esterno alla qualità concettuale del diagramma e può introdurre un bias di tipo formale. Un diagramma semanticamente corretto potrebbe essere penalizzato per mere questioni di serializzazione o, viceversa, un diagramma con errori da un punto di vista di modellazione, potrebbe comunque ottenere un buon punteggio dovuto alla giusta formattazione e leggibilità del codice JSON.

Va infine sottolineata l'assenza dell'attribuzione di pesi specifici a ciascun criterio: tutti contribuiscono in egual misura al punteggio finale, ma non necessariamente incidono con la stessa importanza sulla qualità complessiva del modello. Ad esempio, un diagramma formalmente corretto ma semanticamente scorretto può ottenere un punteggio analogo a uno semanticamente valido ma con lievi errori di notazione, alterando la significatività del punteggio aggregato.

Nel complesso, la griglia risulta funzionale per un'analisi strutturata e ripetibile, ma lascia margine per futuri perfezionamenti, quali l'introduzione di sottocriteri descrittivi, una scala di valutazione più fine o l'adozione di pesi differenziati per le diverse dimensioni della qualità.

ChatGPT

La Tabella 4.2 riporta i punteggi ottenuti da ChatGPT per ciascun esercizio UML in base ai sei criteri definiti nel Capitolo 3. I risultati mostrano la massima regolarità e costanza qualitativa del modello.

Tabella 4.2: Valutazione dettagliata di ChatGPT per ciascun esercizio UML

Esercizio	UML	JSON	Validità	Compl.	Chiarezza	Compr.	Totale
1	2	2	2	2	2	2	12
2	2	2	2	2	2	2	12
3	2	2	2	2	2	2	12
4	2	2	2	2	2	2	12
5	2	2	2	2	2	2	12
6	2	2	2	2	2	2	12
7	2	2	2	2	2	2	12
8	2	2	2	2	2	2	12
9	2	2	2	2	2	2	12
10	2	2	2	2	2	2	12
11	2	2	2	2	2	2	12
12	2	2	2	2	2	2	12
13	2	2	2	2	2	2	12
14	2	2	2	2	2	2	12
15	2	2	2	2	2	2	12

Analisi complessiva delle performance di ChatGPT

1. Qualità Sintattica

ChatGPT mostra un'eccellente padronanza della notazione UML e una buona conformità al formato JSON richiesto da Apollon. In tutti gli esercizi:

- Le classi sono ben strutturate, con nomi coerenti e delimitazione chiara tra attributi e metodi.
- Gli attributi risultano completi, con tipizzazione corretta nella quasi totalità dei casi.
- Le relazioni sono esplicitate in modo corretto e il layout è stabile e leggibile.
- La struttura JSON è sempre conforme.

Ciò dimostra una solidità complessiva nella generazione sintattica di diagrammi UML validi e pronti all'uso nei tool di modellazione.

2. Qualità Semantica

Anche dal punto di vista semantico ChatGPT si comporta molto bene:

- Il modello individua correttamente le entità principali del dominio e le relazioni significative.
- Mostra buona sensibilità nell'identificare strutture complesse come composizioni o generalizzazioni.

L'interpretazione semantica è corretta.

3. Qualità Pragmatica

ChatGPT genera diagrammi ordinati e facili da leggere:

- L'organizzazione spaziale dei diagrammi è chiara e priva di sovrapposizioni.
- Le denominazioni adottate sono autoesplicative e coerenti.

L'efficacia comunicativa del modello è elevata e rende i diagrammi adatti a contesti educativi e professionali.

Valutazione complessiva delle capacità di ChatGPT

ChatGPT è il modello più affidabile nella generazione automatica di diagrammi UML delle classi. I risultati ottenuti mostrano una notevole regolarità e aderenza agli standard sintattici. Nonostante alcune lievi mancanze dal punto di vista semantico avanzato, la qualità complessiva dei diagrammi prodotti è molto elevata.

Gemini

La Tabella 4.3 riporta i punteggi ottenuti da Gemini per ciascun esercizio UML.

Tabella 4.3: Valutazione dettagliata di Gemini per ciascun esercizio UML

Esercizio	UML	JSON	Validità	Compl.	Chiarezza	Compr.	Totale
1	2	2	2	2	2	2	12
2	2	2	2	2	2	2	12
3	2	2	2	2	2	2	12
4	2	2	2	2	2	2	12
5	2	2	2	2	2	2	12
6	2	2	1	1	2	2	10
7	2	2	1	1	2	2	10
8	2	2	2	2	2	2	12
9	2	2	2	2	2	2	12

Continua alla pagina successiva

Esercizio	UML	JSON	Validità	Compl.	Chiarezza	Compr.	Totale
10	2	2	2	2	2	2	12
11	2	2	2	2	2	2	12
12	2	2	2	2	2	2	12
13	2	2	2	2	2	2	12
14	2	2	2	2	2	2	12
15	2	2	2	2	2	2	12

Analisi complessiva delle performance di Gemini

1. Qualità Sintattica

Gemini mostra una padronanza eccellente della notazione UML e del formato JSON compatibile con Apollon, con una media complessiva pari a 8/8 punti possibili nelle due voci di questa dimensione. In tutti gli esercizi:

- Le classi sono correttamente definite e formalmente valide.
- Gli attributi sono completi e i tipi sono quasi sempre specificati correttamente.
- Le relazioni sono dichiarate con molteplicità coerenti e layout stabile.
- La struttura JSON è sempre compatibile, senza errori o problemi di leggibilità.

Questo dimostra che Gemini è altamente affidabile nella generazione di diagrammi UML formalmente corretti e utilizzabili direttamente nei tool di modellazione.

2. Qualità Semantica

Anche sotto il profilo semantico, Gemini si comporta molto bene, con una media complessiva di circa 3.7/4 nei criteri di validità e completezza:

- Il modello coglie in modo accurato le entità rilevanti del dominio, rappresentando correttamente sia le classi principali che le loro relazioni.
- Sono frequentemente incluse generalizzazioni, classi associate e attributi derivati quando opportuno.
- In alcuni casi sono modellati anche aspetti dinamici (es. metodi), anche se non sempre in modo approfondito.

Questo indica una comprensione approfondita delle specifiche testuali e una capacità di astrazione sopra la media nei task di modellazione concettuale.

3. Qualità Pragmatica

Gemini eccelle anche in questa dimensione, ottenendo una media di 4/4 nei criteri di chiarezza e comprensibilità:

- I diagrammi sono visivamente ordinati, con layout coerente e denominazioni esplicite.

- I nomi delle classi, attributi e relazioni sono scelti con cura, facilitando la lettura anche in assenza della traccia.

La qualità pragmatica dei diagrammi è molto alta, e riflette una capacità notevole di comunicare in modo efficace la struttura del dominio.

Valutazione complessiva delle capacità di Gemini

Gemini si dimostra il secondo modello più solido nella generazione automatica di diagrammi UML delle classi tra quelli testati. La produzione di strutture formalmente corrette, semanticamente complete e facilmente leggibili lo rende uno strumento adatto anche a contesti educativi o industriali. La sua performance regolare su tutti gli esercizi evidenzia una notevole robustezza e affidabilità.

Deepseek

La Tabella 4.4 riporta i punteggi ottenuti da Deepseek per ciascun esercizio UML.

Tabella 4.4: Valutazione dettagliata di Deepseek per ciascun esercizio UML

Esercizio	UML	JSON	Validità	Compl.	Chiarezza	Compr.	Totale
1	2	2	1	1	2	2	10
2	2	2	1	1	2	2	10
3	2	2	1	1	2	2	10
4	2	2	2	2	2	2	12
5	2	2	2	2	2	2	12
6	2	2	2	1	2	2	11
7	2	2	2	1	2	2	11
8	2	2	2	2	2	2	12
9	2	2	2	2	2	2	12
10	2	2	2	2	2	2	12
11	2	2	2	2	2	2	12
12	2	2	2	2	2	2	12
13	2	2	2	2	2	2	12
14	2	2	2	2	2	2	12
15	2	2	2	2	2	2	12

Analisi complessiva delle performance di DeepSeek

1. Qualità Sintattica

DeepSeek ha mostrato una buona padronanza della notazione UML e del formato JSON Apollon, con una media complessiva pari a 6.9/8 punti nella dimensione sintattica:

- Le classi sono generalmente ben formate, ma in alcuni esercizi si riscontrano differenze nella dichiarazione degli attributi o nella mancanza di alcuni elementi richiesti (es. metodi, enumerazioni).

- La struttura JSON è valida in tutti gli esercizi, con corretta compatibilità con Apollon, ma in almeno due casi si notano errori di molteplicità o struttura annidata non ottimale.

Complessivamente, DeepSeek si dimostra capace di produrre file leggibili e formalmente corretti, pur con qualche imperfezione ricorrente nella modellazione UML.

