



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

TESI DI LAUREA

# **Generazione Automatica di User Stories a partire da Requisiti Software tramite Prompt Engineering**

RELATORI

Prof. Fabio Palomba

Dott. Francesco Casillo

CANDIDATO

**Giovanni Borrelli**

Matricola: 0522501807

Anno Accademico 2024-2025

*Questa tesi è stata realizzata nel*

sesa<sup>lab</sup>  
SOFTWARE ENGINEERING  
SALERNO

*"Da quando ho imparato a camminare mi piace correre."*

*- Friedrich W. Nietzsche*

## Abstract

Negli sviluppi software moderni, la trasformazione automatica dei requisiti software in user stories rappresenta una fase critica per garantire coerenza, chiarezza e tracciabilità delle funzionalità. Tuttavia, i metodi tradizionali di generazione automatica, insieme alle tecniche classiche di ML/NLP, presentano limiti significativi: non catturano completamente le informazioni chiave (“who”, “what” e “why”), richiedono elevati costi computazionali e non producono output immediatamente utilizzabili. Inoltre, un approccio moderno basato Large Language Models, seppur promettente, presenta ancora delle sfide importanti, come l’ottimizzazione dei prompt e la mancanza di consenso sul miglior LLM.

Questo lavoro propone un approccio basato su Large Language Models per automatizzare la generazione di user stories a partire dai requisiti software. Sono state analizzate diverse tecniche di prompt engineering, tra cui few-shot, meta-prompting e prompt directional stimulus, e confrontati due modelli LLM, LLaMA e Mistral, utilizzando metriche automatiche standard di valutazione testuale (BLEU, ROUGE, METEOR e BERTScore) e un’analisi qualitativa basata su esempi corretti, borderline ed errati.

I risultati mostrano che i LLM producono user stories coerenti, complete e semanticamente corrette, superando le limitazioni dei metodi precedenti. Alcune tecniche di prompt, in particolare few-shot, migliorano ulteriormente le performance, mentre LLaMA ha evidenziato vantaggi leggermente superiori rispetto a Mistral, soprattutto nei casi più complessi o sequenziali. Mistral tende a omettere il “why” più spesso e mostra maggiore ambiguità nella scelta degli attori. Complessivamente, il lavoro conferma la validità e la promettente applicabilità dei LLM, indicando che questo approccio rappresenta la soluzione più efficace per automatizzare la conversione dei requisiti software in user stories.

---

## Indice

---

<b>Elenco delle Figure</b>	<b>iv</b>
<b>Elenco delle Tabelle</b>	<b>v</b>
<b>1 Introduzione</b>	<b>1</b>
1.1 Contesto Applicativo . . . . .	1
1.2 Motivazioni e Obiettivi . . . . .	3
1.3 Risultati Ottenuti . . . . .	4
1.4 Struttura della tesi . . . . .	6
<b>2 Background e Stato dell'arte</b>	<b>7</b>
2.1 Background . . . . .	7
<b>3 Stato dell'Arte</b>	<b>9</b>
3.0.1 Metodi tradizionali per l'ingegneria dei requisiti . . . . .	9
3.0.2 Tecniche di NLP e ML tradizionali applicate alla trasformazione dei requisiti . . . . .	11
3.0.3 Approcci basati su Large Language Models e AI moderna . .	12
3.0.4 Sintesi e lacune individuate . . . . .	14
<b>4 Metodologia</b>	<b>16</b>
4.1 Research Questions . . . . .	16

4.2	Architettura del progetto . . . . .	18
4.3	Descrizione del Dataset . . . . .	20
4.4	Selezione dei progetti . . . . .	20
4.5	Elaborazione dei dati . . . . .	21
4.5.1	Pulizia e Preprocessing dei Dati . . . . .	21
4.6	LLM utilizzati . . . . .	23
4.6.1	Parametri di configurazione . . . . .	24
4.7	Considerazioni sul rischio di <i>data leakage</i> . . . . .	25
4.8	Prompt Engineering . . . . .	26
4.8.1	Zero-shot . . . . .	28
4.8.2	Few-shot . . . . .	29
4.8.3	Chain of Thought . . . . .	30
4.8.4	Self-consistency . . . . .	32
4.8.5	Prompt Generate Knowledge . . . . .	33
4.8.6	Prompt Chaining . . . . .	34
4.8.7	Prompt Directional Stimulus . . . . .	35
4.8.8	Meta-prompting . . . . .	36
4.8.9	Reflection . . . . .	38
4.8.10	Prompt Ensemble . . . . .	39
4.9	Metriche di Validazione . . . . .	40
<b>5</b>	<b>Risultati</b>	<b>41</b>
5.1	RQ1. Confronto tra tecniche di Prompt Engineering . . . . .	42
5.1.1	Analisi quantitativa . . . . .	42
5.1.2	Confronto tra metodi . . . . .	45
5.1.3	Esempi qualitativi . . . . .	48
5.1.4	Interpretazione dei risultati . . . . .	52
5.1.5	Risposta alla RQ1 . . . . .	53
5.2	RQ2. Confronto tra LLM diversi . . . . .	54
5.2.1	Esempi qualitativi . . . . .	55
5.2.2	Interpretazione dei risultati . . . . .	58
5.3	RQ3. Validità dell'utilizzo di LLM . . . . .	59

<b>6</b>	<b>Minacce alla validità</b>	<b>61</b>
6.1	Validità interna . . . . .	61
6.2	Validità esterna . . . . .	62
6.3	Validità di costruzione . . . . .	62
6.4	Validità di conclusione . . . . .	63
<b>7</b>	<b>Conclusioni</b>	<b>64</b>
	<b>Bibliografia</b>	<b>66</b>

---

## Elenco delle figure

---

4.1	Pipeline di generazione e valutazione delle user stories. . . . .	18
4.2	Distribuzione delle righe nel dataset prima della fase di pulizia: numero totale di righe, righe duplicate e righe con valori nulli nella colonna <code>sentence</code> . . . . .	22
5.1	BLEU Score medio per tecnica. . . . .	42
5.2	ROUGE Scores medi per tecnica. . . . .	42
5.3	METEOR Score medio per tecnica. . . . .	43
5.4	BERTScore medio per tecnica. . . . .	43
5.5	Confronto tra LLaMA (blu) e Mistral (rosso) per le metriche. . . . .	54



---

## Elenco delle tabelle

---

4.1	Caratteristiche principali dei progetti selezionati dal dataset PURE. .	20
5.1	Valori medi delle metriche per ciascuna tecnica di prompt engineering.	44
5.2	Confronto tra Prompt Ensemble e Few-shot sulle metriche di valutazione.	45

# CAPITOLO 1

---

## Introduzione

---

### 1.1 Contesto Applicativo

Il **ciclo di vita del software** rappresenta l'insieme strutturato delle fasi che conducono dalla nascita di un'idea progettuale alla realizzazione, manutenzione ed evoluzione di un sistema informatico. Tra i modelli più noti si possono citare il **modello a cascata**, che prevede una sequenza lineare di fasi ben definite (analisi, progettazione, implementazione, test e manutenzione), e i **modelli incrementali o agili**, nei quali lo sviluppo avviene in iterazioni più brevi e flessibili, favorendo un continuo adattamento ai cambiamenti dei requisiti e del contesto applicativo.

In tutti questi approcci, un ruolo centrale è svolto dall'**ingegneria dei requisiti**, ovvero la disciplina che si occupa dell'identificazione, documentazione e gestione dei requisiti di un sistema software, con l'obiettivo di garantire che ciò che viene sviluppato risponda effettivamente alle esigenze degli utenti e degli altri attori coinvolti.

L'**ingegneria dei requisiti** comprende un insieme strutturato di attività fondamentali:

- **Elicitation**: raccolta delle esigenze e delle aspettative attraverso interviste,

workshop, osservazioni o analisi di documentazione esistente.

- **Analisi:** studio di fattibilità, risoluzione di conflitti e prioritarizzazione dei requisiti.
- **Specifica:** formalizzazione dei requisiti in documenti chiari, consistenti e non ambigui.
- **Validazione:** verifica della correttezza e della completezza dei requisiti rispetto agli obiettivi.

Lo scopo ultimo di queste attività è tradurre i bisogni degli stakeholder, spesso espressi in forma informale, in specifiche chiare, consistenti e verificabili, che possano costituire la base per le fasi successive dello sviluppo.

L'impatto dell'ingegneria dei requisiti sul ciclo di vita del software è significativo: requisiti ben definiti riducono il rischio di errori concettuali e ambiguità, facilitano le attività di progettazione e implementazione, e rendono più mirate ed efficaci le fasi di verifica e validazione. In un modello sequenziale come quello a cascata, una scarsa analisi dei requisiti può comportare gravi costi di revisione nelle fasi finali; nei modelli agili, invece, una gestione continua e dinamica dei requisiti permette di accogliere i cambiamenti e ridurre gli sprechi. In ogni caso, la qualità della fase di ingegneria dei requisiti influenza direttamente la qualità del prodotto finale, rappresentando un fattore determinante per il successo o il fallimento di un progetto software.

Negli sviluppi software moderni, la gestione efficace dei requisiti riveste un ruolo cruciale per garantire che le funzionalità implementate soddisfino le reali necessità degli utenti. I *requisiti software*, spesso espressi in documenti formali o in linguaggio naturale, descrivono ciò che un sistema dovrebbe fare, ma risultano talvolta lunghi, ambigui o poco uniformi [1, 2].

Per facilitare la pianificazione e l'organizzazione dello sviluppo, tali requisiti vengono comunemente trasformati in *user stories*, unità di lavoro concise e strutturate che descrivono una funzionalità dal punto di vista dell'utente finale. Una *user story* è generalmente composta da tre elementi principali:

- **Chi:** il ruolo o attore che utilizzerà la funzionalità;

- **Cosa:** l'obiettivo o azione che l'attore intende compiere;
- **Perché:** il beneficio o valore che ne deriva.

Ad esempio: *“Come utente registrato, voglio poter reimpostare la mia password, in modo da recuperare l'accesso al sistema in caso di smarrimento.”*

Attualmente, la traduzione dei requisiti in *user stories* avviene **manualmente**, richiedendo un notevole dispendio di tempo e risorse, oltre a introdurre il rischio di incoerenze, ambiguità o differenze stilistiche tra i vari redattori. L'**automazione di questo processo** rappresenta quindi un'opportunità significativa: consente di accelerare la conversione dei requisiti, ridurre gli errori soggettivi e garantire maggiore uniformità e standardizzazione nelle descrizioni delle funzionalità [3].

Un approccio automatico non solo migliora l'efficienza della fase di *requirements engineering*, ma si inserisce anche in maniera strategica nel **ciclo di vita del software**: la disponibilità immediata di *user stories* coerenti e strutturate supporta infatti la **pianificazione del lavoro**, facilita l'**implementazione e il testing** delle funzionalità e assicura una più chiara **tracciabilità** tra i requisiti iniziali e il prodotto sviluppato. In questo modo, l'automazione contribuisce a ridurre tempi e costi, aumentando al contempo la qualità complessiva del processo di sviluppo.

## 1.2 Motivazioni e Obiettivi

Negli ultimi anni, diverse metodologie sono state proposte per supportare la trasformazione dei requisiti software in *user stories*. I metodi tradizionali si basano su regole, checklist, casi d'uso e formalismi come UML o ontologie, ma presentano limiti significativi in termini di scalabilità, coerenza, tracciabilità e dipendenza dall'esperienza degli analisti. Approcci successivi basati su tecniche di NLP e ML tradizionali, come reti neurali artificiali (ANN), support vector machines (SVM) e algoritmi genetici, hanno introdotto automazione e capacità predittive, tuttavia non garantiscono la generazione completa e coerente delle *user stories* e spesso non catturano il contesto motivazionale (*Why*).

Più recentemente, l'avvento dei Large Language Models (LLM) ha aperto nuove possibilità, consentendo la generazione automatica di *user stories*, il supporto alle

pratiche Agile e l'automazione di molte attività della Requirements Engineering. Nonostante questi progressi, rimangono alcune lacune: mancanza di dataset standardizzati, assenza di valutazioni comparative sulle tecniche di prompt engineering e nessun consenso sul modello LLM più efficace per la generazione di user stories coerenti e complete.

Tali osservazioni evidenziano la necessità di sviluppare approcci metodologici in grado di:

- migliorare l'efficienza e la scalabilità nella trasformazione dei requisiti in user stories;
- ridurre l'intervento manuale e le variabilità dovute a interpretazioni soggettive;
- garantire coerenza, completezza e tracciabilità tra requisiti e user stories generate;
- analizzare come fattori come la formulazione dei prompt e le caratteristiche dei modelli LLM influenzino la qualità delle user stories.

© **Il nostro Obiettivo.** Sviluppare un approccio metodologico basato su LLM e tecniche di prompt engineering, in grado di generare automaticamente user stories coerenti, complete e semanticamente corrette a partire dai requisiti software. Pur essendo presenti lavori precedenti sull'utilizzo di LLM, la tesi si concentra su aspetti ancora poco esplorati, quali l'impatto della formulazione dei prompt, la valutazione comparativa di diverse strategie di prompt engineering e l'analisi della qualità semantica delle user stories generate, oltre ad affrontare le principali limitazioni dei metodi tradizionali e delle tecniche NLP/ML classiche.

## 1.3 Risultati Ottenuti

Dopo un'attenta analisi dello stato dell'arte, il metodo più promettente per la generazione automatica di user stories a partire da requisiti software si è rivelato essere l'utilizzo di Large Language Models (LLM). Nel nostro studio, sono state

applicate diverse tecniche di prompt engineering, tra cui **Few-shot**, **Zero-shot**, **Chain-of-Thought**, **Prompt Chaining**, **Meta-Prompting**, **Prompt Directional Stimulus**, **Self-consistency** e **Reflection**, con l'obiettivo di valutare l'impatto della struttura del prompt sulle performance dei modelli. Inoltre, è stato confrontato il comportamento di due modelli differenti, **LLaMA 3.1 8B** e **Mistral 7B**, per comprendere le differenze nella capacità di generazione automatica di user stories.

I risultati quantitativi, misurati attraverso metriche automatiche come **BLEU**, **ROUGE**, **METEOR** e **BERTScore**, indicano che la tecnica **Few-shot** si conferma la più efficace, con valori medi di BLEU e METEOR superiori a 0.55 e BERTScore oltre 0.70, mentre Zero-shot e le tecniche sequenziali come Chain-of-Thought mostrano performance più basse e maggiore variabilità. L'analisi qualitativa ha confermato questi trend: Few-shot genera user stories coerenti, complete e grammaticalmente corrette, con attori, funzionalità e obiettivi generalmente corretti, mentre in Zero-shot e tecniche complesse sono frequenti omissioni o incoerenze, in particolare nella definizione del "why" e nella scelta degli attori.

Il confronto tra modelli ha evidenziato che **LLaMA** tende a ottenere metriche leggermente superiori rispetto a Mistral, soprattutto nelle tecniche più complesse e sequenziali, suggerendo una migliore gestione del ragionamento passo-passo e della coerenza tra attore, funzionalità e obiettivo. LLaMA ha mostrato performance leggermente superiori rispetto a Mistral, con uno scarto medio di circa il 3% su tutte le metriche considerate; la differenza complessiva tra i due modelli risulta contenuta, confermando che entrambi possono essere considerati strumenti validi per la generazione automatica di user stories.

In sintesi, il lavoro dimostra la validità dell'utilizzo di LLM per questo compito, evidenziando che l'inclusione di esempi contestuali (Few-shot) migliora sensibilmente le performance, mentre tecniche più complesse o prive di esempi mostrano limiti chiari. I risultati ottenuti mostrano inoltre un miglioramento rispetto ai lavori già presenti in letteratura, sia in termini di coerenza delle user stories generate sia nella loro completezza informativa.

## 1.4 Struttura della tesi

Nel *capitolo 2*, si trova una introduzione ai requisiti software, alle user stories e alla loro relazione. La sezione sullo stato dell'arte illustra i principali lavori presenti in letteratura e suggerisce la direzione di ricerca più incoraggiante da seguire.

Nel *capitolo 3*, vengono introdotte le domande di ricerca, il dataset utilizzato e le tecniche di prompt engineering utilizzate.