2. Qualità Semantica
Sul piano semantico, DeepSeek ottiene una media di 2.9/4 nei criteri di validità e completezza:

- I modelli prodotti identificano correttamente le entità principali nella maggior parte degli esercizi, ma a volte ignorano classi implicite o relazioni chiave (es. classi associate mancanti, generalizzazioni non modellate).
- Alcuni attributi risultano semanticamente generici, e si osservano errori nei nomi o nei tipi in più di un esercizio.

Comprensione parziale e non sempre approfondita delle specifiche, con tendenza a soluzioni minimali o conservative.

3. Qualità Pragmatica
La dimensione pragmatica presenta risultati discontinui, con una media di 3.2/4:

- I diagrammi sono per lo più leggibili, ma in certi casi il layout è sovraccarico o poco strutturato, e la nomenclatura non sempre chiara.
- Le denominazioni di classi e attributi sono spesso coerenti; manca in alcuni casi un'attenzione agli aspetti comunicativi del diagramma, come l'adozione di nomi descrittivi e una struttura graficamente ordinata.

La qualità pragmatica è accettabile, ma lascia spazio a miglioramenti soprattutto in termini di chiarezza visiva e usabilità del modello per un lettore esterno.

Valutazione complessiva delle capacità di DeepSeek
DeepSeek si rivela un modello solido, in grado di generare diagrammi UML generalmente corretti e utilizzabili, con un'elevata compatibilità tecnica con Apollon. Tuttavia, rispetto a Gemini e ChatGPT, mostra maggiori incertezze nella copertura semantica e nell'organizzazione visiva. Il suo comportamento risulta meno costante: alcune soluzioni sono molto vicine al riferimento, altre mostrano omissioni rilevanti.

Qwen

La Tabella 4.5 riporta i punteggi ottenuti da Deepseek per ciascun esercizio UML.

Tabella 4.5: Valutazione dettagliata di Qwen per ciascun esercizio UML

Esercizio	UML	JSON	Validità	Compl.	Chiarezza	Compr.	Totale
1	1	2	1	1	1	1	7
2	1	2	1	1	1	1	7
3	1	2	1	1	1	1	7
4	1	0	1	1	1	1	5
5	1	2	1	1	1	1	7
6	2	1	1	1	1	2	7
7	2	2	1	1	2	1	8
8	1	2	1	0	1	1	6
9	1	0	2	1	1	1	6
10	1	0	1	0	1	1	4
11	1	1	1	0	1	1	5
12	1	2	2	1	2	1	9
13	2	2	2	2	1	2	11
14	2	0	1	2	1	1	7
15	2	2	2	2	2	2	12

Analisi complessiva delle performance di Qwen

1. Qualità Sintattica

Qwen mostra una discreta padronanza della notazione UML e del formato JSON compatibile con Apollon, con una media complessiva pari a circa 6/8 punti possibili nelle due voci di questa dimensione. Nella maggior parte degli esercizi:

- Le classi sono correttamente definite.
- Gli attributi, quando presenti, sono formalmente validi, anche se talvolta mancano i tipi.
- Le relazioni sono dichiarate e, in diversi casi, includono correttamente le molteplicità.
- La struttura JSON è quasi sempre compatibile, con quattro casi di errore grave (file non leggibile).

Qwen è tecnicamente competente nel produrre una rappresentazione UML formalmente accettabile.

2. Qualità Semantica

Qui si riscontra una criticità più consistente. La media nei criteri di validità e completezza è inferiore, attorno a 1/2 per ciascun criterio, mostrando i seguenti limiti:

- Qwen riconosce i concetti principali del dominio, ma trascura frequentemente i vincoli dinamici o le funzionalità attese (es. comportamenti, metodi, calcoli automatizzati, attori attivi).
- Le relazioni chiave sono talvolta rappresentate come attributi o descritte in modo ambiguo.

Questo indica che Qwen ha difficoltà nella comprensione profonda del dominio e nell'estrazione degli elementi comportamentali o dinamici, aspetto tipico nei task di modellazione concettuale.

3. Qualità Pragmatica

Qwen mantiene una buona leggibilità generale, con punteggi medi di 1.5/2 in chiarezza e comprensibilità:

- I diagrammi sono visivamente puliti, con layout coerente e nomi spesso espliciti.
- Tuttavia, l'omissione dei nomi delle associazioni e la scarsa esplicitazione dei ruoli rendono alcuni modelli più difficili da interpretare senza conoscenza del testo.

L'aspetto pragmatico è accettabile ma può essere migliorato con un maggiore focus sulla comunicazione efficace del modello.

Valutazione complessiva

Qwen dimostra un livello medio-basso di competenza nella generazione automatica di diagrammi UML delle classi. La capacità di produrre strutture sintatticamente valide e JSON leggibili è solida, ma la comprensione semantica del dominio e delle funzionalità richieste è ancora limitata, risultando in modelli concettualmente incompleti o parzialmente errati.

4.2 Analisi in base alla difficoltà degli esercizi

4.2.1 Classificazione degli esercizi per difficoltà

La Tabella 4.6 riporta la classificazione dei 15 esercizi UML utilizzati, secondo tre dimensioni: complessità sintattica, grado di astrazione/ambiguità, e vincoli impliciti. I valori aggregati (IDC) sono stati utilizzati per suddividere gli esercizi in tre livelli di difficoltà. In questa tabella compare anche l'indice Gulpease di leggibilità, un valore che è stato calcolato ma che tuttavia si è rivelato non rilevante nell'analisi dei risultati. Quest'ultimi non dimostrano infatti nessun collegamento tra la performance dei LLM e la leggibilità del testo (esercizi svelti con performance di livello avevano indifferentemente leggibilità alta, valori alti dell'indice, o bassa) ne tantomeno con la difficoltà degli esercizi (l'esercizio leggibilità più bassa è di difficoltà media).

Tabella 4.6: Classificazione della difficoltà degli esercizi UML

Esercizio	Struttura	Ambiguità	Vincoli	IDC	Difficoltà	Gulpease
Esercizio 1	3.0	2.0	3.0	8.0	Difficile	68.23
Esercizio 2	3.0	2.0	3.0	8.0	Difficile	75.14
Esercizio 3	2.0	1.0	2.0	5.0	Media	49.43
Esercizio 4	3.0	2.0	3.0	8.0	Difficile	70.60
Esercizio 5	3.0	2.0	3.0	8.0	Difficile	71.57
Esercizio 6	2.0	1.0	2.0	5.0	Media	69.00
Esercizio 7	2.0	2.0	2.0	6.0	Media	57.64
Esercizio 8	1.0	1.0	2.0	4.0	Facile	68.75
Esercizio 9	1.0	1.0	2.0	4.0	Facile	69.33
Esercizio 10	3.0	2.0	3.0	8.0	Difficile	59.00
Esercizio 11	3.0	2.0	3.0	8.0	Difficile	64.71
Esercizio 12	3.0	2.0	3.0	8.0	Difficile	59.59
Esercizio 13	2.0	2.0	3.0	7.0	Difficile	75.41
Esercizio 14	2.0	1.0	2.0	5.0	Media	69.00
Esercizio 15	1.0	1.0	1.0	3.0	Facile	73.87

4.2.2 Prestazioni dei modelli per livello IDC

La Tabella 4.7 mostra il punteggio medio complessivo ottenuto da ciascun modello sui tre insiemi di esercizi (facili, medi, difficili) e il valore medio ponderato sulla base dell'indice IDC. Questo permette di evidenziare la sensibilità dei modelli rispetto alla complessità del task.

Tabella 4.7. Confronto delle performance dei modelli per livello di difficoltà

Modello	Media Facile	Media Media	Media Difficile	Media Ponderata IDC
Gemini	12.00	11.00	12.00	11.77
ChatGPT	12.00	12.00	12.00	12.00
Qwen	8.33	8.00	7.12	7.38
DeepSeek	12.00	11.00	11.50	11.44

Come mostrato nella Tabella 4.7, i modelli evidenziano comportamenti significativamente diversi in relazione alla complessità concettuale degli esercizi.

Il modello ChatGPT si distingue per la totale stabilità dei risultati: il punteggio medio rimane invariato (12.00) indipendentemente dal livello di difficoltà. Questa assenza di variazioni può essere interpretata come un segnale di robustezza, ma anche come possibile sintomo di insensibilità alla reale complessità del task: il modello tende cioè a comportarsi in modo uniforme, anche quando la complessità semantica o strutturale

dovrebbe invece rendere più difficile ottenere una generazione corretta. Ciò suggerisce che ChatGPT è in grado di gestire anche i casi più complessi, ma potrebbe altresì "sovra-generalizzare", producendo output formalmente corretti ma non sempre aderenti alla logica profonda del dominio.

Gemini e DeepSeek mostrano invece una leggera flessione in corrispondenza degli esercizi di livello medio, con una ripresa nei casi difficili. Questo andamento non lineare potrebbe riflettere una certa dipendenza del comportamento del modello dalla formulazione testuale del prompt o da specifiche ambiguità presenti negli esercizi medi. Tuttavia, entrambi i modelli mantengono punteggi elevati anche nei casi difficili, denotando una buona capacità di adattamento e generalizzazione.

Il caso più interessante è rappresentato da Qwen, che evidenzia una chiara tendenza regressiva al crescere della difficoltà: si passa da una media di 8.33 negli esercizi facili a soli 7.12 in quelli difficili, con un valore medio ponderato molto inferiore rispetto agli altri modelli (7.38). Questo comportamento segnala una sensibilità marcata alla complessità del task, traducendosi in una perdita di precisione e coerenza nei contesti più sfidanti. L'analisi qualitativa degli output mostra che, nei casi complessi, Qwen commette spesso errori strutturali (ad esempio, uso errato delle associazioni o modelli incompleti), suggerendo che il modello non riesce a mantenere una coerenza interna del diagramma man mano che aumenta il carico cognitivo richiesto dalla traccia.

Nel complesso, il valore della media ponderata IDC fornisce una sintesi efficace della resilienza del modello al variare della difficoltà: ChatGPT conferma prestazioni ottimali e costanti, mentre Qwen subisce un calo significativo. Gemini e DeepSeek si collocano in una posizione intermedia, con un buon equilibrio tra stabilità e sensibilità.

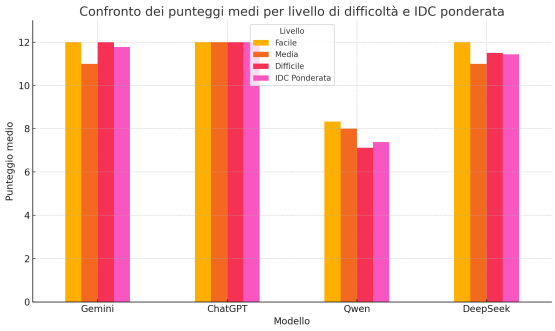


Figura 4.2. Valore della media ponderata IDC

4.3 Analisi qualitativa degli errori commessi

4.3.1 Classificazione e frequenza degli errori

Oltre ai punteggi numerici, è stato utile analizzare la tipologia e la frequenza degli errori commessi da ciascun modello (Tabella 4.8). Si evidenziano quattro categorie principali:

- **Errori di struttura JSON**: violazioni sintattiche che rendono il file incompatibile con Apollon.
- **Errori di molteplicità**: assegnazione errata dei bounds delle associazioni, spesso fuori standard.
- **Errori sulle generalizzazioni**: uso improprio o assenza di relazioni padre-figlio quando richieste.
- **Relazioni mancanti**: omessa modellazione di collegamenti fondamentali (es. tra entità centrali).

La Tabella 4.8 mostra che:

- ChatGPT resta il modello più affidabile: zero JSON non validi, pochi errori di molteplicità e soltanto un caso di generalizzazione impropria. Questo suggerisce che, pur generando output formalmente corretti, il modello talvolta sottostima la complessità delle cardinalità ma mantiene un alto livello di coerenza strutturale.
- Gemini presenta una distribuzione di errori contenuta, ma spiccano due imprecisioni sulle generalizzazioni. Il dato suggerisce che il modello tende a confondere le gerarchie di classe o a non esplicitarle quando necessario, pur gestendo bene la sintassi JSON.
- DeepSeek si distingue per l'assenza di errori sintattici e di concettualizzazione, ma commette due omissioni di relazione. Ciò indica una buona padronanza delle regole formali, con qualche lacuna nel coprire l'intero spettro semantico richiesto dagli esercizi più complessi.
- Qwen rimane il punto critico del confronto: cinque JSON non validi rendono il modello poco utilizzabile "out of the box", e il numero elevato di errori strutturali (molteplicità errate e relazioni mancanti) segnala una difficoltà sistemica nell'internalizzare i vincoli di dominio rispetto agli altri LLM.

Tabella 4.8. Frequenza degli errori per modello e categoria (conteggio aggiornato)

Modello	JSON non validi	Molteplicità errate	Generalizzazioni errate	Relazioni mancanti
ChatGPT	0	2	1	0
Gemini	0	1	2	0
DeepSeek	0	0	0	2
Qwen	5	3	1	3

Nel complesso, le categorie più problematiche per tutti i modelli sono molteplicità e generalizzazioni, mentre i JSON non validi rappresentano un problema esclusivo (e penalizzante) per Qwen. Queste evidenze supportano tre implicazioni pratiche:

Prompt engineering mirato a cardinalità e generalizzazioni: inserire esempi espliciti e controlli post-processing riduce gli errori di molteplicità, anche per i modelli più robusti.

Validazione automatica del JSON: integrare un check sintattico nella pipeline evita output inutilizzabili e rende il sistema più affidabile nell'uso didattico.

Selezione del modello per contesto: ChatGPT o DeepSeek risultano più adatti in scenari educativi dove la correttezza formale è prioritaria, mentre Gemini può essere scelto se si privilegia la flessibilità, purché si presti attenzione alle gerarchie.

Questi risultati completano il quadro sulla qualità dei modelli e ne guidano la scelta in funzione del trade-off tra robustezza, aderenza semantica e effort di post-processing.

4.4 Casi studio esemplificativi

Al fine di approfondire le osservazioni quantitative emerse nella fase di valutazione, questa sezione presenta tre casi studio selezionati per la loro capacità di mettere in luce comportamenti ricorrenti, punti di forza e debolezze specifiche dei modelli testati. Ogni caso è stato scelto in funzione della varietà di errori osservati e della diversità di approccio tra i modelli, a parità di traccia di partenza.

I tre esempi coprono rispettivamente:

- un esercizio di livello *facile*, utile per confrontare la gestione dei dettagli minimi;
- un esercizio di livello *medio*, che ha generato divergenze semantiche significative;
- un esercizio di livello *difficile*, che ha messo in crisi la robustezza sintattica e concettuale dei modelli più deboli.

4.4.1 Caso studio 1 – Esercizio 9 (facile): output corretti ma approcci diversi

L'Esercizio 9, classificato come *facile*, richiedeva di modellare un sistema con struttura lineare e bassa ambiguità. Tutti i modelli hanno prodotto diagrammi formalmente validi, ma si sono osservate differenze interessanti nell'organizzazione e nell'interpretazione concettuale.

ChatGPT ha generato un diagramma perfettamente conforme, con nomi coerenti, layout ordinato e corretta attribuzione delle molteplicità. Anche i concetti impliciti sono stati tradotti in classi o associazioni appropriate.

Gemini ha prodotto un output altrettanto corretto, con lievi variazioni nell'ordine delle classi e nella denominazione di alcuni attributi, senza tuttavia compromettere la qualità complessiva del modello.

Qwen, pur raggiungendo una buona leggibilità, ha rappresentato alcune relazioni come semplici attributi, riducendo così l'esplicitazione dei vincoli strutturali. Questo

comportamento, frequente nei suoi output, suggerisce un'interpretazione più superficiale delle tracce, anche in assenza di reali ambiguità.

DeepSeek ha fornito un diagramma corretto, ma meno curato graficamente: si notano nomi generici per le classi (es. "Entity1", "Info").

Il caso evidenzia come, anche in situazioni semplici, la profondità interpretativa e la qualità espressiva del diagramma possano variare sensibilmente in base al modello utilizzato.

Listing 4.1. Esercizio 9

```
Si vuole progettare un'applicazione in grado di gestire un'agenzia
    immobiliare. Gli immobili sono caratterizzati da un codice
   alfanumerico, indirizzo, cap, città e da una superficie
   espressa in mq attraverso un numero intero. L'agenzia gestisce
   diverse tipologie di immobili: Box, Appartamenti e Ville.
Per i box è riportato il numero di posti auto. Per gli
   appartamenti è riportato in numero di vani e il numero di bagni
   , e per le ville è previsto, in aggiunta, la dimensione in mq
   di giardino rispetto all'appartamento.
Per gli immobili è necessario definire un metodo che restituisca,
   in un formato stringa a piacere, il tipo di immobile e tutte le
    corrispondenti proprietà.
L'agenzia deve poter eseguire delle ricerche specificando una
   chiave. Per le ricerche si considera l'operatore di
   contenimento, cioè si verifica se la chiave di ricerca è
   contenuta all'interno di una proprietà dell'oggetto. Il
   risultato della ricerca è visualizzato a video stampando la
   scheda dell'immobile corrispondente.
```

4.4.2 Caso studio 2 – Esercizio 13 (medio): generalizzazione implicita e relazioni non ovvie

L'Esercizio 13, di difficoltà *media*, ha evidenziato divergenze più marcate nei diagrammi prodotti. La traccia includeva un riferimento implicito a una generalizzazione tra entità (es. *utente semplice* e *amministratore*) e richiedeva la modellazione di un'associazione di tipo "gestisce".