Nel *capitolo 4* sono presenti i risultati ottenuti dai modelli, e le risposte alle domande di ricerca.

Nel *capitolo 5*, è presente una descrizione delle potenziali minacce alla validità e delle relative strategie di mitigazione.

Nel *capitolo 6*, è presente una descrizione di ciò che è stato raggiunto e migliorato, e considerazioni in merito a dei possibili sviluppi nel futuro.

---

### Background e Stato dell'arte

---

#### 2.1 Background

I Requisiti Software rappresentano le specifiche funzionali e non funzionali che un sistema deve soddisfare per rispondere correttamente alle esigenze degli utenti e degli stakeholder. I requisiti funzionali definiscono i comportamenti, le funzioni e i servizi che il software deve fornire, mentre i requisiti non funzionali descrivono vincoli di prestazione, sicurezza, affidabilità, manutenibilità e usabilità. Una definizione chiara e dettagliata dei requisiti è fondamentale per guidare l'analisi, la progettazione e lo sviluppo del software [4], riducendo il rischio di errori costosi [5] e garantendo che il prodotto finale sia efficace, efficiente e di alta qualità [6]. I requisiti costituiscono la base per la verifica e la validazione del sistema, assicurando che esso soddisfi le aspettative degli utenti e rispetti gli standard progettuali.

Le User Stories sono brevi descrizioni, centrali sull'utente, di funzionalità o comportamenti che il software deve implementare. Generalmente seguono la struttura: *"Come [tipo di utente], voglio [azione] così che [beneficio]"* [7]. Esse rappresentano uno strumento chiave nelle metodologie Agile [8], poiché facilitano la comunicazione tra stakeholder e team di sviluppo, supportano la pianificazione iterativa e consentono



di suddividere i requisiti in unità gestibili e testabili.

I requisiti software forniscono la descrizione dettagliata di ciò che il sistema deve fare e dei vincoli che deve rispettare, mentre le user stories traducono tali requisiti in scenari concreti, focalizzati sul valore per l'utente [9]; In altre parole, i requisiti stabiliscono il *cosa* e il *come* tecnico del sistema [10], e le user stories rappresentano il *come* operativo [11], utile al team di sviluppo per implementare e verificare le funzionalità in modo coerente ed efficace.

Il passaggio dai requisiti software alle user stories avviene attraverso un processo di analisi e sintesi che permette di individuare le funzionalità essenziali del sistema e tradurle in descrizioni brevi, chiare e orientate all'utente [12]. In questo modo i requisiti vengono resi più concreti e immediatamente utilizzabili nella pianificazione dello sviluppo, assicurando che ogni funzionalità sia associata a un obiettivo specifico e verificabile.

## CAPITOLO 3

---

### Stato dell'Arte

---

Lo stato dell'arte è organizzato in tre filoni principali, in modo da mostrare l'evoluzione delle tecniche utilizzate nell'ingegneria dei requisiti:

- **Metodi tradizionali:** approcci manuali e basati su regole, checklist, casi d'uso, formalismi (UML, ontologie) e altre tecniche pre-NLP;
- **Tecniche di NLP e ML tradizionali:** reti neurali classiche (ANN), support vector machines (SVM), algoritmi genetici e altre metodologie di machine learning applicate alla trasformazione dei requisiti;
- **Approcci basati su Large Language Models e AI moderna:** LLM come GPT, LLaMA, Mistral e tecniche avanzate di prompt engineering per generare automaticamente user stories e supportare pratiche Agile.

#### 3.0.1 Metodi tradizionali per l'ingegneria dei requisiti

Il filone dei metodi tradizionali comprende approcci manuali e basati su regole, checklist, casi d'uso, formalismi come UML o ontologie, e altre tecniche pre-NLP sviluppate per strutturare e guidare il processo di *Requirements Engineering* (RE). Questi approcci hanno avuto un ruolo fondamentale nello standardizzare il lavoro

degli analisti e nel fornire strumenti metodologici per la raccolta, la documentazione e la gestione dei requisiti.

Tuttavia, tali metodi presentano limiti significativi che ne riducono l'efficacia nelle moderne pratiche di sviluppo software. In particolare:

- **Scalabilità limitata:** l'estrazione e la gestione dei requisiti manuale diventano rapidamente ingestibili con l'aumento del numero e della complessità dei requisiti.
- **Coerenza e tracciabilità difficili da garantire:** requisiti formulati in maniera diversa da diversi analisti possono introdurre ambiguità, riducendo la tracciabilità tra requisiti, implementazione e test.
- **Rigidità e lentezza del processo:** modifiche ai requisiti richiedono aggiornamenti manuali estesi, con elevati costi di revisione nelle fasi successive del ciclo di vita.
- **Dipendenza dall'esperienza degli analisti:** la qualità del lavoro dipende fortemente dalle competenze individuali, rendendo il processo soggetto a variabilità e bias soggettivi.

Questi limiti hanno conseguenze dirette sulle attività di sviluppo: errori o omissioni nella fase di RE possono propagarsi alla progettazione e implementazione, generando costi elevati per correzioni tardive, riducendo l'efficienza e compromettendo la qualità del prodotto finale.

È proprio in risposta a queste criticità che si colloca il contributo della presente tesi: l'adozione di tecniche basate su NLP, ML e LLM mira a superare i vincoli dei metodi tradizionali, introducendo automazione, maggiore scalabilità, uniformità nella generazione di user stories e supporto alla tracciabilità tra requisiti e implementazione.

### 3.0.2 Tecniche di NLP e ML tradizionali applicate alla trasformazione dei requisiti

In questo filone rientrano approcci basati su reti neurali classiche (ANN), support vector machines (SVM), algoritmi genetici (GA) e altre tecniche di machine learning applicate alla *Requirements Engineering* (RE). Questi metodi hanno introdotto automazione e capacità predittive nella trasformazione dei requisiti, ma presentano limitazioni significative, sia dal punto di vista tecnico sia in termini di impatto sulle attività di sviluppo software.

Ad esempio, lavori del 2018 basati su ANN [13, 14] hanno permesso di estrarre il *Who* e il *What* dai requisiti, ma non il *Why*. Questo limite è critico nella RE, poiché la mancanza del contesto motivazionale può portare a user stories incomplete o interpretazioni errate durante la progettazione e lo sviluppo. Inoltre, le ANN richiedono elevate risorse computazionali e tempi di esecuzione lunghi, rendendo il metodo poco scalabile in scenari industriali con grandi quantità di requisiti. Le metriche riportate (precision 52%, recall 50%) evidenziano inoltre una bassa accuratezza, con possibili errori nella generazione delle user stories che potrebbero propagarsi alle fasi successive del ciclo di vita.

Nel 2021, l'utilizzo di SVM [15] ha migliorato i tempi di elaborazione, con metriche di precision e recall più elevate (circa 76%). Tuttavia, le SVM restano classificatori: possono identificare se una frase contiene un attore o una funzione, ma non sono in grado di generare autonomamente user stories complete. Ciò limita l'automazione reale del processo e richiede comunque intervento umano per assemblare le informazioni in forma operativa.

Nel 2023, algoritmi genetici [16] sono stati applicati per estrarre automaticamente elementi tipici dei diagrammi UML, come use case, attore, mittente, destinatario e messaggio. Anche in questo caso, il *Why* non viene catturato, e gli output devono essere definiti a priori, riducendo la flessibilità del metodo. In contesti dinamici e complessi, dove i requisiti possono assumere forme molto diverse, questa limitazione si traduce in una scarsa generalizzabilità.

In generale, le tecniche di NLP e ML tradizionali mostrano diversi punti di forza, come l'introduzione di automazione e la capacità di supportare la classificazione o

l'estrazione di entità dai requisiti, ma i limiti evidenziati impediscono di ottenere una generazione completa, coerente e semanticamente accurata delle user stories. Queste criticità motivano l'adozione di approcci basati su **Large Language Models (LLM)**, in grado di:

- catturare informazioni più ricche e contestuali, inclusi gli aspetti motivazionali (*Why*);
- generare testo coerente e strutturato in forma di user story, riducendo la necessità di intervento umano;
- scalare più facilmente a grandi quantità di requisiti senza compromettere la qualità della produzione.

In questo modo, il contributo della presente tesi si colloca come naturale evoluzione rispetto ai metodi tradizionali di NLP e ML, affrontando alcune delle limitazioni più rilevanti per la RE e migliorando la tracciabilità, coerenza e automazione della trasformazione dei requisiti in user stories.

### 3.0.3 Approcci basati su Large Language Models e AI moderna

Il terzo filone riguarda i contributi più recenti basati su **Large Language Models (LLM)**, come GPT, LLaMA e Mistral, e sulle tecniche avanzate di *prompt engineering*. Questi approcci rappresentano un importante avanzamento nella *Requirements Engineering* (RE), poiché permettono di automatizzare attività che tradizionalmente richiedevano intervento umano e competenze specifiche.

Gli LLM consentono di:

- generare automaticamente *user stories* a partire da requisiti software, traducendo informazioni spesso espresse in linguaggio naturale in artefatti pronti per guidare lo sviluppo;
- supportare le pratiche Agile, facilitando la gestione del backlog, la pianificazione iterativa e la tracciabilità tra requisiti e implementazione;
- automatizzare molte attività della RE, riducendo tempi e costi, migliorando la scalabilità e aumentando la coerenza e la qualità dei risultati.

Studi recenti hanno mostrato l'efficacia degli LLM nella generazione automatica di user stories e nel supporto alle pratiche Agile. Ad esempio, strumenti basati su GPT-4 hanno dimostrato la capacità di creare user stories in formato standardizzato, integrabili con strumenti di gestione progetti come Jira [17]. Altri lavori hanno evidenziato come agenti LLM possano migliorare la qualità delle user stories prodotte in team Agile [18].

Le **tecniche di prompt engineering** consistono nella progettazione e ottimizzazione dei testi di input forniti a un LLM, al fine di guidarne il comportamento e ottenere output più accurati e coerenti. Nel contesto della Requirements Engineering, queste tecniche sono fondamentali per la generazione automatica di user stories, poiché la qualità del risultato dipende fortemente da come viene formulato il prompt. Alcuni approcci comuni includono:

- **Zero-shot:** il modello riceve direttamente il prompt senza esempi precedenti;
- **Few-shot:** il prompt include alcuni esempi di input-output per guidare il modello nella generazione;
- **Chain-of-thought:** il prompt incoraggia il modello a ragionare passo-passo, migliorando la coerenza e la completezza dell'output;

L'uso appropriato di queste tecniche consente di aumentare la qualità delle user stories generate, migliorare la coerenza con i requisiti originali e ridurre la necessità di intervento umano, rappresentando un elemento chiave del contributo della tesi.

Ricapitolando, nonostante tutti questi vantaggi, rimangono delle sfide aperte:

- **Standardizzazione dei dataset:** la mancanza di dataset pubblici e uniformi limita la possibilità di confronti sistematici e valutazioni oggettive delle performance;
- **Ottimizzazione dei prompt:** la qualità delle user stories generate dipende fortemente dalla formulazione dei prompt, e non esiste ancora un consenso su metodologie ottimali;

- **Nessun consenso sul miglior LLM:** l'assenza di consenso su quale LLM risulti più efficace per la RE e la generazione di artefatti software.

Il contributo della presente tesi si colloca in questo contesto. La tesi propone un approccio metodologico basato su LLM e tecniche di prompt engineering, finalizzato a generare user stories coerenti, complete e semanticamente corrette a partire dai requisiti software. In particolare, lo studio analizza come diversi fattori, ovvero la formulazione dei prompt, le strategie di prompt engineering e le caratteristiche dei modelli, influiscano sulla qualità delle user stories prodotte. Questo approccio permette di migliorare l'efficienza, la tracciabilità e la qualità del processo di RE, affrontando alcune delle principali limitazioni dei metodi tradizionali e delle tecniche NLP/ML classiche.

### 3.0.4 Sintesi e lacune individuate

Dall'analisi dei tre filoni principali emerge come la letteratura attuale presenti progressi significativi, ma anche limiti sostanziali che giustificano la ricerca proposta in questa tesi.

I metodi tradizionali hanno fornito fondamenta solide per la *Requirements Engineering*, ma mostrano problemi di scalabilità, coerenza, tracciabilità e dipendenza dall'esperienza degli analisti. Le tecniche di NLP e ML "classiche" hanno introdotto automazione e capacità predittive, tuttavia non garantiscono la generazione completa, coerente e semanticamente accurata delle user stories e faticano a catturare il contesto motivazionale (*Why*). Infine, i contributi più recenti basati su LLM offrono strumenti promettenti per automatizzare la creazione di user stories, supportare pratiche Agile e aumentare la qualità dei processi di RE, ma presentano ancora sfide aperte:

1. **Assenza di dataset pubblici standardizzati**, che limita la possibilità di confronti sistematici e valutazioni oggettive dei metodi;
2. **Mancanza di valutazioni comparative sulle tecniche di prompt engineering**, elemento chiave che influenza la qualità delle user stories generate;

3. **Nessun consenso sul modello LLM più efficace** per la generazione automatica di artefatti software e per il supporto alla RE.

Queste lacune motivano il contributo della tesi, che propone un approccio metodologico basato su LLM e tecniche di prompt engineering. La ricerca si concentra sull'analisi dei fattori che influenzano la qualità delle user stories generate e sull'integrazione di tali strumenti nel ciclo di vita della RE, offrendo un miglioramento rispetto ai metodi tradizionali e alle tecniche NLP/ML classiche.



#### 4.1 Research Questions

L'obiettivo principale di questa tesi è sviluppare un approccio metodologico basato su Large Language Models e tecniche di prompt engineering per generare automaticamente user stories coerenti, complete e semanticamente corrette a partire da requisiti software. Dall'analisi dello stato dell'arte e dei filoni principali nell'Ingegneria dei Requisiti, emergono alcune lacune fondamentali:

- l'assenza di valutazioni comparative sistematiche delle tecniche di prompt engineering nella generazione automatica di user stories;
- la mancanza di confronto tra diversi modelli LLM per individuare quello più efficace nel contesto dell'Ingegneria dei Requisiti;
- la necessità di verificare la validità dei LLM come strumenti per generare user stories complete e coerenti, rispetto ai metodi tradizionali e alle tecniche NLP/ML tradizionali.

A partire da queste considerazioni, le domande di ricerca (**Research Questions, RQ**) sono state formulate per esplorare in maniera strutturata le sfide individuate e

guidare la metodologia della tesi. L'ordine delle RQ riflette la sequenza logica del percorso di indagine: prima si valutano le tecniche di prompt engineering, poi si confrontano i modelli LLM, infine si verifica la validità generale dell'approccio basato su LLM.

**Q RQ<sub>1</sub>.** *Quali tecniche di prompt engineering si rivelano più efficaci nella generazione automatica di user stories?*

Questa domanda si concentra sull'analisi di diverse strategie di prompt e sulla loro influenza sulla qualità delle user stories prodotte. L'efficacia delle tecniche viene valutata utilizzando metriche automatiche come BLEU, ROUGE, METEOR e BERTScore, che permettono di misurare sia la similarità lessicale sia quella semantica tra i requisiti software e le user stories generate.

**Q RQ<sub>2</sub>.** *Tra diversi modelli LLM quale mostra le migliori performance nella generazione automatica di user stories?*

Questa RQ indaga il confronto tra modelli LLM differenti (LLaMA e Mistral), al fine di individuare quale modello offra le migliori prestazioni nel contesto specifico della generazione automatica di user stories. I risultati ottenuti rispondono anche indirettamente alla RQ1, poiché l'efficacia dei prompt può variare in funzione del modello utilizzato.