ChatGPT ha individuato correttamente la gerarchia di classi, utilizzando una relazione di generalizzazione e separando ruoli e responsabilità. Inoltre, ha introdotto una classe associativa per modellare in modo esplicito la gestione di permessi, mostrando attenzione semantica e coerenza strutturale.

Gemini ha riconosciuto la generalizzazione, ma ha rappresentato le relazioni in modo più sintetico: la classe Admin è stata modellata come entità separata, senza legame esplicito con User, suggerendo una comprensione solo parziale dell'implicita struttura gerarchica.

DeepSeek, pur rispettando le convenzioni sintattiche, ha omesso la generalizzazione, trattando le due entità come indipendenti. Inoltre, la relazione di gestione è stata rappresentata come attributo, riducendo la chiarezza concettuale del modello.

Qwen ha prodotto un diagramma formalmente valido, ma semanticamente debole: ha utilizzato nomi generici e non ha esplicitato né la gerarchia né le relazioni funzionali tra gli attori. Questo comportamento è coerente con la sua tendenza a semplificare la struttura nei task più astratti.

Il caso mostra come, in presenza di ambiguità o gerarchie implicite, la capacità di astrazione diventi determinante per la qualità semantica del diagramma.

Listing 4.2. Esercizio 13

```
    Un ristorante decide di lanciare il servizio "Eat @Home", che
       prevede l'ordinazione online dei piatti e la consegna a
       domicilio.
Ogni cliente registrato può accedere al sistema tramite web o app
   e scegliere da un menu i piatti che desidera ricevere per
   comporre un ordine.Per ogni cliente, oltre a nome, cognome,
   email e telefono, è conosciuto l'indirizzo di casa, che
   consente di calcolare il tempo di trasporto stimato tramite
   servizi esterni.Quando un ordine è stato completato, il sistema
    mostra il costo totale e il tempo di consegna stimato,
   calcolato in base al tempo di preparazione dei piatti contenuti
   , al carico di lavoro attuale della cucina (in funzione del
   numero di piatti in preparazione) e al tempo di trasporto.Il
   cliente può confermare l'ordine e procedere al pagamento con
   carta di credito. Dopo che il pagamento è andato a buon fine, l
   'ordine può essere trasmesso alla cucina.Quando il personale di
    cucina prende in carico l'ordine, il sistema invia un
   messaggio al cliente indicando l'inizio della preparazione e
   fornendo una stima aggiornata del tempo di consegna.Quando il
   personale di cucina consegna i piatti al reparto consegne, il
   sistema invia un messaggio al cliente informandolo della
   spedizione e fornendo una nuova stima del tempo di consegna.
   Quando il personale di consegna ha consegnato i piatti al
   cliente, segnala la conclusione dell'ordine.In qualsiasi
   momento il cliente può accedere al sistema e controllare lo
   stato del proprio ordine e il tempo di consegna stimato
```

4.4.3 Caso studio 3 – Esercizio 8 (difficile): errori strutturali e fallimenti formali

L'Esercizio 8, pur classificato come *facile* per la struttura, è risultato particolarmente critico per alcuni modelli, a causa di formulazioni linguistiche ambigue e di un vincolo implicito sulla gestione dei ruoli familiari.

ChatGPT ha trattato correttamente la relazione ricorsiva tra persone, distinguendo il concetto di "parente" da quello di "figlio" mediante ruoli e molteplicità. Il diagramma risultante è semanticamente accurato e formalmente impeccabile.

Gemini ha generato una soluzione simile, ma con una modellazione più semplice: la relazione è stata definita genericamente come "relazione familiare", senza specificare i ruoli. Questo riduce la precisione semantica, ma non compromette la comprensibilità.

DeepSeek ha presentato una modellazione più confusa: il concetto di "padre" e "figlio" è stato tradotto in classi distinte anziché ruoli, e la relazione ricorsiva è stata omessa. Formalmente il file è valido, ma il significato concettuale risulta forzato.

Qwen ha prodotto un output con gravi errori:

- struttura JSON non compatibile (Apollon non carica il file),
- classi Padre, Madre, Figlio trattate come entità a sé, violando i principi della modellazione UML (dove tali concetti andrebbero modellati come ruoli o proprietà),
- relazione familiare non presente, o rappresentata come attributo generico.

Il diagramma, pur contenente elementi sintattici validi, non veicola correttamente il significato del dominio, e risulta inutilizzabile ai fini comunicativi o analitici.

Listing 4.3. Esercizio 8

```
    Sistema di controllo degli ascensori in un palazzo di m piani.
        Il sistema di controllo deve gestire il movimento degli
       ascensori tra i diversi piani in accordo con i seguenti
       vincoli:
Ogni ascensore ha un insieme di bottoni corrispondenti ai diversi
   piani. Questi bottoni si
illuminano quando vengono premuti indicando la richiesta di
   arresto dell'ascensore al raggiungimento del piano
   corrispondente.
Vincoli (segue): In ogni piano, ad eccezione del primo e dell'
   ultimo, esistono due bottoni per chiamare l'ascensore
   specificando se si vuole salire o scendere. I bottoni si
   illuminano quando vengono premuti. L'illuminazione si spegne
   quando l'ascensore arriva al piano muovendosi nella direzione
   corrispondente al bottone illuminato
```

4.4.4 Conclusioni dai casi studio

L'analisi qualitativa dei tre esempi selezionati conferma quanto osservato nei risultati aggregati:

- **ChatGPT** è il modello più affidabile, capace di combinare correttezza sintattica, profondità semantica e chiarezza espositiva in modo sistematico.
- **Gemini** si dimostra solido ma talvolta conservativo, soprattutto nella gestione di generalizzazioni e vincoli impliciti.
- **DeepSeek** oscilla tra buone intuizioni strutturali e semplificazioni eccessive, con scarsa attenzione alla semantica del dominio.
- **Qwen**, pur mantenendo compatibilità sintattica in alcuni casi, mostra limiti severi nella comprensione concettuale e nell'espressività.

Questi casi esemplificano concretamente come, a parità di input, i modelli possano produrre risultati molto differenti, e come la qualità del prompting, la robustezza semantica e la sensibilità ai dettagli influenzino la qualità dell'output generato.

4.5 Effetto del prompting sui risultati di ChatGPT

4.5.1 Confronto tra Prompt 1 e Prompt 2

I due prompt impiegati con ChatGPT differiscono sotto tre dimensioni chiave:

- granularità delle istruzioni
- copertura dei vincoli sintattici
- strategie di disambiguazione semantica

Il *Prompt 1* fornisce indicazioni complete sul formato JSON e include esempi few-shot, ma lascia alcune regole (pes. posizionamento dei `bounds` o coordinate dei `path`) implicitamente demandate al modello. Il *Prompt 2*, introdotto in un secondo momento, esplicita tutti i vincoli formali, impone coordinate positive e path con larghezza > 0, proibisce collezioni come tipi attributo e aggiunge una checklist finale che riepiloga i campi obbligatori. Queste differenze trasformano il prompt da descrizione "orientativa" a *specifica eseguibile* per l'LLM, riducendo drasticamente lo spazio di ambiguità.

4.5.2 Analisi comparativa per criterio

Nella Tabella 4.9 è riportata la media dei punteggi ottenuti da ChatGPT per ciascun criterio valutativo, distinguendo i risultati ottenuti con il Prompt 1 e il Prompt 2.

Tabella 4.9. Confronto tra Prompt 1 e Prompt 2 per ChatGPT (media su 15 esercizi)

Prompt	UML	JSON	Validità	Completezza	Chiarezza	Comprensibilità
1	1.50	2.00	1.33	0.93	1.90	1.67
2	2.00	2.00	2.00	2.00	2.00	2.00

La Tabella 4.9 riporta la media dei punteggi ottenuti da ChatGPT sui quindici esercizi usando i due prompt. I valori mostrano che:

- la dimensione sintattica resta alta con entrambi i prompt (JSON sempre valido; sintassi UML da 1.50 a 2.00);
- gli scarti più ampi riguardano completezza (0.93 $\rightarrow$ 2.00) e validità semantica (1.33 $\rightarrow$ 2.00);
- chiarezza grafica e comprensibilità migliorano ma partivano già da valori prossimi al massimo.

Il guadagno medio complessivo è di +1.1 punti per esercizio, concentrato quasi interamente sulle dimensioni semantiche e pragmatiche.

4.5.3 Motivazioni delle differenze osservate

L'analisi delle tracce generate da ChatGPT nei due scenari sperimentali evidenzia quattro cause prevalenti del miglioramento osservato con il Prompt 2:

1. *Esplicitazione assertiva delle regole strutturali.* Il Prompt 2 ripete in modo sistematico e ridondante i vincoli formali (esistenza di `owner`, `bounds`, `path`) e li formula al tempo imperativo, specificando chiaramente anche cosa *non* deve essere fatto (ad esempio: non inserire oggetti all'interno del blocco `interactive`). Questo stimola efficacemente l'*instruction-following* del modello, riducendo omissioni ed errori di struttura, e migliorando la compatibilità con Apollon.