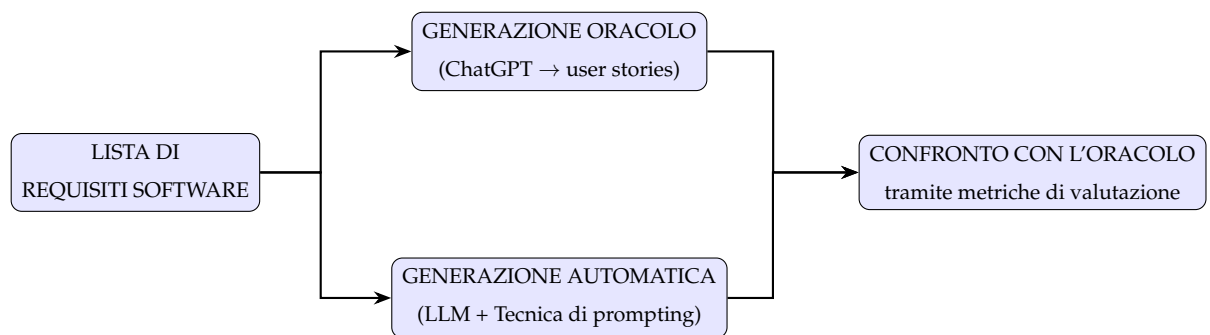
**Q RQ<sub>3</sub>.** *L'utilizzo di LLM è un metodo valido e promettente per generare automaticamente user stories a partire da requisiti software?*

Infine, questa RQ verifica la validità complessiva dell'approccio proposto. Essa integra i risultati delle precedenti domande, fornendo una valutazione globale sulla capacità dei LLM di produrre user stories complete, coerenti e semanticamente corrette, affrontando le principali limitazioni dei metodi tradizionali e delle tecniche NLP/ML classiche.

## 4.2 Architettura del progetto

Per rispondere alle research question formulate, è stata definita una metodologia sperimentale articolata nelle fasi seguenti; In primo luogo, è stato selezionato un dataset di requisiti software, raggruppati per progetto, in modo da garantire coerenza semantica e contesto applicativo. Successivamente, sono state definite le tecniche di prompt engineering e scelti i modelli di linguaggio da utilizzare (LLaMA e Mistral), al fine di esplorare diverse configurazioni di generazione automatica.

Per ciascun progetto, sono state generate due liste di user stories: una tramite *oracolo* (ChatGPT) e una tramite generazione automatica (LLM + tecnica di prompting). Le due liste sono state poi confrontate mediante metriche automatiche (BLEU, ROUGE, METEOR, BERTScore), che permettono di valutare la similarità lessicale e semantica tra i risultati.



**Figura 4.1:** Pipeline di generazione e valutazione delle user stories.

### Input: lista di requisiti software

Per ogni progetto selezionato (CCHIT 2011, LACountyEHR Reqs. e iTRUST), viene considerata la lista completa dei requisiti software estratti dal dataset. Grazie alla presenza dell'identificatore del documento di provenienza, è possibile raggruppare i requisiti per progetto.

### Generazione delle user stories

La generazione delle user stories avviene su due livelli:

- **Oracolo:** utilizzo di ChatGPT per produrre la lista di user stories di riferimento a partire dai requisiti del progetto. L'oracolo fornisce un punto di confronto

standardizzato, poiché non esistono dataset pubblici con requisiti associati a user stories già generate.

- **Automatica:** utilizzo di un LLM selezionato tra LLaMA e Mistral, combinato con una delle 10 tecniche di prompt engineering individuate. Ogni combinazione modello-tecnica viene testata su tutti e tre i progetti, per un totale di 60 esperimenti.

In questa fase, i requisiti software vengono forniti come input all’LLM tramite prompt progettati per guidarne il comportamento, in modo da generare user stories coerenti con la struttura standardizzata *“Come [utente], voglio [azione], così che [beneficio]”*.

### Confronto e valutazione

Le user stories generate automaticamente vengono confrontate con quelle prodotte dall’oracolo tramite metriche automatiche, tra cui:

- **BLEU, ROUGE e METEOR:** valutano la similarità lessicale tra user stories generate e reference.
- **BERTScore:** valuta la similarità semantica tra le due liste di user stories, catturando anche la coerenza e il significato del testo generato.

Questo confronto permette di analizzare sia l’efficacia della tecnica di prompt engineering adottata, sia le performance del modello LLM selezionato, fornendo indicazioni su quali combinazioni siano più promettenti per la generazione automatica di user stories.

Riepilogando, la pipeline del progetto integra l’input dei requisiti, la generazione automatica delle user stories e la valutazione comparativa, costituendo un flusso metodologico completo che consente di rispondere alle domande di ricerca, misurando l’efficacia di diversi approcci basati su LLM e prompt engineering.

## 4.3 Descrizione del Dataset

Per condurre gli esperimenti di questa tesi, è stato necessario individuare un dataset di requisiti software. Su Kaggle è disponibile un dataset<sup>1</sup> costituito da requisiti software estratti da PURE<sup>2</sup> (Public REquirements). PURE è una raccolta di documenti ottenuti dal Web, al cui interno sono presenti requisiti software. Ad ogni requisito è assegnato un ID che indica il documento di provenienza, e ogni documento corrisponde a un progetto differente.

Le colonne aggiuntive del dataset, come *security*, *reliability* e *NFR\_boolean*, forniscono informazioni aggiuntive sui requisiti, indicando rispettivamente se il requisito riguarda sicurezza, affidabilità o se è non-funzionale.

**Tabella 4.1:** Caratteristiche principali dei progetti selezionati dal dataset PURE.

Progetto	N. di requisiti	Dominio	Annotazioni
CCHIT 2011	300	EHR / Sanitario	Security, Reliability, NFR
LACountyEHR Reqs.	1.200	EHR / Sanitario	Security, Reliability, NFR
iTRUST	1.200	EHR / Sanitario	Security, Reliability, NFR

## 4.4 Selezione dei progetti

La selezione dei progetti è avvenuta sulla base di criteri oggettivi, quali la numerosità dei requisiti, completezza delle formulazioni e disponibilità pubblica. Pur essendo disponibili in rete diversi dataset contenenti requisiti software, non avevano ogni requisito associato al documento di provenienza. Questa informazione permette di raggruppare i requisiti per progetto e di generare successivamente le user stories su insiemi coerenti di requisiti, garantendo la correttezza delle valutazioni sperimentali.

I progetti sono stati selezionati secondo criteri di:

<sup>1</sup><https://www.kaggle.com/datasets/computerscience3/public-requirementspure-dataset>

<sup>2</sup><https://zenodo.org/record/1414117>

- **Numerosità:** sufficiente per una valutazione statistica robusta;
- **Completezza:** requisiti correttamente formulati e privi di frasi incomplete;
- **Disponibilità pubblica:** dataset accessibili e documentati.

Sono stati scelti tre progetti particolarmente rappresentativi:

- **CCHIT Certified 2011 Ambulatory EHR Criteria 20110517**, contenente circa 300 requisiti;
- **LACountyEHR Requirements**, contenente circa 1.200 requisiti;
- **iTRUST**, contenente circa 1.200 requisiti.

Tutti e tre i progetti appartengono all'ambito medico, in particolare alle EHR (*Electronic Health Records*), poiché la maggior parte dei documenti del dataset ha natura sanitaria e gli unici progetti che soddisfano i criteri di selezione appartengono al dominio EHR.

## 4.5 Elaborazione dei dati

### 4.5.1 Pulizia e Preprocessing dei Dati

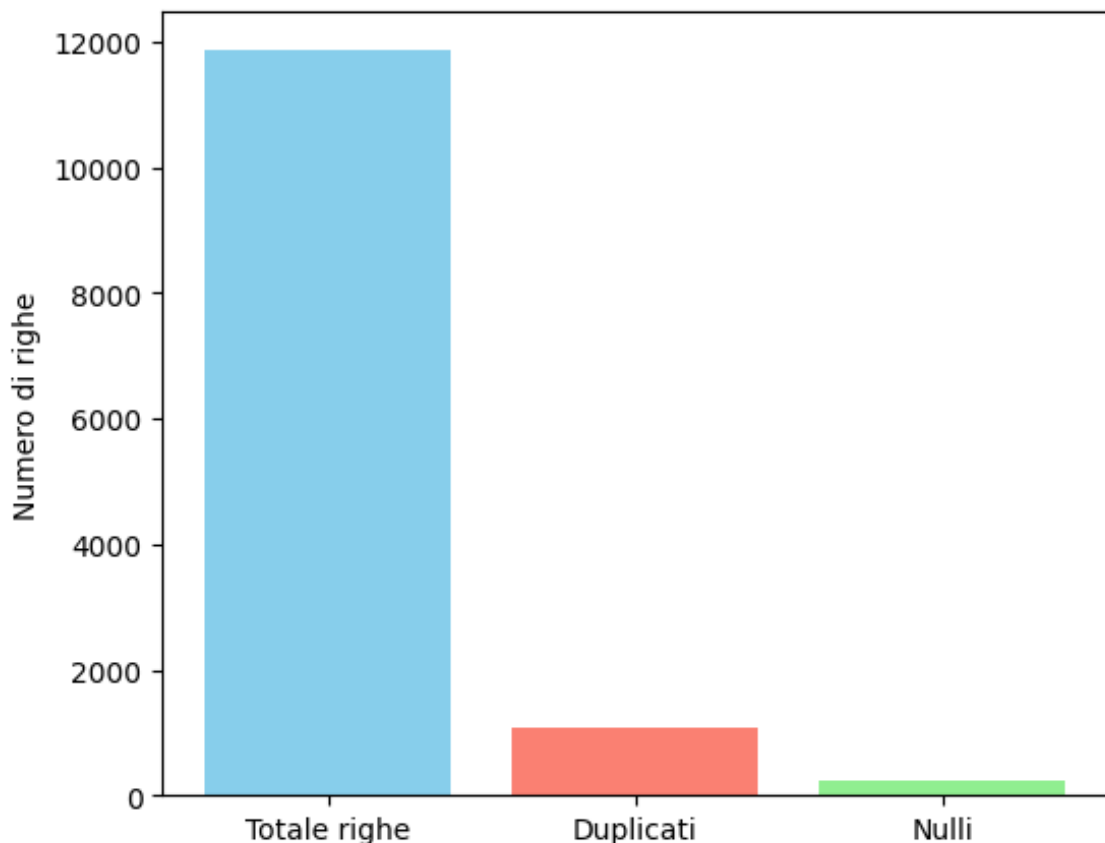
Per garantire la qualità dei dati utilizzati negli esperimenti, è stata effettuata una fase preliminare di **data cleaning** sul dataset. Questa fase comprende le seguenti operazioni:

- **Rimozione dei valori nulli:** sono state eliminate le righe in cui la colonna `sentence` risultava vuota, poiché non contenevano informazioni utili per la generazione delle user stories.
- **Rimozione dei duplicati:** le righe contenenti requisiti identici nella colonna `sentence` sono state eliminate per evitare ridondanza nei dati e garantire una valutazione più accurata delle performance.

- **Gestione della codifica:** il dataset originale presentava problemi di decodifica UTF-8. È stato specificato l'encoding corretto durante il caricamento, permettendo la corretta lettura di tutti i requisiti.

Dopo queste operazioni, il dataset è risultato più coerente e pronto per essere utilizzato nella generazione automatica delle user stories. I risultati della pulizia sono sintetizzati di seguito:

- Numero di righe iniziali pre-pulizia: 11.876
- Numero di duplicati iniziali nella colonna `sentence`: 1.062
- Numero di valori nulli iniziali nella colonna `sentence`: 223
- Numero di righe dopo la pulizia: 10813



**Figura 4.2:** Distribuzione delle righe nel dataset prima della fase di pulizia: numero totale di righe, righe duplicate e righe con valori nulli nella colonna `sentence`.

Inoltre, sono state effettuate operazioni di **normalizzazione** dei requisiti:

- rimozione di caratteri non riconosciuti presenti in molti requisiti;
- rimozione di spazi bianchi all'inizio del requisito;
- aggiunta del punto alla fine di ogni requisito se non presente;
- rimozione di apici, doppi apici o parentesi se racchiudono l'intero requisito, o se vengono aperti e mai chiusi.

Questa fase di pulizia e normalizzazione rappresenta un passaggio fondamentale del preprocessing, poiché garantisce che i dati di input siano completi, non ridondanti e correttamente interpretati dagli algoritmi di generazione automatica.

## 4.6 LLM utilizzati

Per la generazione automatica delle user stories sono stati selezionati due Large Language Models (LLM): **LLaMA 3.1 8B** e **Mistral 7B**. La scelta di questi modelli è stata guidata dai seguenti criteri:

- **Disponibilità open-source:** entrambi i modelli possono essere utilizzati liberamente senza vincoli proprietari, facilitando la replicabilità e la condivisione dei risultati.
- **Compatibilità con risorse limitate:** le dimensioni di 8B e 7B rappresentano un buon compromesso tra capacità di catturare informazioni contestuali complesse e requisiti computazionali gestibili su piattaforme come Google Colab.
- **Integrazione via API:** sia LLaMA sia Mistral possono essere richiamati tramite API, semplificando l'automazione del processo di generazione delle user stories e l'integrazione nella pipeline sperimentale.
- **Adattabilità al dominio applicativo:** grazie alla conoscenza generalista e alla capacità di generare testo coerente e strutturato, i modelli si prestano bene alla trasformazione dei requisiti software in user stories, pur senza richiedere un fine-tuning specifico.



Questa selezione garantisce un equilibrio tra qualità della generazione, efficienza computazionale e riproducibilità dei risultati, fornendo strumenti adeguati per valutare in maniera comparativa le diverse tecniche di prompt engineering implementate nella tesi.

### 4.6.1 Parametri di configurazione

Per garantire la riproducibilità e comprendere le scelte metodologiche, sono riportati i principali parametri di configurazione adottati per la generazione automatica delle user stories con **LLaMA 3.1 8B** e **Mistral 7B**. Per entrambi i modelli, sono stati utilizzati gli stessi valori dei parametri:

- **Temperatura** (`temperature`, default = 0.8): controlla il livello di creatività e variabilità dell'output. Un valore basso (es. 0.2–0.3) tende a produrre testi ripetitivi e deterministici, mentre valori troppo elevati (>1.0) aumentano la variabilità ma riducono la coerenza. Nel nostro caso, la generazione richiede una struttura standardizzata (*"Come [utente], voglio [azione], così che [beneficio]"*); mantenere il valore di default 0.8 ha permesso un buon equilibrio tra variabilità e coerenza.
- **Numero massimo di token generati** (`max_tokens`): impone un limite superiore alla lunghezza dell'output. Il valore di default (128) sarebbe sufficiente per una singola user story, tipicamente breve (15–25 parole). Tuttavia, poiché l'obiettivo era generare lunghe liste di user stories per progetto, si è innalzato il limite al massimo, ovvero 2048 token, ed è stata adottata una strategia di batching, suddividendo in blocchi per poi unire i risultati. Questo approccio permette di sfruttare pienamente la capacità del modello e ridurre il rischio di output troncati.
- **Top-p sampling** (`top_p`, default = 0.9): detta anche nucleus sampling, questa tecnica limita le scelte del modello al sottoinsieme più probabile di token, la cui somma di probabilità non supera  $p$ . Il valore 0.9 rappresenta un compromesso che consente una moderata varietà lessicale senza sacrificare la precisione. Per questo motivo è stato mantenuto.

- **Top-k sampling** (`top_k`, default = 40): restringe lo spazio di campionamento ai  $k$  token più probabili. Il valore di default garantisce un buon equilibrio tra coerenza e diversificazione delle generazioni, ed è stato lasciato invariato.
- **Seed di randomizzazione**: non è stato esplicitamente impostato, pertanto ogni esecuzione può produrre leggere variazioni nell'output, anche a parità di input e parametri. L'obiettivo era valutare le tecniche di prompting in condizioni realistiche, dove la generazione è intrinsecamente non deterministica.

Complessivamente, i parametri adottati rappresentano un compromesso tra la necessità di coerenza strutturale delle user stories e la variabilità necessaria per valutare diverse tecniche di prompting. L'impostazione di `max_tokens` a 2048 è stata la scelta più rilevante per adeguare il processo alle dimensioni del dataset.

## 4.7 Considerazioni sul rischio di *data leakage*

Un aspetto critico nella sperimentazione con Large Language Models (LLM) riguarda la possibile presenza di *data leakage*. Tale fenomeno si verifica quando i modelli, durante il training, hanno avuto accesso a dati simili o identici a quelli utilizzati negli esperimenti, ad esempio requisiti software o user stories provenienti da dataset pubblici. In questo scenario, le prestazioni osservate potrebbero non riflettere realmente la capacità del modello di generalizzare a dati nuovi, ma piuttosto la sua familiarità con esempi già visti.

Nel contesto di questa tesi, sono stati adottati i seguenti accorgimenti e considerazioni:

- **Dataset pubblico**: il dataset PURE utilizzato è accessibile pubblicamente e, sebbene i modelli LLaMA 3.1 8B e Mistral 7B siano stati addestrati su grandi quantità di dati testuali, non è noto se esso sia stato incluso nei rispettivi training set. Questo introduce un potenziale rischio di *overfitting implicito*.
- **Analisi critica dei risultati**: i risultati ottenuti devono essere interpretati con cautela, considerando che valori elevati di similarità o coerenza potrebbero essere parzialmente influenzati da conoscenze pregresse del modello.

- **Riproducibilità e confronti relativi:** nonostante il possibile data leakage, la metodologia comparativa (testando diversi LLM e tecniche di prompt engineering sullo stesso dataset) permette comunque di valutare l'efficacia relativa delle diverse strategie, minimizzando l'impatto sulla validità dei confronti interni.
- **Trasparenza metodologica:** la discussione sul potenziale data leakage è riportata esplicitamente per garantire chiarezza sui limiti dell'esperimento e permettere al lettore di contestualizzare le prestazioni dei modelli.

Questa riflessione evidenzia come la sperimentazione con LLM debba sempre considerare possibili interferenze dovute a dati già noti, soprattutto quando si utilizzano dataset pubblici o modelli pre-addestrati, e rappresenta un elemento fondamentale per una valutazione critica e scientificamente rigorosa dei risultati.

## 4.8 Prompt Engineering

Il prompt engineering [19] è l'insieme di tecniche volte a progettare in modo accurato gli input forniti a un modello linguistico, con l'obiettivo di guidarne il comportamento e migliorare la qualità delle risposte. A differenza di un'interazione *naïve* [20], in cui si formula una domanda in linguaggio naturale e si attende la risposta, il prompt engineering sfrutta schemi, esempi e strategie di guida che aiutano il modello a produrre risultati più coerenti, chiari e vicini alle esigenze applicative. Nel contesto di questa tesi, il prompt engineering è stato applicato per trasformare requisiti software scritti in linguaggio naturale in user stories. Ogni tecnica di prompt engineering è stata applicata a tutti e tre i progetti, con l'obiettivo di calcolare successivamente la media dei risultati. Infine, per generare le user stories a partire dai requisiti software, è stato utilizzato ChatGPT come oracolo, interrogato tramite il sito ufficiale<sup>3</sup>.

Di seguito sono riportate le principali tecniche<sup>4</sup> di prompt engineering esplorate:

1. **Zero-shot:** il modello affronta il compito senza esempi preliminari, basandosi unicamente sulla propria conoscenza pre-addestrata.

---

<sup>3</sup><https://chatgpt.com/>

<sup>4</sup><https://www.promptingguide.ai/it/techniques>

2. **Few-shot**: il modello riceve un numero limitato di esempi prima di eseguire il compito, così da imitare lo schema fornito.
3. **Chain of Thought**: il modello esplicita i passaggi logici intermedi per giungere alla risposta, migliorando la coerenza del ragionamento.
4. **Self-consistency**: combina approccio few-shot e chain of thought, fornendo esempi e incoraggiando il modello a ragionare per step, così da produrre risposte più affidabili.
5. **Prompt Generate Knowledge**: sfrutta la conoscenza pregressa del modello per arricchire la trasformazione in user stories con contesto aggiuntivo.
6. **Prompt Chaining**: suddivide la generazione in passaggi sequenziali (ad esempio attore, funzionalità, obiettivo) per comporre gradualmente una user story completa.
7. **Prompt with Directional Stimulus**: introduce indizi e linee guida per orientare la risposta del modello verso una forma desiderata.
8. **Meta-prompting**: pone l'enfasi sulla struttura della risposta (ad esempio il formato delle user stories), piuttosto che sul contenuto specifico.
9. **Reflection**: induce il modello a rivedere e criticare la propria risposta iniziale, per poi proporre una versione revisionata e migliorata.
10. **Prompt Ensemble**: consiste nella combinazione delle tre tecniche più efficaci identificate sperimentalmente, con l'obiettivo di unire i rispettivi punti di forza e ottenere user stories più accurate e consistenti.

La selezione delle tecniche di prompt engineering è stata guidata da considerazioni metodologiche. In particolare:

- Sono state incluse le tecniche **di base** (Zero-shot e Few-shot), poiché costituiscono i riferimenti più comuni in letteratura e permettono di stabilire un confronto minimo tra approcci con o senza esempi preliminari.

- Sono state considerate tecniche che **esplicitano il ragionamento del modello** (Chain of Thought, Self-consistency), in quanto la trasformazione di requisiti in user stories richiede coerenza logica e completezza, non solo somiglianza superficiale.
- Sono state selezionate tecniche che **arricchiscono il contesto** o strutturano il processo (Prompt Generate Knowledge, Prompt Chaining, Directional Stimulus), particolarmente adatte per ridurre ambiguità e guidare la generazione verso il formato Agile desiderato.
- Sono state incluse tecniche più recenti come **Meta-prompting e Reflection**, per valutare se l'enfasi sulla forma della risposta o l'autocorrezione del modello potessero migliorare la qualità delle user stories.
- Infine, è stato sperimentato un **approccio ensemble**, con l'obiettivo di verificare se la combinazione delle strategie più promettenti producesse un miglioramento rispetto all'uso singolo delle tecniche.

In questo modo, il set di tecniche copre un ampio spettro: dagli approcci più semplici e consolidati a quelli più sofisticati e recenti, consentendo un'analisi comparativa più completa. Ogni tecnica sarà approfondita nelle prossime sottosezioni.

#### 4.8.1 Zero-shot

La tecnica Zero-shot consiste nel richiedere al modello di svolgere un compito senza fornirgli esempi o spiegazioni preliminari. In questo approccio, il modello si affida esclusivamente alla propria conoscenza pre-addestrata per interpretare la richiesta e produrre una risposta coerente. È una delle tecniche più semplici di prompt engineering, ma rappresenta anche un punto di partenza utile per valutare le capacità *intrinseche* [21] di un LLM in un determinato compito.

Nel contesto della generazione automatica di user stories, l'approccio Zero-shot è stato utilizzato per chiedere direttamente al modello di trasformare i requisiti software in user stories secondo il formato standard "As a [actor], I want [functionality] so that [goal]". Questo permette di valutare fino a che punto il modello riesca, senza

alcun supporto esterno, a individuare attori, funzionalità e obiettivi a partire da un requisito espresso in linguaggio naturale.

Di seguito è riportato il prompt utilizzato con la tecnica Zero-shot.

Generate User Stories from the following requirements.  
Write every User Story in the following format:  
As a [actor], I want [functionality] so that [goal].

### 4.8.2 Few-shot

La tecnica Few-shot si basa sul fornire al modello alcuni esempi di input e output corretti prima di sottoporgli il compito da svolgere. In questo modo, il modello viene “guidato” e può inferire lo schema desiderato [22], riducendo ambiguità e aumentando la probabilità di produrre risposte coerenti con il formato atteso. Rispetto all’approccio Zero-shot, questa tecnica sfrutta la capacità dei LLM di apprendere rapidamente lo stile e la struttura richiesta a partire da un numero limitato di esempi.

Nel caso della trasformazione dei requisiti software in user stories, l’approccio Few-shot è stato implementato fornendo al modello **tre coppie di esempi requisito-user story**. Il numero di esempi è stato definito in modo sperimentale: un numero troppo ridotto (1-2) non permetteva al modello di stabilizzare la struttura. La scelta di fornire tre esempi è stata guidata da valutazioni empiriche: sono state effettuate prove preliminari con un numero variabile di esempi (3, 5, 10 e 20), osservando che le metriche di valutazione (BLEU, ROUGE, METEOR, BERTScore) risultavano pressoché identiche. Pertanto, l’adozione di tre esempi è stata ritenuta un compromesso adeguato, capace di garantire un supporto sufficiente al modello senza introdurre complessità superflua.

Gli esempi sono stati selezionati tra i requisiti del dominio medico, garantendo eterogeneità in termini di struttura linguistica e contenuto (creazione di record, associazione di identificatori, gestione di più valori). In questo modo, il modello ha potuto apprendere implicitamente la struttura standardizzata della user story “*As a [actor], I want [functionality] so that [goal]*”, osservandone diverse istanze applicate a casi concreti.

L'estratto seguente mostra i tre esempi utilizzati:

**Examples:**

**Requirement 1:** The system shall create a single patient record for each patient.

**User Story 1:** As a System User, I want the system to create a single patient record for each patient so that all data about a patient is stored in one place.

**Requirement 2:** The system shall associate (store and link) key identifier information (e.g. system ID, medical record number) with each patient record.

**User Story 2:** As a System User, I want the system to associate key identifier information (e.g., system ID, medical record number) with each patient record so that I can correctly identify each patient.

**Requirement 3:** The system shall provide the ability to store more than one identifier for each patient record.

**User Story 3:** As a System User, I want the system to allow multiple identifiers to be associated with each patient record so that patients with different identifiers can be found accurately.

Generate User Stories from the following requirements. Write every User Story in the following format: As a [actor], I want [functionality] so that [goal].

### 4.8.3 Chain of Thought

La tecnica *Chain-of-Thought* (CoT) richiede al modello di esplicitare passaggi intermedi del proprio ragionamento prima di produrre l'output finale. Per la trasformazione di requisiti software in user stories abbiamo scelto una scomposizione in quattro passaggi: (1) identificazione degli attori, (2) estrazione delle funzionalità per ciascun attore, (3) identificazione dell'obiettivo (*why*) per ogni coppia attore/funzionalità, (4) composizione finale nello schema *As a [actor], I want [functionality] so that [goal]*, per motivi sia teorici sia empirici.

Di seguito sono riportate le motivazioni che hanno portato a scegliere una scomposizione in quattro passaggi:

- **Mappatura concettuale:** una user story si costruisce attorno a tre elementi essenziali (*who*, *what*, *why*). Separare l'estrazione di ciascun elemento riduce il rischio che uno di essi venga omissso o confuso nella generazione diretta.
- **Problemi linguistici reali:** gli attori non sempre vengono nominati nello stesso modo (ad esempio *user*, *clinician*, *system administrator*) e a volte non sono nemmeno nominati esplicitamente: possono essere sottintesi o richiamati indirettamente con pronomi o riferimenti testuali. Le funzionalità possono essere multiple o implicite; il *why* è spesso sottinteso e richiede inferenza pragmatica. Uno step dedicato al *why* migliora la probabilità che la motivazione venga effettivamente catturata.
- **Sperimentazione empirica:** prove esplorative hanno mostrato che, a volte, prompt che chiedevano direttamente la user story tendevano a omettere il *why* o a generare formulazioni vaghe; richiedere esplicitamente *who/what/why* in sotto-passaggi ha aumentato la completezza delle storie generate. L'estratto mostra il prompt utilizzato.

For each one of the following software requirements:

1. Identify all the actors.
2. For each actor, identify all the functionalities you can find.
3. For each actor/functionalit  couple, identify the reason/objective/goal.
4. Write a list of each triple (actor/functionalit /goal) in the following format: As a [actor], I want [functionalit ] so that [goal].

### Single-step (Chain-of-Thought) vs Multi-step (Prompt Chaining)

Nel contesto del prompt engineering, un compito complesso come la generazione di user stories pu  essere affrontato tramite due approcci principali, che si differenziano per la gestione dei passaggi intermedi:



- **Single-step:** il modello riceve un unico prompt contenente l'intero compito e, se richiesto, esplicita i passaggi logici (Chain-of-Thought) prima di fornire la risposta finale.
  - *Pro:* semplice da implementare, singola chiamata API, mantiene tutto il contesto integro.
  - *Contro:* output spesso molto verboso (uso elevato di token), maggiore rischio che il modello salti o confonda elementi.
- **Multi-step (Prompt Chaining):** il compito viene suddiviso in più richieste successive, ad esempio: identificazione degli attori, individuazione delle funzionalità, determinazione degli obiettivi, composizione finale della user story.
  - *Pro:* maggiore controllo e trasparenza (ogni step verificabile), gestione di input lunghi (batching), riduzione delle omissioni degli elementi fondamentali.
  - *Contro:* richiede orchestrazione (più chiamate API), necessità di conservare e normalizzare lo stato tra i passaggi, implementazione leggermente più complessa.

Concludendo, la Chain-of-Thought rappresenta uno stile di ragionamento interno al modello, mentre il Prompt Chaining, che verrà trattato più avanti, è una strategia operativa per suddividere il compito in più step controllabili.

#### 4.8.4 Self-consistency

La tecnica *Self-consistency* prevede la generazione di più risposte per lo stesso input, per poi selezionare quella più coerente o più frequente tra le varianti prodotte. Questo approccio sfrutta il fatto che i modelli di linguaggio possono proporre più soluzioni plausibili e consente di ridurre errori dovuti a risposte isolate poco accurate [23]. In altre parole, invece di fidarsi di una singola generazione, il modello esplora diverse possibilità e viene scelto il risultato più stabile tra tutte.

Nel contesto della trasformazione dei requisiti software in user stories, il prompt è stato costruito combinando due tecniche già descritte:

- **Few-shot:** alcuni esempi di requisiti e le relative user stories, per guidare il modello sul formato desiderato.
- **Chain of Thought:** esplicitazione dei passaggi logici per individuare attori, funzionalità e obiettivi prima di produrre la user story completa.

La generazione viene quindi ripetuta più volte per lo stesso requisito, e tra le risposte ottenute viene selezionata quella che appare più coerente rispetto agli esempi forniti e alla struttura standardizzata.

[Stesso testo di Few-shot] + [Stesso testo di Chain of Thoughts]

*Nota:* per la Self-consistency, questo prompt viene inviato più volte al modello e tra le risposte generate viene scelta quella più frequente o coerente.

#### 4.8.5 Prompt Generate Knowledge

La tecnica *Prompt Generate Knowledge* ha l'obiettivo di fornire al modello informazioni aggiuntive e contesto preliminare, così da migliorare la precisione e l'aderenza delle risposte allo scenario d'uso. L'idea alla base è che un LLM dotato di conoscenza contestuale rilevante possa generare user stories più corrette e complete rispetto a un approccio *naïve*.

Nel contesto della generazione di user stories dai requisiti software, questa tecnica è stata applicata in due fasi:

1. Fornitura di conoscenza preliminare: il modello riceve brevi descrizioni sul formato standard delle user stories e sugli elementi fondamentali da includere (attore, funzionalità, obiettivo). Questo contesto serve a guidare la generazione, riducendo omissioni o fraintendimenti.
2. Generazione delle user stories: successivamente, il modello utilizza la conoscenza appena acquisita per trasformare ciascun requisito software in una user story completa, rispettando la struttura "As a [actor], I want [functionality] so that [goal]".

La formulazione del prompt utilizzata è la seguente:

Using prior knowledge of standard user story formats, transform each software requirement into the following format:  
'As a [actor], I want [functionality] so that [goal]'.

Questa impostazione garantisce che il modello sfrutti il contesto fornito per produrre risposte più coerenti e aderenti agli esempi tipici di user story, riducendo il rischio di omissioni o formulazioni scorrette.

### 4.8.6 Prompt Chaining

La tecnica **Prompt Chaining** consiste nel scomporre un compito complesso in più passaggi sequenziali, ciascuno rappresentato da un prompt distinto. L'output di un prompt diventa l'input per il successivo, fino a ottenere la risposta finale. In questo senso, il Prompt Chaining è una modalità *multi-step* della generazione guidata da LLM, che consente maggiore controllo sui risultati e una gestione più trasparente degli step intermedi [24].

Nel contesto della generazione di user stories, la pipeline è stata strutturata come segue:

1. Identificazione degli attori principali associati a ciascun requisito software;
2. Individuazione delle funzionalità desiderate per ogni attore;
3. Determinazione degli obiettivi che l'attore intende raggiungere con ciascuna funzionalità;
4. Composizione finale della user story combinando attori, funzionalità e obiettivi nel formato standardizzato "As a [actor], I want [functionality] so that [goal]".