2. *Controllo puntuale del layout grafico.* Rispetto al Prompt 1, il Prompt 2 impone vincoli espliciti sulle coordinate (`x > 0`, `y > 0`), sull'ampiezza dei `path` e sull'uso di direzioni standardizzate (`Left`, `Right`, ecc.). Ciò consente di evitare sovrapposizioni e intersezioni, migliorando la leggibilità visiva e la qualità pragmatica del diagramma prodotto.

3. *Validazione implicita e checklist strutturata.* Il Prompt 2 include una sezione riepilogativa dei campi obbligatori da rispettare, agendo come una checklist interna al prompt stesso. Inoltre, la presenza di due esempi completi (testo + JSON compatibile) rafforza l'apprendimento in-context: eventuali deviazioni dello stile vengono scoraggiate perché divergono dalle strutture apprese. Insieme, questi elementi migliorano la robustezza e la coerenza degli output anche in assenza di supervisione esterna.

4. *Disambiguazione semantica dei concetti avanzati.* Il Prompt 2 chiarisce come gestire concetti spesso critici nei diagrammi UML: generalizzazioni implicite, classi associative, ruoli concettuali (come "Padre" e "Figlio") e attributi composti. Introduce inoltre regole restrittive sul tipo degli attributi (solo primitivi), sulla modellazione delle molteplicità e sull'uso delle relazioni parte-tutto. Questo riduce significativamente la variabilità interpretativa e migliora la validità semantica e la completezza concettuale del modello generato.

Nel complesso, il Prompt 2 si comporta come una vera e propria specifica eseguibile per l'LLM: non solo guida la generazione, ma ne vincola fortemente l'output, assicurando una maggiore affidabilità rispetto alla versione precedente.

4.5.4 Implicazioni per la progettazione dei prompt

L'analisi delle trace di ChatGPT evidenzia quattro cause prevalenti del salto di qualità:

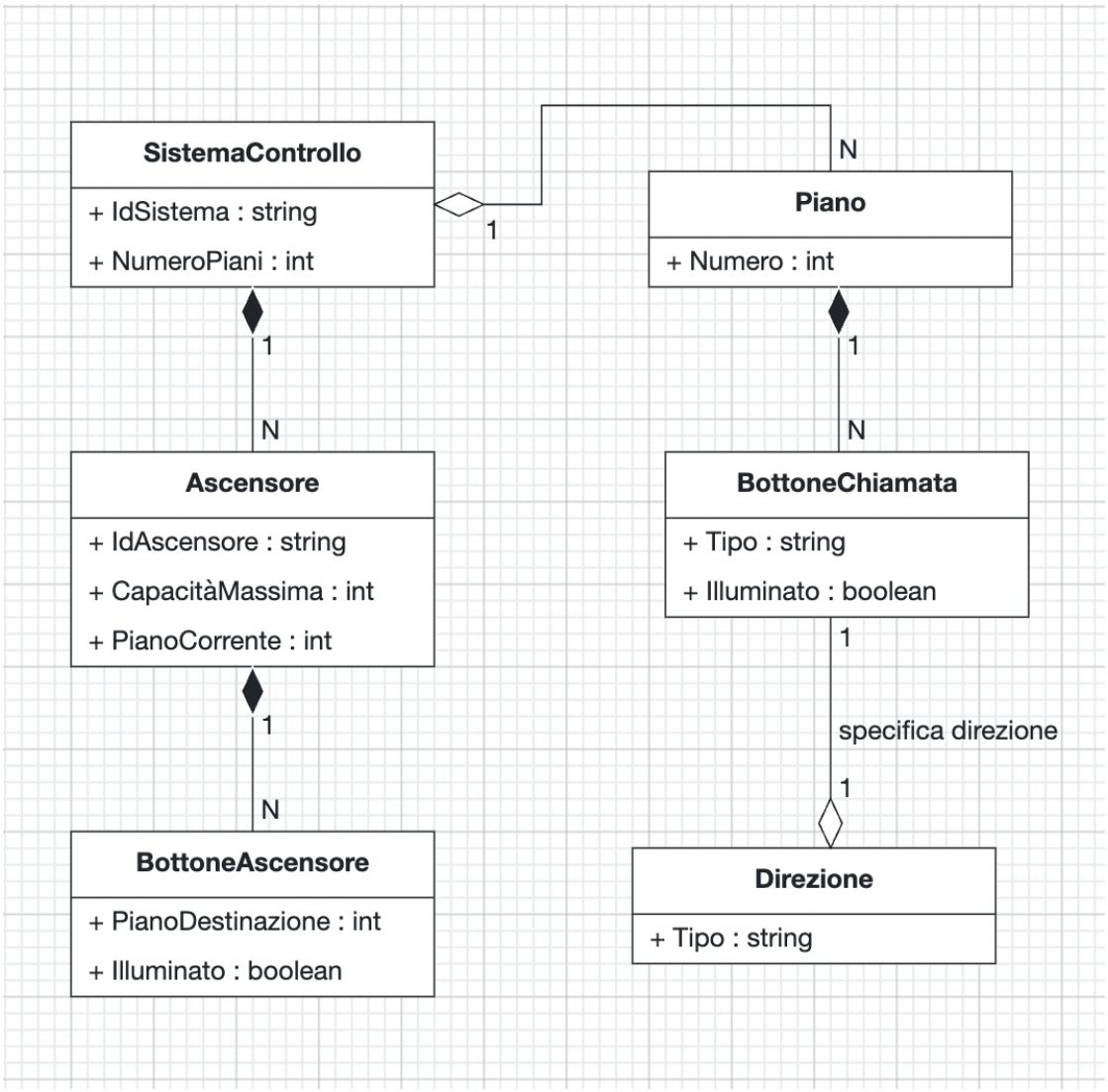
- *Ridondanza controllata delle istruzioni.* Il Prompt 2 ripete sistematicamente i vincoli strutturali fondamentali (come la presenza obbligatoria di `owner`, `bounds`, `path`) formulandoli al tempo imperativo. Questa ripetizione intenzionale stimola il comportamento di instruction-following del modello e riduce gli errori di omissione, specialmente nelle sezioni più lunghe del prompt.
- *Validazione implicita tramite esempi.* I due esempi completi inclusi nel prompt fungono da test-suite in-context: la dissonanza tra l'output e la struttura attesa penalizza, a livello di probabilità, le soluzioni scorrette, guidando il modello verso produzioni più aderenti. Gli esempi devono essere verificati sul tool di destinazione (es. Apollon), per vincolare in modo efficace la distribuzione dell'output.
- *Controllo esplicito del layout.* L'inserimento di vincoli su coordinate, dimensioni dei path e direzioni (`Left|Right|Up|Down`) riduce intersezioni, sovrapposizioni e dispersione visiva. Questo tipo di controllo incrementa significativamente la leggibilità e l'usabilità del diagramma finale.
- *Disambiguazione dei concetti avanzati.* Il prompt distingue ruoli, generalizzazioni e aggregazioni attraverso regole dedicate, evitando errori comuni come il modellare i ruoli ("Padre", "Figlio") come classi o l'inversione del diamante nelle composizioni. Questa esplicitazione migliora la validità semantica e la completezza concettuale dei modelli generati.

Alla luce di queste osservazioni, è possibile formulare tre raccomandazioni operative per la progettazione di prompt robusti nei task di generazione strutturata:

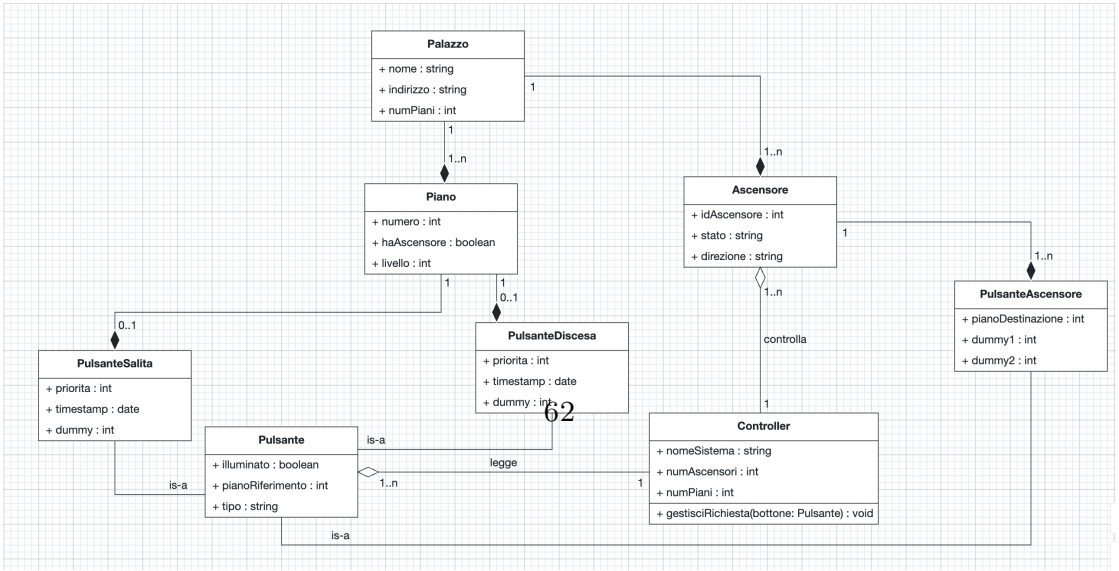
- Introdurre checklist finali di validazione interna, affinché il modello ricapitoli i vincoli prima di generare l'output.
- Inserire almeno due esempi completi, conformi e compatibili con il tool di destinazione, così da fornire al modello un riferimento diretto e vincolante.
- Combinare formulazioni positive e negative delle istruzioni (es. "non inserire elementi in `interactive`"), così da delimitare con chiarezza lo spazio delle soluzioni lecite.

4.5.5 Soluzioni esercizio 8 dei due prompt

L'esercizio 8, che richiedeva di modellare relazioni di parentela ricorsive, è emblematico: con il Prompt 1 ChatGPT ha omesso la relazione padre-figlio e usato molteplicità incoerenti (punteggio 0.93 in completezza), mentre il Prompt 2 ha prodotto un diagramma corretto e caricato senza errori in Apollon (punteggio 2 su tutti i criteri). Le Figure 4.4-a e 4.4-b, mostrano visivamente la differenza di layout e di correttezza semantica. In sintesi, la sperimentazione conferma che ChatGPT può raggiungere performance ottimali solo quando il prompt riduce al minimo sia l'ambiguità sintattica sia l'ambiguità semantica; in assenza di tali vincoli, il modello tende a interpretazioni economiche del dominio, penalizzando soprattutto la completezza concettuale.



(a) Soluzione Prompt 1



(b) Soluzione Prompt 2

Figura 4.3. Confronto tra i diagrammi generati con Prompt 1 e Prompt 2 per l'esercizio 8

4.6 Discussione dei risultati e confronto tra modelli

4.6.1 Stabilità e sensibilità

Uno degli obiettivi centrali della sperimentazione è stato valutare non solo la qualità media dei diagrammi generati, ma anche la stabilità dei modelli nel mantenere prestazioni costanti, e la loro sensibilità ai fattori di complessità concettuale o variazioni di input.

Stabilità intra-modello La stabilità è stata osservata considerando la varianza dei punteggi ottenuti da ciascun modello sui diversi esercizi. In particolare:

ChatGPT ha mostrato la massima stabilità possibile: ha ottenuto il punteggio massimo (12/12) in tutti gli esercizi, senza alcuna variazione tra un task e l'altro. La sua produzione è completamente deterministica nel contesto valutato, segno di una perfetta internalizzazione del prompt e una elevata robustezza architetturale.

Gemini presenta una stabilità molto elevata, con una sola flessione nei casi di esercizi moderatamente ambigui (es. Esercizi 6 e 7). Anche in questi casi, tuttavia, le variazioni sono lievi e circoscritte alla dimensione semantica. Complessivamente, il comportamento è regolare e prevedibile.

DeepSeek mostra un comportamento più irregolare: alterna esercizi con prestazioni elevate (punteggi massimi) ad altri in cui la semantica viene interpretata in modo incompleto o impreciso. Questo pattern riflette una minore consistenza nella gestione del contenuto testuale e una tendenza a risposte minimali in esercizi più ambigui.

Qwen è il modello più instabile: i punteggi variano in modo significativo da un esercizio all'altro, con output che spaziano da diagrammi completi e corretti a soluzioni sintatticamente errate o semanticamente incoerenti. La stabilità è compromessa sia nella dimensione sintattica (4 errori JSON), sia in quella semantica (relazioni mancanti, ruoli modellati come classi).

Sensibilità alla difficoltà del task La sensibilità è stata misurata confrontando le performance dei modelli sui tre livelli di difficoltà definiti tramite l'Indice di Complessità Concettuale (IDC). I risultati mostrano che:

ChatGPT è totalmente insensibile alla difficoltà: mantiene performance perfettamente costanti (12/12) anche in presenza di tracce ambigue, con vincoli impliciti o formulazioni articolate. Questo indica una capacità superiore di adattarsi alla complessità testuale senza degradazione della qualità.

Gemini manifesta una leggera sensibilità nella fascia "media", dove talvolta omette dettagli o semplifica la rappresentazione (es. omissione di ruoli o classi associate). Tuttavia, riesce a recuperare pienamente nei task "difficili", suggerendo che le sue prestazioni migliorano al crescere della struttura logica del task.

DeepSeek mostra una sensibilità marcata: i punteggi decrescono progressivamente all'aumentare della difficoltà, in particolare nella completezza semantica. Ciò suggerisce che la sua capacità di astrazione è più fragile, e che in presenza di ambiguità il modello tende a fornire risposte conservative o incomplete.

Qwen è il più sensibile di tutti: la qualità dell'output cala bruscamente negli esercizi difficili, sia per la maggiore incidenza di errori strutturali (es. file JSON invalidi), sia per l'inadeguata rappresentazione delle relazioni e dei concetti astratti. L'indice IDC si rivela quindi un predittore affidabile delle difficoltà che Qwen incontra.

Implicazioni Questi risultatu evidenziano un legame diretto tra:

- la maturità del modello (in termini di capacità di reasoning e instruction-following).
- e la capacità di mantenere performance costanti e robuste di fronte a task variabili.

Nel complesso:

- ChatGPT e Gemini risultano essere i modelli più robusti e consistenti.
- DeepSeek mostra un comportamento intermittente, sensibile al carico cognitivo implicito.
- Qwen si rivela instabile e fragile, sia rispetto alla difficoltà del task, sia rispetto alla coerenza interna dei suoi output.

4.6.2 Osservazioni conclusive sui modelli

L'analisi complessiva dei risultati sperimentali consente di delineare un quadro chiaro e articolato delle performance dei modelli valutati. Oltre alle differenze nei punteggi aggregati, è emersa una varietà di comportamenti che riflette le specifiche capacità, i limiti architetturali e la sensibilità semantica di ciascun LLM.

ChatGPT (GPT-4) ChatGPT si conferma il modello più affidabile e avanzato tra quelli testati. Le sue performance si distinguono per:

- **Eccellenza sintattica**: tutti i diagrammi prodotti sono formalmente corretti e immediatamente compatibili con Apollon.
- **Completezza semantica**: le entità, le relazioni e i vincoli impliciti sono sistematicamente individuati e rappresentati correttamente.
- **Chiarezza pragmatica**: l'output è leggibile, ben strutturato e conforme alle convenzioni del linguaggio UML.
- **Robustezza e stabilità**: nessuna variazione qualitativa si osserva al variare della difficoltà del task o della formulazione del prompt.

Il comportamento di ChatGPT evidenzia una comprensione avanzata delle istruzioni, una notevole capacità di astrazione e un ottimo bilanciamento tra precisione formale e profondità concettuale. È il modello che più si avvicina a un assistente esperto di modellazione.

Gemini Gemini si posiziona subito dopo ChatGPT in termini di affidabilità e qualità dei risultati. Le sue caratteristiche salienti includono:

- **Alta correttezza sintattica**, anche nei task più complessi.
- **Buona copertura semantica**, sebbene con occasionali semplificazioni (es. ruoli non modellati, gerarchie non esplicitate).
- **Chiarezza espositiva** costante, con denominazioni coerenti e layout leggibile.
- **Sensibilità limitata alla difficoltà**, con leggere flessioni in presenza di ambiguità.

Gemini rappresenta una soluzione efficace e robusta, adatta a contesti educativi o professionali in cui si richieda precisione strutturale e interpretazione semantica solida, anche se meno profonda rispetto a ChatGPT.

DeepSeek DeepSeek mostra prestazioni più variabili e una maggiore sensibilità alla complessità del task. Le sue principali caratteristiche sono:

- **Ottima compatibilità JSON** e adesione al formato richiesto, anche in presenza di prompt minimali.
- **Semantica discontinua**: buone intuizioni nei task semplici, ma semplificazioni e omissioni nei casi più strutturati.
- **Qualità pragmatica accettabile**, ma talvolta penalizzata da denominazioni generiche o layout poco curato.
- **Elevata variabilità intra-modello**, con output che alternano soluzioni valide ad altre approssimative.

DeepSeek è un modello capace, ma non sempre affidabile. Può produrre output validi, ma richiede un controllo umano attento, soprattutto nei task concettualmente più esigenti.