Questa scomposizione multi-step ha permesso di ottenere output più coerenti, strutturati e facilmente verificabili, riducendo il rischio di omissioni o confusione tra gli elementi della user story.

**Prompt 1:** "Identify the main actor associated with each software requirement."

**Prompt 2:** "Define the desired functionality for the identified actor."

**Prompt 3:** "Describe the goal the actor wants to achieve with this functionality."

**Final Prompt:** "Combine the results from each step to create a user story in the format: 'As a [actor], I want [functionality] so that [goal]'."

#### 4.8.7 Prompt Directional Stimulus

La tecnica **Prompt Directional Stimulus** consiste nel fornire al modello delle “direzioni” o esempi parziali che guidano la forma della risposta, senza però costruirla integralmente. Questa strategia permette di bilanciare la libertà creativa del modello con vincoli leggeri, aumentando la controllabilità delle risposte generate [25].

Nel contesto di questa tesi, sono stati forniti al modello esempi di attori tipici (*System User*, *System Administrator*, *Healthcare Provider*) come stimolo. La scelta di tre attori principali è stata motivata da considerazioni empiriche:

- Limitare il numero di hint evita di appesantire il prompt e di occupare inutilmente spazio nella context window;
- Tre attori rappresentano le categorie più ricorrenti nei requisiti dei progetti EHR selezionati, fornendo al modello un contesto sufficiente per identificare correttamente i ruoli senza introdurre ambiguità;
- Test preliminari hanno evidenziato che aggiungere ulteriori attori non migliorava significativamente le metriche (BLEU, ROUGE, METEOR, BERTScore), mentre ridurre gli hint a uno o due comportava omissioni o confusione nell'assegnazione dei ruoli.

In questo modo, gli hint hanno consentito al modello di generare user stories più coerenti con i ruoli effettivi presenti nei requisiti software, senza vincolare eccessivamente la creatività del modello.

[stesso testo di Zero-shot]

Hint: System User, System Administrator, Healthcare Provider.

### 4.8.8 Meta-prompting

La tecnica **Meta-prompting** consiste nell'includere nel prompt istruzioni esplicite su come il modello dovrebbe generare la risposta, specificando formato, livello di dettaglio e priorità degli elementi da considerare. Questo approccio aiuta il modello ad autoregolarsi e ad aderire meglio alle aspettative dell'utente, riducendo omissioni o errori di formattazione [26].

Nel contesto della generazione automatica di user stories, il meta-prompting è stato implementato fornendo al modello istruzioni dettagliate per ciascuno degli elementi fondamentali di una user story: attore, funzionalità e obiettivo. La scelta di includere istruzioni così dettagliate è stata motivata dalle seguenti considerazioni:

- **Role Identification:** Si richiede al modello di definire l'attore principale per ogni requisito. Questo è fondamentale perché gli attori non sono sempre indicati con lo stesso termine (es. utente, medico, amministratore di sistema) e possono essere menzionati tramite pronomi o sinonimi che fanno riferimento allo stesso soggetto (coreferenza). Fornire linee guida riduce la probabilità che il modello ometta o confonda il ruolo.
- **Function Categorization:** Si indica di astrarre l'azione principale richiesta dal requisito. La specificazione dei verbi più ricorrenti (*create, associate, allow*) consente di uniformare le user stories e ridurre la variabilità lessicale.
- **Goal Structuring:** Si chiede al modello di esplicitare l'obiettivo della funzionalità, così da garantire che ogni user story contenga un "why" chiaro. Questo

passaggio è stato motivato da test preliminari che hanno evidenziato omissioni occasionali del goal quando non era esplicitamente richiesto.

- **Template:** Viene specificato il formato da rispettare: 'As a [actor], I want [functionality] so that [goal]'. La motivazione è doppia: assicura consistenza tra le storie e facilita il confronto automatico con l'oracolo.
- **Example of structured output:** Fornire un esempio astratto di user story consente al modello di comprendere la sintassi e la struttura generale senza vincolare il contenuto specifico.

Questa granularità nel prompt è stata scelta per ridurre errori comuni osservati durante test preliminari, come omissioni dell'attore o del goal, confusione tra le funzionalità, o deviazioni dal formato standard. L'obiettivo era quindi garantire una generazione di user stories più coerente, completa e replicabile.

To generate user stories from software requirements, apply a structured template emphasizing the roles, goals, and functions involved in each requirement. For each requirement, identify and categorize the following components to create a coherent user story:

**Role Identification:** Define the primary *actor* (e.g., System User, System Administrator, Healthcare Provider) relevant to the requirement.

**Function Categorization:** Abstract the primary action or functionality requested, such as *create*, *associate*, or *allow*.

**Goal Structuring:** Frame the objective or purpose behind the functionality (e.g., *to create a single patient record for each patient, to merge patient information from two records into one*).

**Template:** For each user story, use the following structure without specific content details: 'As a [actor], I want [functionality] so that [goal].'

**Example of structured output:** 'As a [role/category], I want [general functionality] so that [general goal].'

Apply this template abstractly, ensuring that the syntax of each user story adheres to the above structural pattern, even if specific role or goal details vary.

### 4.8.9 Reflection

La tecnica Reflection si basa sull'idea di far ragionare il modello sui propri output, invitandolo a valutare eventuali errori, omissioni o formulazioni ambigue, e a generare una versione migliorata del testo. Questo approccio simula un processo iterativo di autocorrezione, analogo al comportamento umano durante un compito complesso [27].

Nel contesto della generazione di user stories, Reflection è stata utilizzata per ridurre casi di incompletezza (mancanza di *who*, *what* o *why*) e per aumentare la coerenza delle storie prodotte. La tecnica è stata applicata passando al modello sia il prompt originale (Few-shot) sia la lista dei requisiti e delle user stories generate in precedenza, invitandolo a criticare e riformulare la risposta.

**Vantaggi:** migliora la qualità finale delle user stories, riduce omissioni e incoerenze, supporta la generazione iterativa senza intervento umano.

**Limiti:** comporta un aumento della complessità del prompting e dei tempi di generazione, soprattutto su dataset molto grandi.

La scelta di includere la tecnica Reflection è stata guidata da due motivazioni principali. Primo, si è voluto massimizzare la correttezza e la completezza delle user stories generate, riducendo omissioni di elementi fondamentali (*attore*, *funzionalità*, *obiettivo*) e migliorando la coerenza complessiva. Secondo, Reflection consente di simulare un processo iterativo di revisione, fornendo un meccanismo automatico di autocorrezione che, pur aumentando la complessità computazionale, riduce la necessità di intervento umano per la verifica e la correzione manuale dei risultati. Questo approccio è particolarmente utile quando si lavora su dataset di grandi dimensioni, come nel caso della lista di 1200 requisiti utilizzata in questa tesi, dove la valutazione manuale di tutte le user stories sarebbe impraticabile.

Copia del primo prompt: stesso testo di Few-shot + lista dei requisiti + lista di user stories generate nella risposta:

[Esempi:

Requisito 1: xxx

User story 1: yyy

...

Lista requisiti: x y z

Risposta del LLM]

---

This was the answer you gave, and now, using reflection and criticizing your response, elaborate on your last answer so you can give a better answer.

#### 4.8.10 Prompt Ensemble

L'approccio di Prompt Ensemble prevede di utilizzare più prompt diversi per lo stesso requisito, generando quindi più possibili user stories. Le tecniche da combinare che sono state scelte, sono le 3 che hanno ottenuto i risultati migliori in base alle metriche di validazione. Questo approccio consente di ridurre la dipendenza da un singolo prompt, mitigando i bias associati a una sola formulazione, e di ottenere user stories più robuste e coerenti grazie alla diversità delle prospettive generate.

La scelta di implementare Prompt Ensemble è stata motivata dalla volontà di sfruttare i punti di forza complementari delle diverse tecniche di prompt engineering. Combinando Meta-prompting, Prompt Directional Stimulus e Few-shot, si ottiene una maggiore copertura dei possibili scenari e una riduzione del rischio che una singola formulazione generi output incompleti o incoerenti. Questo approccio è particolarmente utile nel contesto della generazione automatica di user stories da requisiti software, in quanto consente di integrare prospettive diverse, migliorando la robustezza e l'affidabilità delle storie prodotte.



Combinazione di Meta-prompting, Prompt Directional Stimulus, Few-shot

## 4.9 Metriche di Validazione

Per valutare la qualità delle user stories generate automaticamente dai modelli linguistici, sono state adottate diverse metriche comunemente utilizzate nell'ambito del Natural Language Generation (NLG) [28]. L'obiettivo è misurare quanto il testo prodotto sia vicino a un risultato atteso, valutando aspetti come la sovrapposizione lessicale, la similarità semantica e la capacità di mantenere il significato.

BLEU è una metrica basata sul confronto tra n-gram (sequenze di parole) del testo generato e quelli del testo di riferimento. È stata utilizzata in quanto fornisce una misura quantitativa della somiglianza lessicale ed è utile per valutare la correttezza superficiale delle frasi generate rispetto alle reference.

ROUGE è una famiglia di metriche che calcola l'overlap tra n-gram, parole e sottosequenze tra testo generato e riferimento. In particolare, ROUGE-L tiene conto delle sequenze più lunghe comuni. È stata utilizzata perché è complementare a BLEU, in quanto si focalizza sul recall e quindi sulla copertura informativa del testo generato rispetto al riferimento.

METEOR è una metrica che considera corrispondenze non solo esatte tra parole, ma anche basate su sinonimi, stemming e variazioni morfologiche. È stata scelta perché migliora la robustezza rispetto a BLEU e ROUGE, permettendo di riconoscere come corrette anche riformulazioni valide che non condividono le stesse parole.

BERTScore utilizza modelli di language representation come BERT per calcolare la similarità semantica tra il testo generato e quello di riferimento a livello di embedding. È stata impiegata perché consente di valutare la vicinanza semantica tra frasi anche quando la superficie lessicale è molto diversa, risultando particolarmente adatta a testi brevi e variabili come le user stories [29].

## CAPITOLO 5

---

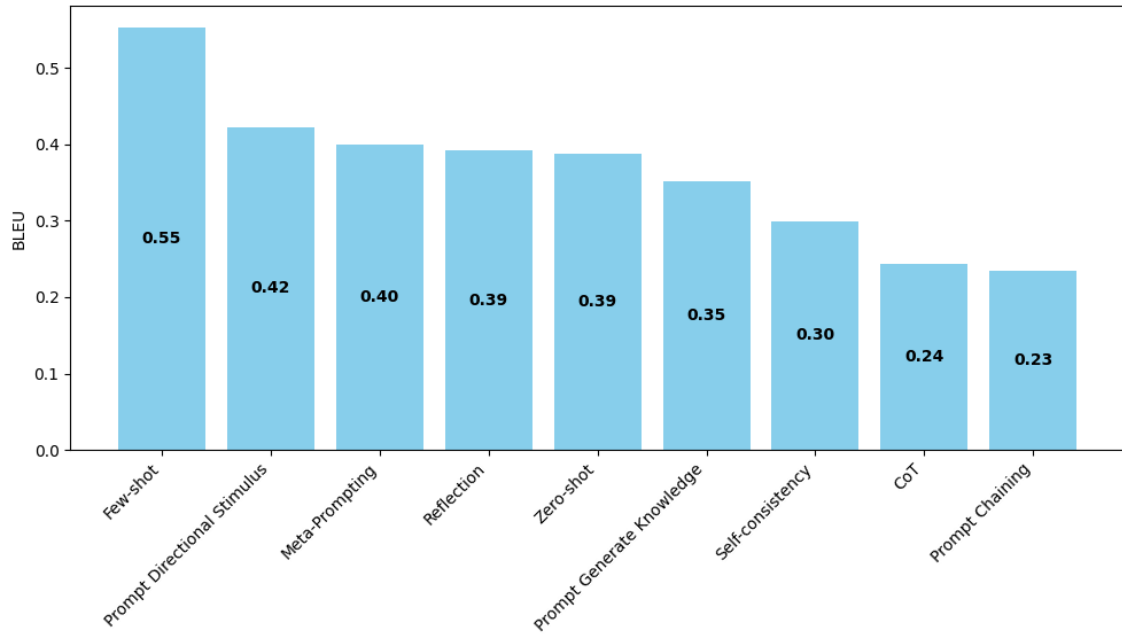
### Risultati

---

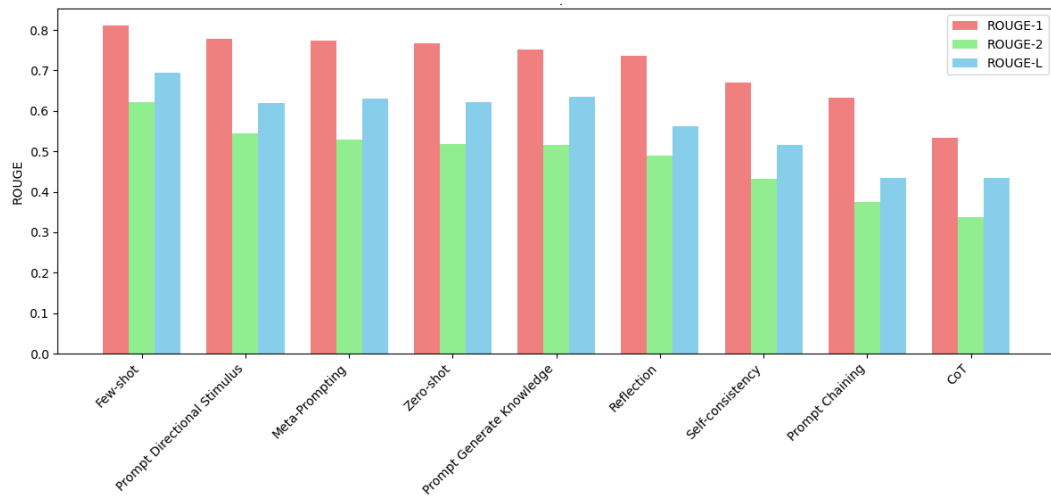
Di seguito sono presentate le analisi e le risposte alle domande di ricerca, sviluppate attraverso una combinazione di valutazioni quantitative, confronto tra diverse tecniche di prompt engineering e modelli LLM, ed esempi qualitativi mirati a evidenziare differenze e criticità nella generazione automatica di user stories.

## 5.1 RQ1. Confronto tra tecniche di Prompt Engineering

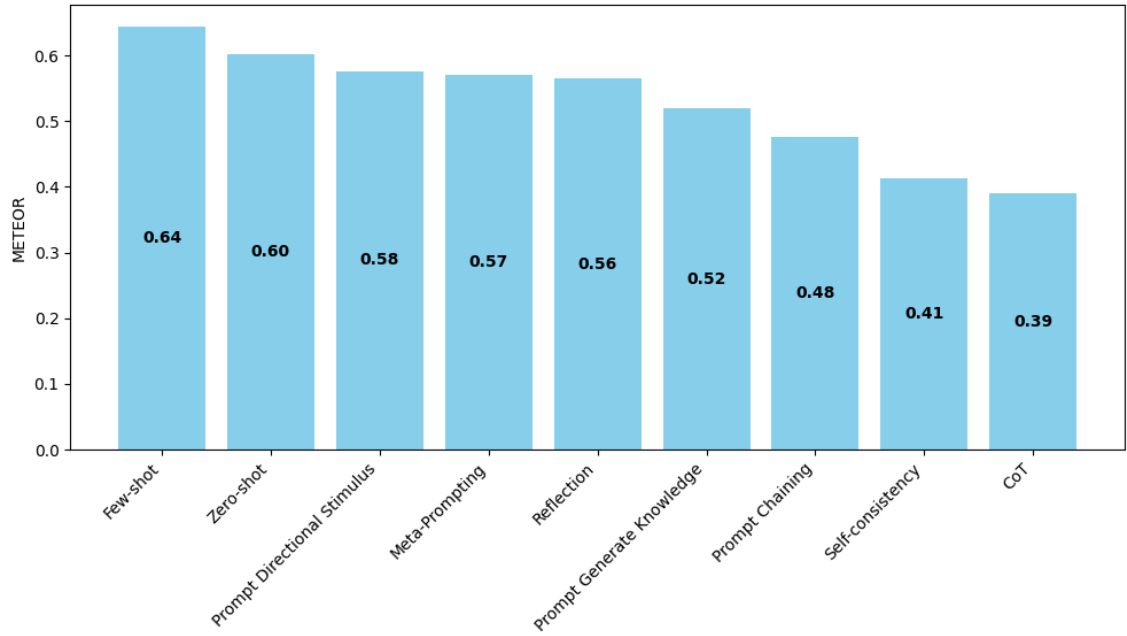
### 5.1.1 Analisi quantitativa



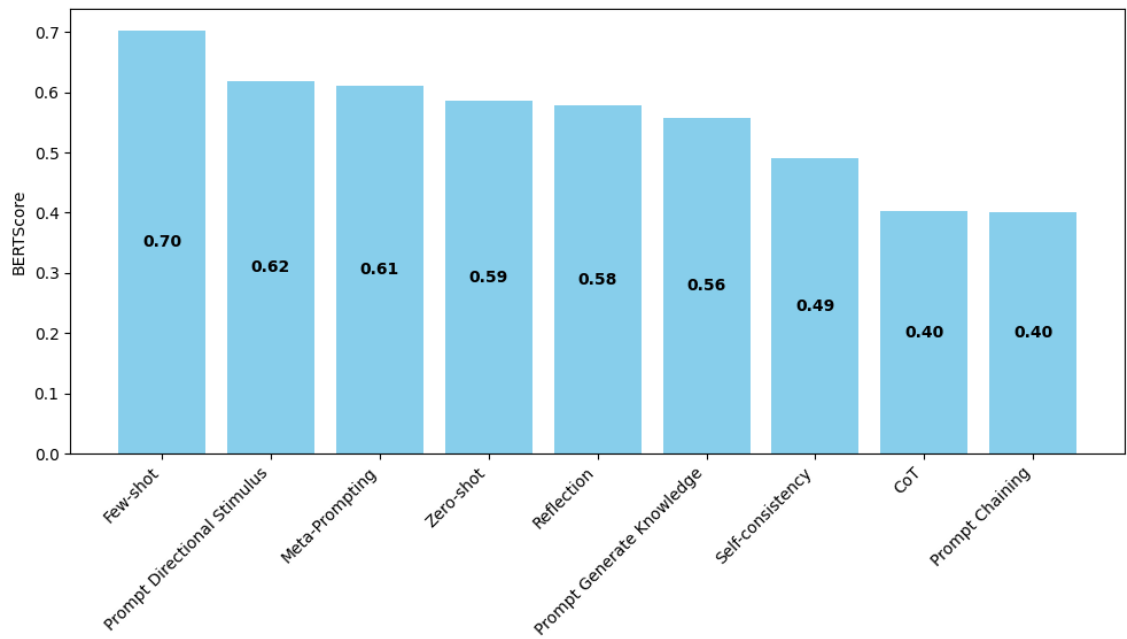
**Figura 5.1:** BLEU Score medio per tecnica.



**Figura 5.2:** ROUGE Scores medi per tecnica.



**Figura 5.3:** METEOR Score medio per tecnica.



**Figura 5.4:** BERTScore medio per tecnica.

Tecnica	BLEU	R1	R2	RL	MET	BERT
Few-shot	0,55	0,81	0,62	0,70	0,64	0,70
Zero-shot	0,39	0,77	0,52	0,62	0,60	0,59
CoT	0,24	0,53	0,34	0,43	0,39	0,40
Self-consistency	0,30	0,67	0,43	0,51	0,41	0,49
Prompt Gen Know.	0,35	0,75	0,52	0,63	0,52	0,56
Prompt Chaining	0,23	0,63	0,37	0,43	0,48	0,40
Directional Stimulus	0,42	0,78	0,54	0,62	0,58	0,62
Meta-Prompting	0,40	0,77	0,53	0,63	0,57	0,61
Reflection	0,39	0,74	0,49	0,56	0,56	0,58

**Tabella 5.1:** Valori medi delle metriche per ciascuna tecnica di prompt engineering.

Dai risultati dei valori medi, **Few-shot** risulta costantemente la tecnica più efficace, con punteggi elevati e stabili in tutte le metriche principali. Ciò conferma quanto emerso in letteratura [30]: fornire esempi contestuali permette al modello di cogliere meglio lo schema desiderato e ridurre ambiguità. **Prompt Directional Stimulus** e **Meta-Prompting** ottengono buone performance, seppur leggermente inferiori. Queste tecniche introducono un controllo leggero, ad esempio hint sugli attori o istruzioni sul formato, che aumentano la consistenza delle risposte ma non garantiscono la stessa adattabilità e precisione ottenuta con esempi concreti. Al contrario, **Chain-of-Thought** e **Prompt Chaining** mostrano valori più bassi e instabili: nel contesto delle user stories, la richiesta di esplicitare passaggi intermedi o concatenare più prompt porta a un rumore maggiore, generando omissioni di attori o obiettivi. Questo risultato suggerisce che la complessità logica non sempre si traduce in maggiore qualità, soprattutto per task che richiedono una struttura fissa e ripetitiva. Bisogna anche considerare l'influenza del dominio applicativo sui risultati: in domini con terminologia più standardizzata i punteggi potrebbero essere più alti, mentre in scenari meno strutturati il modello potrebbe faticare a mantenere consistenza.

Tecnica	BLEU	ROUGE1	ROUGE2	ROUGEL	METEOR	BERTScore
Pr. Ensemble	0,56	0,82	0,62	0,70	0,66	0,71
Few-shot	0,55	0,81	0,62	0,69	0,64	0,70

**Tabella 5.2:** Confronto tra Prompt Ensemble e Few-shot sulle metriche di valutazione.

La differenza tra Few-shot e Prompt Ensemble è marginale, ma significativa da un punto di vista metodologico: indica che combinare strategie complementari può apportare benefici senza compromettere la stabilità dei risultati. Il **Prompt Ensemble**, combinando Few-shot, Meta-Prompting e Prompt Directional Stimulus, raggiunge in media i punteggi migliori, ma il guadagno rispetto a Few-shot puro è marginale (0,01–0,02). Ciò suggerisce che l'integrazione di strategie complementari produce un miglioramento lieve ma coerente, confermando la robustezza dell'approccio ensemble.

### 5.1.2 Confronto tra metodi

In questa sezione si confrontano le tecniche di prompt engineering considerando non solo i valori quantitativi, ma anche le caratteristiche dei prompt e lo scenario applicativo ). L'obiettivo è comprendere perché alcune strategie risultino più efficaci di altre.

#### Few-shot

Few-shot si conferma la tecnica più efficace in quasi tutte le metriche. Questo risultato è coerente con le aspettative: fornire esempi concreti consente al modello di cogliere lo schema desiderato e ridurre ambiguità nella generazione delle user stories. La semplicità della tecnica nasconde però un vantaggio notevole: rispetto a metodi più complessi, Few-shot richiede meno passaggi logici intermedi e meno orchestrazione, riducendo il rischio di omissioni di attori, funzionalità o obiettivi. Un limite è che la qualità degli esempi forniti influenza direttamente l'output: esempi poco chiari o incoerenti si riflettono negativamente sulle storie generate.

### **Prompt Ensemble**

Il Prompt Ensemble combina Few-shot, Meta-Prompting e Prompt Directional Stimulus. In teoria, la fusione di più approcci dovrebbe garantire un vantaggio significativo, ma i risultati mostrano un miglioramento marginale rispetto a Few-shot puro (0,01–0,02). L'effetto positivo dell'ensemble risulta più evidente in scenari con requisiti complessi o ambigui, dove le indicazioni extra possono aiutare a risolvere casi borderline. D'altro canto, l'orchestrazione di più tecniche aumenta la complessità dell'approccio e riduce l'immediatezza operativa, rendendolo meno pratico in scenari reali.

### **Zero-shot**

Zero-shot ottiene punteggi medi inferiori rispetto a Few-shot, come previsto. L'assenza di esempi concreti limita la capacità del modello di identificare correttamente lo schema desiderato, aumentando il rischio di omissioni o formulazioni incomplete. Questa tecnica mostra comunque un comportamento relativamente stabile, suggerendo che il modello dispone di conoscenza pregressa sufficiente per compiti generali, ma non ottimale per la strutturazione precisa di user stories. Il vantaggio principale è la rapidità: non richiede costruzione manuale di esempi, risultando quindi utile in scenari esplorativi o a basso costo.

### **Prompt Generate Knowledge**

Prompt Generate Knowledge migliora la performance rispetto a Zero-shot fornendo al modello informazioni aggiuntive sul formato e sugli elementi da includere. Questa tecnica dimostra che un contesto aggiuntivo può incrementare la coerenza e ridurre omissioni. Nel contesto delle user stories, il vantaggio è evidente quando i requisiti sono complessi o formulati in modo ambiguo, perché il modello sfrutta le istruzioni extra come vincoli di guida. Il limite è che, se le istruzioni sono troppo generiche o astratte, il modello rischia comunque di deviare dallo schema corretto.

### **Meta-Prompting e Prompt Directional Stimulus**

Queste tecniche di controllo leggero ottengono risultati buoni ma inferiori a Few-shot. Le istruzioni sul formato (Meta-Prompting) e gli hint sugli attori (Prompt Directional Stimulus) migliorano la consistenza e riducono omissioni, ma la mancanza di esempi concreti limita l'adattabilità del modello a requisiti variabili. In particolare, Prompt Directional Stimulus si rivela utile quando i requisiti contengono riferimenti ambigui agli attori, mentre Meta-Prompting migliora la regolarità stilistica. Tuttavia, entrambi non risolvono completamente i casi complessi o con ambiguità semantiche.

### **Chain-of-Thought e Prompt Chaining**

Queste tecniche sequenziali e complesse hanno performance più basse e instabili. La gestione di più passaggi intermedi aumenta la probabilità di omissioni o incoerenze tra attore, funzionalità e obiettivo. Nel contesto delle user stories, dove ogni elemento deve essere presente e coerente, la complessità del ragionamento sequenziale può diventare un limite anziché un vantaggio. Questi risultati suggeriscono che tecniche di reasoning sono più adatte a compiti che richiedono inferenze logiche profonde, mentre sono controproducenti quando il compito richiede struttura rigida e ripetitiva.

### **Self-consistency e Reflection**

Self-consistency e Reflection ottengono valori intermedi. Generare più risposte o rivedere criticamente l'output migliora la robustezza, ma il sovraccarico computazionale e la gestione di molteplici output rendono queste tecniche meno pratiche con dataset grandi come quello dei requisiti software. Il vantaggio principale è l'aumento di affidabilità in scenari critici, ma a fronte di un costo operativo che ne limita l'applicabilità su larga scala.



### 5.1.3 Esempi qualitativi

Per comprendere meglio le differenze tra le tecniche di prompt engineering, è utile considerare anche esempi concreti di user stories generate. In questa sezione selezioniamo tre tipologie di esempi per ciascuna tecnica:

- **Esempio corretto:** la user story rispetta il formato atteso, con attore, funzionalità e obiettivo coerenti.
- **Esempio borderline:** la user story è generalmente comprensibile, ma presenta qualche incongruenza, ad esempio nella scelta dell'attore o nella formulazione del goal.
- **Esempio errato:** la user story contiene errori significativi, come un attore irrealistico, un obiettivo non coerente o una funzionalità fraintesa.

Gli esempi sono stati selezionati da tre categorie di tecniche:

1. **Zero-shot**, come punto di partenza, per osservare come il modello si comporta senza esempi.
2. **Few-shot**, una tecnica semplice ma molto efficace, che fornisce esempi chiari e concreti, e consente di riflettere sul motivo per cui funziona meglio di approcci più complessi.
3. **Chain-of-Thought**, una tecnica complessa e sequenziale che ha ottenuto risultati mediamente bassi, utile per capire quali passaggi logici possono introdurre errori o incoerenze.

Analizzando questi esempi possiamo comprendere non solo quali errori si verificano, ma anche le cause, come la gestione del contesto, la scelta dell'attore o la complessità del prompt.

#### Esempi qualitativi: Zero-shot

##### Esempio corretto:

Requisito originale: *The system shall provide the ability to merge patient information from*

*two patient records into a single patient record.*

User story generata: *As a healthcare provider, I want to merge patient information from two patient records into a single patient record so that duplicate entries can be avoided.*

**Motivazione:** L'output rispetta il formato "As a [actor], I want [functionality] so that [goal]", l'attore è adeguato, la funzionalità cattura correttamente il requisito e l'obiettivo è chiaro e pertinente.

**Esempio borderline:**

Requisito originale: *The system shall provide the ability to capture and maintain the principal provider responsible for the care of an individual patient.*

User story generata: *As an EHR system, I want to capture and maintain the principal provider responsible for the care of an individual patient so that accountability for patient care can be ensured.*

**Motivazione:** La funzionalità e l'obiettivo sono corretti, ma l'attore "EHR system" è poco realistico: avrebbe dovuto essere un utente umano come un amministratore o un provider. Ciò riflette un limite tipico dello zero-shot: senza esempi, il modello confonde chi utilizza il sistema con il sistema stesso.

**Esempio errato:**

Requisito: *The system shall provide the ability to maintain and make available historic information for demographic data including prior names, addresses, phone numbers, and email addresses.*

User story Zero-shot: *As a historian, I want to maintain and make available historic information for demographic data including prior names, addresses, phone numbers, and email addresses so that historical context can be preserved.*

**Motivazione:** L'attore "historian" non è realistico in un contesto EHR; pur essendo funzionali il task e il goal, l'interpretazione dell'attore compromette la correttezza complessiva della user story. Lo zero-shot, privo di esempi, si ancora troppo a parole superficiali come "historic", perdendo di vista il contesto clinico.

**Esempi qualitativi: Few-shot**

- **Esempio corretto:**

Requisito: *The system shall provide the ability to merge patient infor-*

mation from two patient records into a single patient record.

User story Few-shot: As a System User, I want the system to merge patient information from two patient records into a single patient record so that patient data is updated efficiently.

L'output è coerente con il requisito originale, includendo attore, azione e obiettivo chiaramente definiti.

- **Esempio borderline:**

Requisito: The system shall provide a field which will identify patients as being exempt from reporting functions.

User story Few-shot: As a Healthcare Administrator, I want to identify patients as being exempt from reporting functions via a specific field so that I can manage and report patient data correctly.

L'attore è leggermente diverso rispetto all'oracolo (Healthcare Administrator vs System Administrator), ma la user story rimane comprensibile e coerente con il requisito. Questo mostra un pregio e un limite della tecnica: flessibilità interpretativa utile, ma rischio di discrepanze sottili nei progetti reali.

- **Esempio errato:**

Requisito: The system shall provide the ability to merge patient information from two patient records into a single patient record.

User story Few-shot: As a System User, I want the system to merge patient information from two patient records into a single patient record so that patient data is updated efficiently.

Sebbene la struttura sia corretta, l'obiettivo ("patient data is updated efficiently") è meno preciso rispetto all'oracolo ("there is no duplication of patient data"). Ciò evidenzia una comprensione incompleta del requisito: la tecnica garantisce struttura corretta, ma può ridurre la fedeltà semantica ai dettagli.

### **Esempi qualitativi: Chain-of-Thought**

- **Esempio corretto:**

Requisito: The system shall provide the ability to merge patient information from two patient records into a single patient record.

User story CoT: As an Healthcare Provider, I want to be able to merge patient information from multiple sources so that accurate and complete medical history is maintained.

L'output è coerente con il requisito originale: attore, azione e obiettivo sono presenti, anche se il linguaggio è leggermente più generico.

- **Esempio borderline:**

Requisito: The system shall provide the ability to maintain the onset date of the problem or diagnosis.

User story CoT: As an Healthcare Provider, I want to be able to capture the start date of a problem or diagnosis so that accurate records are maintained.

L'azione è chiara, ma l'obiettivo è troppo generico ("accurate records are maintained") e non riflette completamente l'intento di monitoraggio clinico del requisito originale. Questo mostra come il ragionamento passo-passo del CoT non sempre aggiunga precisione, ma talvolta porti a un appiattimento semantico.

- **Esempio errato:**

Requisito: The system shall provide the ability to store demographic information in discrete data fields.

User story CoT: As an Administrator, I want to be able to print reports so that healthcare professionals have relevant and accurate data.