Qwen Qwen è il modello che presenta le maggiori criticità:

- **Errori frequenti di struttura JSON**, con diversi file non caricabili in Apollon.
- **Modellazione superficiale**: entità spesso mal collegate, relazioni assenti o rappresentate come attributi.
- **Interpretazione debole dei vincoli e dei ruoli**, con generalizzazioni spesso ignorate o mal poste.
- **Instabilità elevata**: performance altalenanti, anche all'interno di esercizi dello stesso livello di difficoltà.

Qwen dimostra alcune capacità base nella produzione di classi e attributi, ma manca di un'effettiva comprensione semantica del dominio. La sua utilità è limitata a contesti in cui si accettano soluzioni parziali e si prevede una revisione sistematica da parte dell'utente.

Sintesi comparativa La Tabella 4.10 fornisce una sintesi qualitativa delle principali caratteristiche osservate nei quattro modelli, secondo sei dimensioni fondamentali.

Tabella 4.10. Confronto qualitativo sintetico tra i modelli

Dimensione	**ChatGPT**	**Gemini**	**DeepSeek**	**Qwen**
Correttezza sintattica	Eccellente	Ottima	Buona	Discreta
Completezza semantica	Eccellente	Buona	Variabile	Debole
Espressività pragmatica	Ottima	Ottima	Sufficiente	Sufficiente
Stabilità dei risultati	Totale	Alta	Bassa	Molto bassa
Sensibilità alla difficoltà	Nulla	Limitata	Elevata	Molto elevata
Affidabilità complessiva	Massima	Elevata	Media	Bassa

Nel complesso, la sperimentazione conferma che la qualità dei diagrammi UML generati da un LLM dipende in larga parte:

- dalla maturità del modello.
- dalla coerenza dell'addestramento sulle istruzioni complesse.
- e dalla sua capacità di ragionare sui vincoli semantici impliciti nel testo.

Capitolo 5

Conclusioni e sviluppi futuri

5.1 Sintesi del lavoro svolto

Nel corso di questo lavoro di tesi è stata affrontata una sfida di crescente rilevanza nel panorama dell'ingegneria del software: la possibilità di utilizzare i Large Language Models (LLM) per generare automaticamente diagrammi UML delle classi a partire da una descrizione testuale in linguaggio naturale. L'obiettivo è stato affrontato con un approccio sperimentale rigoroso, supportato da una solida base teorica e da una struttura metodologica chiara.

Dopo aver analizzato il funzionamento e le potenzialità dei LLM, nonché le tecniche più efficaci di prompt engineering per ottenere output strutturati, è stato costruito un dataset composto da 15 esercizi di modellazione UML realistici, tutti in lingua italiana, selezionati da fonti accademiche e suddivisi per livello di difficoltà secondo un indice composito. Questo dataset ha costituito la base su cui sono stati testati quattro modelli di ultima generazione: GPT-4o, Gemini 2.5 Pro, Qwen 3 e DeepSeek v3.

Ogni modello è stato istruito tramite un prompt uniforme e standardizzato, progettato per produrre un output JSON compatibile con Apollon. I diagrammi generati sono stati valutati secondo sei criteri, due per ciascuna delle dimensioni fondamentali della qualità del modello (sintattica, semantica e pragmatica). Oltre alla valutazione numerica, sono stati analizzati gli errori ricorrenti, gli andamenti rispetto alla difficoltà degli esercizi e l'efficacia delle strategie di prompting adottate.

5.2 Risposte alle domande di ricerca

L'obiettivo della tesi è stato quello di rispondere a quattro domande di ricerca fondamentali, che sono state riprese e discusse nel Capitolo 4. Di seguito si riportano le principali conclusioni raggiunte.

- La prima domanda riguardava l'affidabilità dei LLM nel generare automaticamente diagrammi UML. I risultati mostrano che, sebbene tutti i modelli siano in grado di produrre file formalmente validi e sintatticamente corretti, le differenze tra i modelli

sono significative, specialmente in termini di completezza semantica. GPT-4o e Gemini hanno ottenuto prestazioni complessivamente più elevate e stabili, mentre Qwen 3 e DeepSeek v3 hanno incontrato maggiori difficoltà, specialmente nella corretta rappresentazione delle relazioni e delle gerarchie.

- La seconda domanda verteva sull'impatto della difficoltà dell'esercizio. L'indice di difficoltà composito ha confermato la sua utilità nel classificare gli esercizi in modo coerente con la variabilità delle prestazioni: tutti i modelli hanno mostrato un decadimento nella qualità dell'output al crescere della difficoltà, con peggioramenti evidenti nelle esercitazioni più complesse, dove l'interpretazione dei vincoli impliciti e delle ambiguità testuali risulta più ardua.

- La terza domanda indagava l'effetto delle strategie di prompting. Il confronto tra varianti di prompt, condotto in particolare su GPT-4o, ha evidenziato come un prompt ben strutturato, modulare e dotato di esempi coerenti possa migliorare sensibilmente la qualità dell'output, soprattutto in termini di chiarezza e completezza. Questo suggerisce che il prompting rappresenta un fattore critico di successo per la generazione di modelli strutturati.

- Per la quarta e ultima domanda infine, si è rilevata una tipologia ricorrente di errori, tra cui: relazioni errate o mancanti, generalizzazioni omesse, nomi di classi ambigui, attributi ridondanti o non corrispondenti ai requisiti testuali. Tali errori risultano più frequenti negli esercizi complessi, confermando la necessità di strategie più raffinate per la comprensione semantica da parte dei modelli.

5.3 Contributi della tesi

Questo lavoro fornisce diversi contributi originali al campo della modellazione concettuale automatica e all'analisi dei LLM in contesti educativi e strutturati.

- In primo luogo, è stato realizzato un dataset eterogeneo e realistico di esercizi UML, annotati e valutati secondo criteri di difficoltà. Questo dataset può fungere da base per ulteriori esperimenti futuri, benchmark o attività di addestramento.

- In secondo luogo, è stata adottata una griglia di valutazione articolata e sistematica, che consente di valutare l'output generato dai modelli in modo strutturato, andando oltre la semplice correttezza sintattica.

- In terzo luogo, il lavoro ha messo in luce le potenzialità e i limiti attuali dei LLM, proponendo una lettura critica e dettagliata dei fattori che influenzano la qualità dei risultati. È emerso come il prompting giochi un ruolo determinante nella prestazione dei modelli, ma anche come alcune carenze siano legate alla natura stessa dell'architettura dei LLM e alle loro modalità di generalizzazione.

- Infine, il lavoro ha fornito una base solida per l'impiego di LLM in contesti educativi, proponendo scenari d'uso concreti in cui gli studenti possano ricevere supporto alla modellazione, feedback immediato e strumenti per l'autovalutazione.

5.4 Limiti dello studio

Come ogni studio sperimentale, anche questo lavoro presenta alcuni limiti strutturali e metodologici che è opportuno evidenziare per una lettura critica e consapevole dei risultati ottenuti.

- In primo luogo, il numero di esercizi considerati, sebbene sufficiente per condurre un'analisi comparativa strutturata, rimane limitato. Il dataset è stato costruito con attenzione per garantire varietà e rappresentatività, ma non copre l'intera gamma di situazioni reali che potrebbero emergere nella pratica della modellazione UML. Molti esercizi presenti nella letteratura accademica o nei contesti professionali si riferiscono a domini applicativi specifici, oppure introducono vincoli concettuali particolari non trattati in questa analisi. Questa limitazione implica che i risultati, pur affidabili nel perimetro considerato, non possano essere estesi automaticamente a qualunque scenario modellistico.

- Un secondo limite riguarda la modalità di valutazione adottata. L'assegnazione dei punteggi ai diagrammi generati è stata condotta manualmente, secondo criteri predefiniti e condivisi. Pur essendo stati adottati accorgimenti per garantire coerenza e replicabilità (quali griglie di valutazione dettagliate e una definizione rigorosa dei criteri), la valutazione resta soggetta a un certo grado di interpretazione da parte del valutatore. In particolare, i criteri legati alla validità semantica e alla comprensibilità del diagramma richiedono necessariamente un intervento soggettivo, soprattutto quando la traccia presenta ambiguità o elementi impliciti. Questo introduce un margine di variabilità, inevitabile in assenza di strumenti automatizzati di supporto.

- Un ulteriore limite è rappresentato dall'ambito ristretto dell'analisi, focalizzato unicamente sui diagrammi delle classi. Tale scelta è motivata dalla centralità che questo tipo di diagramma riveste nella modellazione concettuale, nonché dalla disponibilità di strumenti di validazione formale del loro contenuto. Tuttavia, il linguaggio UML comprende numerosi altri tipi di diagrammi, come quelli di sequenza, di attività o di stati, che presentano caratteristiche diverse sia in termini di struttura che di contenuto informativo. Concentrarsi esclusivamente sui class diagram ha permesso di circoscrivere l'esperimento e mantenerne la coerenza interna, ma ha al contempo escluso altre dimensioni della modellazione che potrebbero rivelarsi critiche nel contesto della generazione automatica.