L'output non corrisponde al requisito: l'attore e l'azione non riguardano la memorizzazione dei dati, ma la stampa di report. È un esempio di *drift logico*: il CoT, cercando di costruire catene di pensiero, devia dal requisito iniziale. Qui emerge un limite critico: maggiore complessità non garantisce maggiore affidabilità.

### 5.1.4 Interpretazione dei risultati

L'analisi combinata dei dati quantitativi e degli esempi qualitativi permette di comprendere in modo più profondo le differenze tra le tecniche di prompt engineering. Few-shot emerge come la tecnica più efficace, confermando che esempi concreti guidano il modello nella generazione di user stories complete e coerenti. Gli esempi qualitativi mostrano che, con Few-shot, l'attore, la funzionalità e l'obiettivo sono quasi sempre corretti, mentre eventuali discrepanze (es. piccole differenze nell'attore rispetto all'oracolo) non compromettono la comprensibilità complessiva. Questo evidenzia come la chiarezza e la pertinenza degli esempi compensino la semplicità del prompt, riducendo ambiguità e omissioni.

Zero-shot, invece, pur essendo stabile in termini di output generali, soffre di problemi specifici evidenziati dagli esempi qualitativi: l'attore scelto può risultare poco realistico o non coerente con il contesto (ad esempio "historian" o "EHR system"), mentre la funzionalità e l'obiettivo rimangono generalmente comprensibili ma non sempre precisi. Ciò conferma che la conoscenza pregressa del modello non sostituisce l'indicazione esplicita degli esempi, soprattutto in scenari complessi come la generazione di user stories da requisiti software.

Le tecniche complesse e sequenziali, come Chain-of-Thought, mostrano valori quantitativi bassi e alta variabilità. Gli esempi qualitativi chiariscono il perché: il modello può generare output coerenti in casi semplici, ma spesso l'obiettivo risulta troppo generico o l'attore mal selezionato, e in casi peggiori la funzionalità viene fraintesa. Questo suggerisce che la gestione di passaggi intermedi e il ragionamento sequenziale aumentano la probabilità di errori e di incoerenze, limitando l'efficacia della tecnica nonostante la sua complessità teorica.

In sintesi, il confronto tra le tre categorie di tecniche mostra pattern chiari:

- **Few-shot:** alta precisione e coerenza grazie agli esempi concreti, con limiti minimi nella scelta dell'attore o nella formulazione dell'obiettivo.
- **Zero-shot:** output generalmente comprensibili ma con attori poco realistici e obiettivi meno precisi, confermando il ruolo critico degli esempi.

- **Chain-of-Thought:** output variabili e instabili; la complessità sequenziale aumenta il rischio di omissioni o incoerenze, soprattutto nell'identificazione degli attori o nella formulazione dei goal.

Queste osservazioni indicano che, per la generazione automatica di user stories da requisiti software, la qualità e la pertinenza degli esempi forniti al modello hanno un impatto maggiore sulla precisione finale rispetto alla complessità del ragionamento richiesto. L'analisi qualitativa fornisce quindi insight essenziali per comprendere gli errori tipici di ciascuna tecnica e guidare la progettazione di prompt più efficaci. Queste riflessioni forniscono le basi per rispondere in modo esplicito alla RQ1.

### 5.1.5 Risposta alla RQ1

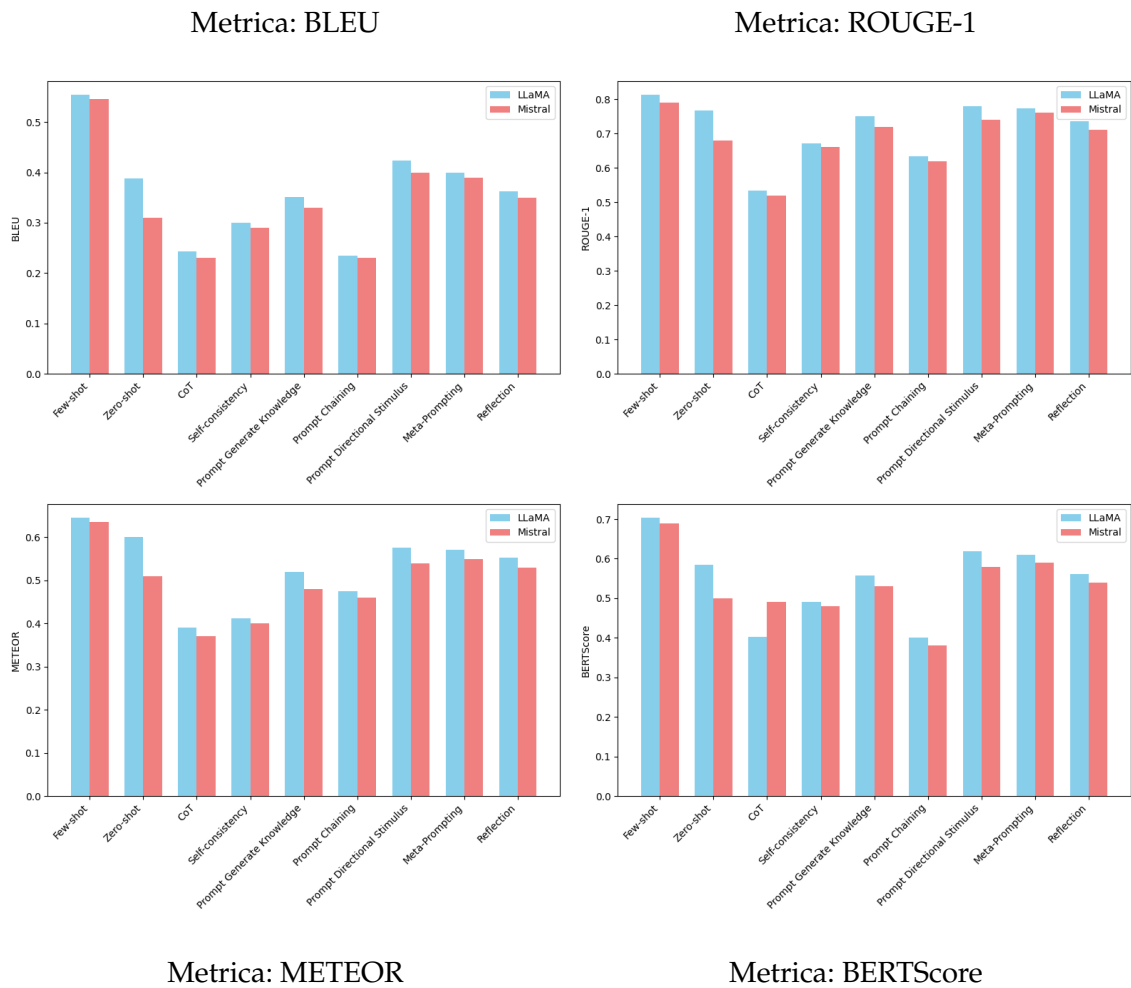
Dall'analisi quantitativa e qualitativa emerge che la tecnica di **Few-shot** si rivela la più efficace nella generazione automatica di user stories da requisiti software.

- I valori medi delle metriche (BLEU, ROUGE, METEOR e BERTScore) sono costantemente più alti rispetto a tutte le altre tecniche, mostrando stabilità e affidabilità nella generazione.
- Gli esempi qualitativi confermano che Few-shot produce user stories coerenti, complete e grammaticalmente corrette, con attori, funzionalità e obiettivi generalmente corretti. Anche eventuali discrepanze minori non compromettono la comprensibilità e la pertinenza del testo.
- Tecniche come **Zero-shot** o **Chain-of-Thought**, pur utili per comprendere il comportamento del modello senza esempi o in scenari sequenziali, presentano limiti significativi: attori non realistici, obiettivi generici e incoerenze nella funzionalità.

Ricapitolando, fornire esempi concreti contestuali al modello (Few-shot) permette di cogliere meglio lo schema desiderato delle user stories, riducendo ambiguità e omissioni, e rappresenta quindi la strategia più performante rispetto a tecniche più complesse o prive di esempi.

## 5.2 RQ2. Confronto tra LLM diversi

La seconda research question (RQ2) si concentra sull'identificazione del modello LLM più performante nella generazione automatica di user story. Per rispondere a questa domanda sono stati confrontati due modelli, LLaMA 3.1 8B e Mistral 7B.



**Figura 5.5:** Confronto tra LLaMA (blu) e Mistral (rosso) per le metriche.

Le analisi mostrano che, utilizzando la tecnica Few-shot, la differenza tra i due modelli è minima, suggerendo che l'inclusione di esempi guida entrambi i modelli verso prestazioni elevate.

Al contrario, nel caso Zero-shot, Mistral appare maggiormente penalizzato, mentre LLaMA mantiene performance più solide, evidenziando una maggiore capacità di generalizzazione senza supporto di esempi.

Per le tecniche più complesse, come Prompt Chaining o Chain-of-Thought, LLaMA mostra un vantaggio consistente, con uno scarto medio di circa il 3% rispetto a Mistral. Questo suggerisce che LLaMA gestisce meglio compiti che richiedono ragionamento sequenziale o una maggiore organizzazione del prompt, probabilmente grazie alla sua architettura e al training più ottimizzato per questi scenari.

Complessivamente, le differenze tra i due modelli risultano contenute, confermando che entrambi possono essere considerati strumenti validi per la generazione automatica di user stories, con LLaMA leggermente più performante nei casi più complessi e meno dipendenti dall'uso di esempi.

### 5.2.1 Esempi qualitativi

#### Confronto qualitativo: Zero-shot

Per la tecnica **Zero-shot**, confrontiamo le user stories generate da LLaMA e Mistral, selezionando esempi *corretti*, *borderline* ed *errati*.

#### Esempio corretto

- **Requisito:** The system shall be scalable and adaptable to meet future growth and expansion needs.
- **LLaMA:** As an IT Manager, I want the system to be scalable and adaptable to meet future growth and expansion needs so that we can easily add new features and accommodate increasing user demands.
- **Mistral:** As a Business Decision Maker, I want the system to be scalable and adaptable to meet future growth and expansion needs so that it can evolve with our organization's demands.

Entrambi i modelli catturano correttamente la funzionalità e includono un obiettivo. La differenza principale è nell'attore: LLaMA propone un ruolo tecnico (IT Manager), Mistral un ruolo decisionale (Business Decision Maker). In entrambi i casi, la user story è coerente e completa.



**Esempio borderline**

- **Requisito:** The screens shall be highly re-configurable, providing ability to reposition and rename field labels, remove or return-off unused fields, maintain data and allow addition of custom-defined fields.
- **LLaMA:** As a User, I want the screens to be highly re-configurable, allowing me to reposition and rename field labels, remove or return-off unused fields, maintain data, and add custom-defined fields so that I can tailor my experience to meet my needs.
- **Mistral:** As a System Designer, I want the screens to be highly re-configurable, providing the ability to reposition and rename field labels, remove or return-off unused fields, maintain data, and allow addition of custom-defined fields so that they are tailored to our specific needs.

Entrambi i modelli catturano la funzionalità completa, ma l'attore non riflette sempre il contesto ideale (User generico vs System Designer). L'obiettivo è presente in entrambe le versioni, ma leggermente più generico in LLaMA.

**Esempio errato**

- **Requisito:** The system shall provide the ability to create and possibly modify edits and business rules which determine The acceptance or correctness of data.
- **LLaMA:** As a Data Quality Specialist, I want the system to provide the ability to create and possibly modify edits and business rules which determine the acceptance or correctness of data so that we can ensure the accuracy and integrity of our data.
- **Mistral:** As a Data Manager, I want the system to provide the ability to create, modify, and edit business rules which determine The acceptance or correctness of data so that it maintains quality and accuracy.

LLaMA interpreta correttamente la funzionalità e il goal. Mistral fornisce una versione più chiara e leggibile, ma potrebbe semplificare troppo la gestione del requisito

originale. In generale, entrambi i modelli hanno difficoltà con requisiti complessi e ambigui, mostrando come Zero-shot non sempre cattura tutte le sfumature.

### Confronto qualitativo: Few-shot

Per la tecnica **Few-shot**, confrontiamo le user stories generate da LLaMA e Mistral, selezionando esempi *corretti*, *borderline* ed *errati*.

#### Esempio corretto

- **Requisito:** The system shall provide the ability to merge patient information from two patient records into a single patient record.
- **LLaMA:** As a System User, I want the system to merge patient information from two patient records into a single patient record so that patient data is updated efficiently.
- **Mistral:** As a System User, I want the ability to merge patient information from two patient records into a single patient record.

Entrambi i modelli catturano correttamente la funzionalità richiesta. LLaMA aggiunge un obiettivo (“so that patient data is updated efficiently”), rendendo la user story più completa, mentre Mistral fornisce solo la funzionalità, senza l’obiettivo.

#### Esempio borderline

- **Requisito:** The system shall provide a field which will identify patients as being exempt from reporting functions.
- **LLaMA:** As a Healthcare Administrator, I want the system to identify patients as being exempt from reporting functions via a specific field so that I can manage and report patient data correctly.
- **Mistral:** As a System User, I want the system to provide a field which will identify patients as being exempt from reporting functions.

LLaMA mantiene la struttura completa ma l'attore differisce leggermente (Healthcare Administrator vs System Administrator). - Mistral genera un output corretto ma manca del tutto l'obiettivo ("so that ..."), rendendo la user story meno informativa.

### Esempio errato

- **Requisito:** The system shall provide the ability to maintain and make available historic information for demographic data including prior names, addresses, phone numbers, and email addresses.
- **LLaMA:** As an Historian, I want the system to maintain and make available historic information for demographic data including prior names, addresses, phone numbers, and email addresses so that past patient data is accessible.
- **Mistral:** As a System User, I want the system to provide the ability to maintain and make available historic information for demographic data including prior names, addresses, phone numbers, and email addresses.

LLaMA interpreta l'attore in modo non realistico ("Historian"), pur mantenendo la funzionalità e un obiettivo generico. Mistral sceglie un attore neutro e generico ("System User") ma omette completamente il goal, riducendo la completezza della user story.

## 5.2.2 Interpretazione dei risultati

L'analisi qualitativa evidenzia differenze interessanti tra i due modelli nel generare user stories. Utilizzando la tecnica **Few-shot**, entrambi i modelli raggiungono prestazioni elevate, ma emergono alcune differenze qualitative: Mistral tende più frequentemente a semplificare o omettere il *why* della user story, mentre LLaMA mantiene una maggiore coerenza e completezza nell'esprimere obiettivi e motivazioni. Mistral potrebbe privilegiare la generazione della parte "azione" della user story (what e who), trascurando il why perché considera il goal meno critico per la correttezza sintattica o funzionale della frase. Questo suggerisce che la sua architettura o il training tendono a ottimizzare la coerenza locale della frase più che la rappresentazione completa dell'intento. L'omissione del "why" può riflettere una

minore capacità del modello di catturare la relazione tra funzionalità e obiettivo in contesti complessi, suggerendo che Mistral tende a concentrarsi maggiormente sulle singole azioni, mentre LLaMA mantiene meglio la struttura completa della user story.

Gli attori e le funzionalità sono generalmente corretti in entrambi i modelli, anche se in Mistral si osserva più spesso ambiguità nella scelta dei ruoli.

Con la tecnica **Zero-shot**, le differenze diventano più marcate: LLaMA mostra una capacità di generalizzazione superiore, generando user stories più complete e coerenti, mentre Mistral appare maggiormente penalizzato, con omissioni sia nel *why* sia nella strutturazione complessiva delle user stories.

Per tecniche più complesse come **Chain-of-Thought** o **Prompt Chaining**, LLaMA continua a mostrare un vantaggio, gestendo meglio i passaggi sequenziali e la logica più articolata, probabilmente grazie a un training più ottimizzato per scenari di maggiore complessità.

In sintesi, entrambi i modelli sono validi strumenti per la generazione automatica di user stories, ma LLaMA risulta leggermente più performante in scenari complessi o in assenza di esempi, mentre Mistral eccelle solo quando viene guidato da prompt *Few-shot*, sebbene con maggiore tendenza a semplificazioni o omissioni nel *why*.

Le considerazioni appena effettuate ci permettono di rispondere alla RQ2.