- Infine, lo studio è stato condotto interamente in lingua italiana, e ha utilizzato un unico formato di output: la rappresentazione JSON compatibile con il tool Apollon. Questa scelta, dettata dalla necessità di garantire uniformità tra i casi

di test e facilitare l'analisi automatica dei risultati, comporta tuttavia una ridotta generalizzabilità delle conclusioni. I modelli linguistici di grandi dimensioni possono infatti comportarsi in modo differente a seconda della lingua di input, soprattutto quando si tratta di comprendere concetti specialistici o ambigui. Inoltre, l'adesione a un singolo standard di output limita la valutazione alle sue specificità sintattiche, trascurando eventuali problemi che potrebbero emergere con formati alternativi (ad esempio, PlantUML, XMI, DSL personalizzate).

5.5 Sviluppi futuri

I risultati ottenuti nel presente studio aprono numerosi spunti per approfondimenti e ampliamenti futuri, sia dal punto di vista metodologico che applicativo.

- Un primo sviluppo riguarda l'estensione del dataset. L'inclusione di un numero maggiore di esercizi, provenienti da domini applicativi eterogenei (es. sanità, logistica, amministrazione pubblica), permetterebbe di testare la generalizzabilità dei risultati ottenuti. Inoltre, l'inserimento di tracce con livelli di ambiguità controllata consentirebbe di analizzare con maggiore precisione la capacità dei modelli LLM di gestire incertezze concettuali, ambivalenze terminologiche e descrizioni parziali, elementi tipici della modellazione concettuale nella pratica.

- Un secondo asse di sviluppo riguarda l'esplorazione di altri tipi di diagrammi UML oltre ai class diagram. Diagrammi di sequenza, attività o stati, pur essendo più complessi da rappresentare in formato testuale strutturato, rappresentano una parte fondamentale della modellazione dinamica dei sistemi e porrebbero nuove sfide sia in termini di rappresentazione che di valutazione automatica.

- Un terzo ambito di interesse è legato all'evoluzione della strategia di prompting. La progettazione iterativa dei prompt ha dimostrato un impatto diretto sulla qualità degli output generati. Futuri lavori potrebbero esplorare in modo sistematico il legame tra struttura del prompt, capacità del modello e tipologia del compito assegnato, anche attraverso l'impiego di tecniche di ablation study o l'inserimento controllato di esempi contrastivi all'interno del contesto.

- Particolarmente rilevante, infine, è il perfezionamento del sistema di valutazione adottato. La griglia utilizzata, pur offrendo una struttura rigorosa e replicabile, presenta alcune limitazioni in termini di granularità, pesatura dei criteri e dipendenza dal giudizio soggettivo. In futuro, si potrebbe introdurre una scala di valutazione più fine, capace di cogliere sfumature intermedie tra gli attuali valori discreti, migliorando così la capacità discriminativa della griglia. Inoltre, l'attribuzione di pesi differenti ai criteri — ad esempio assegnando maggiore rilevanza alla correttezza semantica rispetto alla sola validità sintattica — permetterebbe di riflettere in modo più realistico l'impatto effettivo di ciascuna dimensione sulla qualità complessiva del diagramma.

- Per ridurre la componente soggettiva della valutazione, potrebbe essere utile affiancare alla valutazione manuale un sistema di scoring automatizzato, basato su metriche computazionali (ad esempio, misure di distanza semantica tra grafi, matching tra strutture JSON, o metriche ontologiche). Tali strumenti non sostituirebbero il giudizio umano, ma lo integrerebbero, offrendo un supporto oggettivo e scalabile, particolarmente utile in scenari di valutazione massiva o ripetuta.

- Infine, una possibile linea di ricerca trasversale riguarda l'adattabilità del sistema a contesti multilingua e a formati di output alternativi. Sperimentare gli stessi modelli con tracce in inglese, francese o spagnolo, oppure confrontare la qualità dei diagrammi generati in formato JSON con quelli prodotti in linguaggi visuali alternativi (come PlantUML o DSL specializzate), potrebbe fornire informazioni preziose sulla robustezza e sull'adozione internazionale delle soluzioni proposte.

Nel complesso, lo studio svolto rappresenta un primo passo verso una comprensione più sistematica delle potenzialità e dei limiti dei LLM nella generazione automatica di modelli UML. Gli sviluppi futuri qui delineati mirano a rafforzare ulteriormente la validità scientifica dei risultati, espandere il perimetro di applicazione e affinare gli strumenti metodologici messi in campo.

5.6 Conclusione finale

L'elaborato ha dimostrato che i Large Language Models, se opportunamente istruiti, sono in grado di generare rappresentazioni UML delle classi partendo da descrizioni testuali in linguaggio naturale, con un livello di accuratezza sufficiente per essere valutato secondo criteri formalizzati. Le prestazioni ottenute dai modelli analizzati variano in funzione della loro architettura, delle dimensioni del contesto e della qualità del prompt fornito, ma mostrano comunque un'evoluzione significativa.

La metodologia adottata ha permesso di condurre una valutazione comparativa su quattro modelli recenti, tramite un dataset realistico e una griglia di analisi articolata su tre dimensioni. I risultati ottenuti evidenziano una correlazione tra la difficoltà dell'esercizio e il peggioramento delle performance, nonché una forte incidenza delle tecniche di prompt engineering sulla qualità dell'output.

L'approccio proposto può essere replicato per analizzare versioni successive dei modelli o per estendere la valutazione ad altri tipi di diagrammi. I risultati ottenuti non costituiscono un indicatore assoluto delle capacità dei modelli, ma offrono una base di riferimento utile per successive analisi.

In sintesi, il lavoro ha fornito evidenze empiriche sulla fattibilità dell'impiego dei LLM nella generazione assistita di modelli concettuali, delineando al contempo i limiti attuali e le condizioni necessarie per un impiego più esteso in contesti educativi o professionali. Ulteriori approfondimenti saranno necessari per estendere la validità dei risultati, automatizzare la valutazione e migliorare la capacità dei modelli di trattare la componente semantica implicita nei requisiti.

Appendice A

Dataset e Analisi

La raccolta con i quindici esercizi (per ciascun esercizio sono forniti il testo, la soluzione proposta, la soluzione del LLM JSON e il diagramma UML generato dall'UML Modeler) e i file CSV con tutte le valutazioni sono disponibili al seguente link: https://figshare.com/articles/dataset/ANALISI_ESERCIZI/29530661

Bibliografia

[1] Ali Almusawi, Maria Kutar, and Simon Rogerson. Evaluating the visual syntax of uml: An analysis of the cognitive effectiveness of the uml family of diagrams. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 12(2):502–510, 2021.

[2] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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.

[3] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[4] Ahmad Fauzan et al. Automated grading of uml class diagrams using semantic similarity. In *Proc. of ICSE*, 2021.

[5] Alessio Ferrari, Simone Bugiani, and Paola Spoletini. Chatgpt in software engineering: A study on uml diagram generation from requirements. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, pages 321–327. IEEE, 2023.

[6] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[7] J. Gómez-Abajo et al. Automated generation and correction of diagram-based exercises for moodle. *Computers & Applications in Engineering Education*, 2023.

[8] Rohit Jain et al. Uml model assessment tools for education. In *ICSEA 2015*, 2015.

[9] Stephan Krusche, Maximilian Weber, Lukas Alberts, Julian Bräunlein, Matthias Fejes, and Loris Reiff. An interactive learning method to engage students in modeling. In *Proceedings of the 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 29–38. ACM, 2020.

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

[11] H. Liu et al. Usability and support tools for uml modeling. *Journal of Systems and Software*, 2018.

[12] Object Management Group. *Object Constraint Language (OCL), Version 2.4*, 2014.

[13] Object Management Group. *OMG Unified Modeling Language (OMG UML), Version 2.5.1*, 2017.

[14] Javier Romero et al. A tool to automate uml diagram assessment. In *International Conference on Advanced Learning Technologies*, 2015.

[15] M. Santos et al. Exploring students' difficulties in conceptual modeling tasks. *Journal of Information Systems Education*, 2020.

[16] S. Tanaka et al. Common mistakes in conceptual modeling by novice analysts. In *Conference on Software Engineering Education and Training*, 2019.

[17] Mohsin Umar, Abdul Hameed, et al. Evaluation of automatically generated uml diagrams: A study on correctness and completeness. In *2021 IEEE International Conference on Software Engineering Education and Training (CSEE&T)*, pages 49–58. IEEE, 2021.

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

[19] Nicolas Wehrhahn. Automatic grading of uml diagrams using multimodal large language models. Master's thesis, Technical University of Munich (TUM), 2025. Available at TUM library or on request.

[20] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[21] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[22] Jon Whittle et al. Model-driven prompt engineering. In *Proceedings of the 27th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2024.

[23] Jiefu Zhuo et al. Promptware engineering: Towards software engineering principles for prompt design. *arXiv preprint arXiv:2401.09107*, 2024.