🔗 **Risposta alla RQ<sub>2</sub>.** LLaMA ha mostrato performance leggermente superiori rispetto a Mistral, con uno scarto medio di circa il 3% su tutte le metriche considerate. Inoltre, Mistral tende a omettere il *why* nelle user stories. Queste differenze indicano un vantaggio consistente ma non marcato, suggerendo che entrambi i modelli sono generalmente comparabili nel contesto della generazione automatica di user stories.

## 5.3 RQ3. Validità dell'utilizzo di LLM

L'analisi dei risultati mostra che l'utilizzo di LLM rappresenta un metodo promettente per generare user stories a partire dai requisiti software. Le user stories prodotte dai modelli sono risultate generalmente coerenti, complete e grammaticalmente corrette, includendo le informazioni fondamentali "who", "what" e "why". L'analisi

qualitativa ha evidenziato che, sebbene possano comparire discrepanze minori negli attori o nelle formulazioni del goal, gli output mantengono una struttura chiara e comprensibile, idonea all'uso pratico.

Le metriche automatiche confermano questi risultati: valori medi di BLEU e METEOR superiori a 0.55 e BERTScore oltre 0.70 indicano una buona similarità lessicale e semantica rispetto agli output attesi. ROUGE-1 mostra che la maggior parte dei contenuti chiave dei requisiti viene catturata correttamente nelle user stories generate. La combinazione di metriche lessicali e semantiche consente quindi di verificare sia la correttezza terminologica sia la fedeltà informativa degli output dei modelli.

Dal confronto tra modelli e tecniche emerge inoltre un quadro più dettagliato: con la tecnica Few-shot, entrambi i modelli raggiungono performance elevate, confermando che fornire esempi concreti guida il modello e riduce omissioni o ambiguità. Tuttavia, l'analisi qualitativa mostra differenze interessanti: ad esempio, Mistral tende a omettere più frequentemente il "why", indicando una minore capacità di collegare la funzionalità all'obiettivo, mentre LLaMA mantiene meglio la struttura completa della user story. Questa osservazione suggerisce che l'uso di esempi contestuali è cruciale per ottenere output completi e che alcuni modelli possono privilegiare la generazione di azioni isolate rispetto alla relazione semantica completa tra componenti della user story.

Riassumendo, i risultati quantitativi e qualitativi confermano che l'impiego di LLM per la generazione automatica di user stories è non solo fattibile, ma produce output di qualità elevata e utilizzabile in contesti reali. L'approccio risulta quindi valido, anche in assenza di dataset standardizzati, offrendo una soluzione scalabile e adattabile alle esigenze dei progetti software.

🔗 **Risposta alla RQ<sub>3</sub>.** Sì, l'utilizzo di LLM è un metodo valido e promettente per generare automaticamente user stories a partire da requisiti software, in grado di produrre output coerenti, completi e comprensibili, con prestazioni elevate soprattutto se supportato da esempi contestuali.

---

### Minacce alla validità

---

In questa sezione vengono discusse le principali minacce alla validità dello studio, distinguendo tra validità interna, esterna, di costruzione e di conclusione.

#### 6.1 Validità interna

Una prima minaccia riguarda la formulazione dei prompt: variazioni anche minime nel testo possono influenzare i risultati ottenuti, introducendo una componente di variabilità difficilmente controllabile. Per ridurre tale effetto, i prompt sono stati costruiti seguendo linee guida coerenti e standardizzate, e ogni tecnica è stata testata più volte.

Un secondo elemento critico è la configurazione dei modelli, poiché parametri come temperatura, lunghezza massima dell'output e strategie di decoding incidono sensibilmente sulla qualità della generazione. In questo lavoro tali parametri sono stati mantenuti ai valori di default, fatta eccezione per il numero massimo di token in output, esteso a 2048. Questa scelta è stata effettuata per garantire trasparenza e riproducibilità, ma resta l'assenza di un'analisi di sensibilità che avrebbe consentito di comprendere meglio l'impatto di ciascun parametro.

Infine, le risorse hardware possono introdurre ulteriori fattori di variabilità. Per questo motivo gli esperimenti sono stati condotti su Google Colab, mantenendo costanti le configurazioni disponibili e riducendo il rischio che differenze di prestazioni dipendessero da vincoli computazionali.

## 6.2 Validità esterna

Un limite rilevante è rappresentato dalla natura del dataset, che comprende esclusivamente requisiti provenienti dal dominio medico. Ciò riduce la possibilità di generalizzare i risultati ad altri contesti applicativi, nei quali modelli e prompt potrebbero comportarsi in maniera diversa. D'altra parte, la scelta di un dominio uniforme ha consentito di focalizzare l'analisi sul confronto metodologico tra tecniche di prompting e modelli, minimizzando la variabilità introdotta da differenze di dominio.

Un'ulteriore minaccia riguarda il numero limitato di modelli analizzati, circoscritto a LLaMA 3.1 8B e Mistral 7B. Questa selezione, guidata da motivazioni di disponibilità open-source e vincoli computazionali, non esaurisce l'ampio panorama di LLM oggi disponibili, ma ha permesso di garantire la fattibilità dello studio in un contesto sperimentale controllato.

## 6.3 Validità di costruzione

La valutazione della qualità delle user stories è stata basata esclusivamente su metriche automatiche (BLEU, ROUGE, METEOR, BERTScore), che stimano la similarità lessicale e semantica rispetto alle reference prodotte dall'oracolo. Tali misure, pur offrendo un quadro quantitativo utile e robusto grazie alla loro combinazione, non riflettono aspetti più qualitativi come chiarezza, coerenza interna o utilità percepita da un team di sviluppo. L'assenza di una valutazione umana rappresenta quindi una minaccia rilevante, che andrebbe affrontata in studi futuri integrando metriche qualitative.

## 6.4 Validità di conclusione

Le conclusioni tratte si basano su un numero ridotto di progetti e non sono supportate da analisi statistiche di significatività, poiché la dimensione del campione non lo consentirebbe. Le differenze osservate tra tecniche di prompting e tra modelli vanno pertanto interpretate come indicazioni preliminari. Lo studio si colloca quindi in una prospettiva esplorativa, con l'obiettivo di fornire basi metodologiche che potranno essere validate successivamente su campioni più ampi e con strumenti statistici più solidi.



## CAPITOLO 7

---

### Conclusioni

---

I risultati ottenuti indicano che i Large Language Models rappresentano attualmente il metodo più valido e promettente per la generazione automatica di user stories a partire dai requisiti software. Rispetto ai metodi precedentemente utilizzati in letteratura, i LLM sono in grado di produrre user stories coerenti, complete e semanticamente corrette, includendo le informazioni chiave *who*, *what* e *why*.

L'analisi comparativa delle tecniche di prompt ha evidenziato che alcune strategie, come Meta-prompting, Prompt Directional Stimulus e soprattutto Few-shot, migliorano sensibilmente le performance, suggerendo margini di ottimizzazione futuri. Inoltre, il confronto tra modelli (LLaMA vs Mistral) ha mostrato che modelli più avanzati tendono a generare output più accurati, confermando il potenziale dell'approccio.

In questo modo, il lavoro ha fornito risposte concrete alle tre RQ individuate:

- RQ1, identificando le tecniche di prompt più efficaci;
- RQ2, determinando quale modello LLM mostra le migliori performance nel contesto analizzato.
- RQ3, confermando la validità e la promettente applicabilità dei LLM;

Per sviluppi futuri, sarebbe interessante aumentare il numero di progetti analizzati per consentire analisi statistiche di significatività più robuste. Inoltre, l'impiego di LLM più recenti rispetto a quelli utilizzati in questo lavoro potrebbe ulteriormente migliorare i risultati e consolidare le indicazioni già ottenute.

La *repository* di **Github** contenente tutte le informazioni relative al progetto è disponibile al seguente link:

`https://github.com/GiovanniBorrelli/Software-Requirements-to-User-Stories`

---

## Bibliografia

---

- [1] E. Gentili and D. Falessi, "Characterizing requirements smells," in *Lecture Notes in Computer Science*. Springer Nature, 2023, pp. 387–398. (Citato a pagina 2)
- [2] J.-M. Bruel, S. Ebersold, F. Galinier, A. Naumchev, M. Mazzara, and B. Meyer, "The role of formalism in system requirements," *arXiv preprint arXiv:1911.02564*, 2019. [Online]. Available: <https://arxiv.org/abs/1911.02564> (Citato a pagina 2)
- [3] A. Ibrahim and A. Ahmad, "Automated requirements engineering framework for agile model-driven development," *Frontiers in Computer Science*, 2025, includes tools such as RACE, UML Generator translating requirements documents into UML artifacts via NLP. [Online]. Available: <https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2025.1537100/full> (Citato a pagina 3)
- [4] P. Kumar and P. Bhatia, "Software requirement analysis: Research challenges and technical approaches," in *Proceedings of the 2018 International Conference on Computing, Power and Communication Technologies (GUCON)*, 2018. [Online]. Available: [https://www.researchgate.net/publication/324997752\\_Software\\_requirement\\_analysis\\_Research\\_challenges\\_and\\_technical\\_approaches](https://www.researchgate.net/publication/324997752_Software_requirement_analysis_Research_challenges_and_technical_approaches) (Citato a pagina 7)

- 
- [5] A. van Lamsweerde, "Requirements engineering in the year 00: A research perspective," *Software Engineering Journal / Information and Software Technology*, 2000. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S0164121210002876> (Citato a pagina 7)
- [6] S. Wagner, A. Ferrari, L. Guzmán, H. Femmer *et al.*, "Requirements quality research: a harmonized theory, evaluation, and roadmap," *arXiv preprint arXiv:2309.10355*, 2023. [Online]. Available: <https://arxiv.org/abs/2309.10355> (Citato a pagina 7)
- [7] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "Improving agile requirements: The quality user story framework and tool," *Requirements Engineering*, 2016. [Online]. Available: <https://doi.org/10.1007/s00766-016-0250-x> (Citato a pagina 7)
- [8] A. M. S. Ferreira, A. R. da Silva, and A. C. R. Paiva, "Towards the art of writing agile requirements with user stories, acceptance criteria, and related constructs," in *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. SciTePress, 2022. (Citato a pagina 7)
- [9] T. Editorial, "Requirements vs. user stories in software development," <https://www.techtarget.com/searchsoftwarequality/tip/Requirements-vs-user-stories-in-software-development>, July 2024. (Citato a pagina 8)
- [10] A. L. Inc., "User stories vs requirements: Which is the difference?" <https://www.aha.io/blog/user-stories-vs-product-requirements>, February 2025. (Citato a pagina 8)
- [11] Scrum.org, "Are user stories requirements?" <https://www.scrum.org/resources/blog/are-user-stories-requirements>, 2021. (Citato a pagina 8)
- [12] E. Dietrich, "How to turn requirements into user stories," <https://daedtech.com/how-to-turn-requirements-into-user-stories/>, 2014, accessed: 2025-08-30. (Citato a pagina 8)

- 
- [13] A. Aysh, T. I. Ayad, and A.-H. Rawan, "The use of artificial neural networks for extracting actions and actors from requirements document," *Information and Software Technology*, 2018. (Citato a pagina 11)
- [14] A. Iyad and A. T. Imam, "An algorithmic approach to extract actions and actors (aaeea)," *Conference Paper*, 2018. (Citato a pagina 11)
- [15] A. Aysh, T. I. Ayad, and W. AL-zyadat, "Svm machine learning classifier to automate the extraction of srs elements," *International Journal of Advanced Computer Science and Applications*, 2021. (Citato a pagina 11)
- [16] A. T. Imam, "The automatic definition of the intuitive linguistic heuristics set to recognize the elements of uml analysis and design models in english," *Information Technology*, 2023. (Citato a pagina 11)
- [17] T. Rahman and Y. Zhu, "Automated user story generation with test case specification using large language model," *arXiv preprint arXiv:2404.01558*, 2024. [Online]. Available: <https://arxiv.org/abs/2404.01558> (Citato a pagina 13)
- [18] Z. Zhang, M. Rayhan, T. Herda, M. Goisau, and P. Abrahamsson, "Llm-based agents for automating the enhancement of user story quality: An early report," *arXiv preprint arXiv:2403.09442*, 2024. [Online]. Available: <https://arxiv.org/abs/2403.09442> (Citato a pagina 13)
- [19] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A systematic survey of prompt engineering in large language models: Techniques and applications," *arXiv preprint arXiv:2402.07927*, 2024, survey on prompt engineering techniques, applications, and challenges. [Online]. Available: <https://arxiv.org/abs/2402.07927> (Citato a pagina 26)
- [20] J. Shin, C. Tang, T. Mohati, M. Nayeibi, S. Wang, and H. Hemmati, "Prompt engineering or fine tuning: An empirical assessment of large language models in automated software engineering tasks," *arXiv preprint arXiv:2310.10508*, 2023, basic prompting (naïve) compared with prompting engineering techniques. [Online]. Available: <https://arxiv.org/abs/2310.10508> (Citato a pagina 26)

- [21] L. Gao, "Harnessing the intrinsic knowledge of pretrained language models for challenging text classification settings," Master's thesis, University of Copenhagen, 2024, master's Thesis. [Online]. Available: <https://arxiv.org/abs/2408.15650> (Citato a pagina 28)
- [22] T. B. B. et al., "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020, introduces few-shot prompting providing examples to guide LLMs without fine-tuning. [Online]. Available: <https://arxiv.org/abs/2005.14165> (Citato a pagina 29)
- [23] J. Xin, S. Tan, B. Wang, X. L. Chen, M.-Y. Kan, and D. Dahlmeier, "A survey of hallucination in natural language generation," *ACM Computing Surveys*, 2023. (Citato a pagina 32)
- [24] S. Sun, R. Yuan, Z. Cao, W. Li, and P. Liu, "Prompt chaining or stepwise prompt? refinement in text summarization," *Preprint*, 2024, compares prompt chaining vs. stepwise prompting showing chaining improves summarization accuracy. [Online]. Available: <https://arxiv.org/abs/2406.00507> (Citato a pagina 34)
- [25] Z. Li, B. Peng, P. He, M. Galley, J. Gao, and X. Yan, "Guiding large language models via directional stimulus prompting," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023, introduces mild constraints via instance-specific auxiliary prompts to balance model freedom and output control. [Online]. Available: <https://arxiv.org/abs/2302.11520> (Citato a pagina 35)
- [26] M. Suzgun and A. T. Kalai, "Meta-prompting: Enhancing language models with task-agnostic scaffolding," *arXiv preprint arXiv:2401.12954*, 2024, IM acts as conductor, decomposing tasks and coordinating expert instances. [Online]. Available: <https://arxiv.org/abs/2401.12954> (Citato a pagina 36)
- [27] A. Piché, A. Milios, D. Bahdanau, and C. Pal, "Llms can learn self-restraint through iterative self-reflection," *arXiv preprint arXiv:2405.13022*, 2024, introduces ReSearch: iterative self-prompting and self-evaluation

- to enable adaptive, self-correcting outputs. [Online]. Available: <https://arxiv.org/abs/2405.13022> (Citato a pagina 38)
- [28] C. Chhun, P. Colombo, F. M. Suchanek, and C. Clavel, "Of human criteria and automatic metrics: A benchmark of the evaluation of story generation," in *Proceedings of the 29th International Conference on Computational Linguistics (COLING)*. International Committee on Computational Linguistics, 2022. [Online]. Available: <https://aclanthology.org/2022.coling-1.509/> (Citato a pagina 40)
- [29] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," in *International Conference on Learning Representations (ICLR)*, 2020, preprint available at arXiv:1904.09675. [Online]. Available: <https://arxiv.org/abs/1904.09675> (Citato a pagina 40)
- [30] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020. [Online]. Available: <https://arxiv.org/abs/2005.14165> (Citato a pagina 44)

---

## Ringraziamenti

---

*Grazie a chi ha reso possibile questa tesi:  
Ai miei genitori per le possibilità che mi hanno dato,  
al mio relatore di tesi per la disponibilità dimostrata,  
e al mio tutor Francesco per il supporto ricevuto.*

*Grazie a tutte le persone che ho incontrato  
durante il mio percorso universitario:  
La persona che sono diventata oggi  
è merito di ognuno di loro.*