UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

TESI DI LAUREA

Definizione e Valutazione di un Catalogo di Template di Prompt Engineering per la Sicurezza e la Qualità del Codice Generato da Large Language Model

RELATORE

Prof. Fabio Palomba

Dott. Giammaria Giordano

Università degli Studi di Salerno

CANDIDATO

Matteo Cicalese

Matricola: 0522501516

Anno Accademico 2023-2024

Questa tesi è stata realizzata nel

Never settle

Abstract

I Large Language Model (LLM) hanno acquisito una rilevanza cruciale nel contesto dello sviluppo software, consentendo agli sviluppatori di accelerare e ottimizzare i processi di produzione. Tuttavia, la loro capacità di generare codice sicuro e qualitativo e la loro efficacia in contesti di produzione reali rappresentano questioni ancora dibattute.

L'obiettivo di questa tesi è determinare l'efficacia di vari approcci di prompt engineering sulla sicurezza e la qualità del codice prodotto. A tale scopo, sono state valutate diverse strategie, definendo un catalogo di template di prompting e determinando i più efficaci in base all'impatto riscontrato sulla sicurezza e la qualità del codice generato. I risultati preliminari ottenuti hanno guidato la sperimentazione definitiva, effettuata su un campione più ampio di problematiche di programmazione, rappresentativo della complessità e dell'eterogeneità di applicazioni reali.

I risultati della sperimentazione effettuata hanno evidenziato un miglioramento rispetto alla baseline fino al 53% rispetto ai CWE e fino al 37% rispetto alla qualità del codice, con la riduzione delle tipologie di debolezze riscontrate, l'eliminazione dei problemi di qualità più gravi ed un impatto statisticamente significativo su tutti i modelli considerati.

Questo studio dimostra l'efficacia del prompt engineering nella mitigazione di aspetti di sicurezza e qualità del codice, proponendo un approccio diretto ed efficace in scenari di produzione reali, in grado di migliorare le prestazioni fornite dai LLM.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

Questo capitolo fornisce una panoramica generale dello studio condotto, sottolineando l'impatto dei LLM nella società, e introducendo il prompt engineering come soluzione per ottimizzare l'interazione con tali modelli. Sono esposte le motivazioni che guidano questo studio, stabilendo come obiettivo la definizione e valutazione di un catalogo di template di prompt engineering per la sicurezza e la qualità del codice generato dai LLM. Vengono poi presentati i risultati principali dello studio, definendo infine la struttura generale del documento.

1.1 Contesto

Un Large Language Model (LLM) è un sistema di intelligenza artificiale, addestrato su vasti insiemi di dati testuali, progettato per l'elaborazione e la manipolazione del linguaggio naturale. Grazie alla capacità di apprendere le strutture linguistiche e le relazioni semantiche presenti nei dati di addestramento, questi modelli sono in grado di eseguire una vasta gamma di compiti, come l'analisi testuale, la sintesi di informazioni, e la generazione di contenuti [2].

I LLM stanno avendo un impatto profondo in ogni ambito della società, rivoluzionando le modalità con cui individui e organizzazioni operano in contesti professionali [3]. L'adozione di tali tecnologie ha fatto riscontrare un significativo miglioramento della qualità dei processi e dell'efficienza operativa, con una notevole riduzione dei tempi di produzione [4]. In campo informatico, questi modelli hanno consentito la semplificazione di attività complesse come il mining e l'analisi dei dati [5], l'identificazione di vulnerabilità [6], e la generazione di codice sorgente [7]. Tuttavia, sono presenti diverse questioni legate all'affidabilità e alla correttezza delle loro risposte, specie in virtù del loro impiego in settori dove sicurezza e qualità rappresentano requisiti critici [8, 4, 9].

Una soluzione sempre più affermata rispetto a tale problematica è il **prompt engineering**, una tecnica di formulazione e ottimizzazione delle richieste rivolte ai LLM, utilizzata per ottenere risposte più accurate e pertinenti rispetto alle problematiche poste [10]. Questa pratica consente di affinare la qualità e la precisione dei risultati generati, migliorando la comprensione del modello rispetto al compito richiesto. Nel contesto dello sviluppo software, il prompt engineering ha dimostrato di essere in grado di migliorare la correttezza, la sicurezza, e l'efficienza del codice generato, ragione per cui sta emergendo come una delle tecniche più valide per ottimizzare l'interazione con tali modelli [11, 12].

1.2 Motivazioni e Obiettivi

Nell'attuale scenario dello sviluppo software, dove i requisiti di sicurezza e qualità del codice sono sempre più rigorosi, è essenziale che i LLM garantiscano prestazioni eccellenti. L'adozione di tecniche di prompt engineering in tal senso assume dunque un'importanza strategica, in quanto può incidere significativamente sulla qualità complessiva delle risposte fornite [13, 12]. Sebbene il prompt engineering stia dimostrando la sua efficacia in diversi ambiti, l'effetto di queste tecniche non è stato ancora esplorato in profondità, soprattutto rispetto a contesti di produzione reali, caratterizzati da problematiche complesse ed eterogenee [14]. Queste limitazioni guidano l'obiettivo primario di questo studio, che prevede la definizione e la valutazione di un catalogo di template di prompt engineering per la sicurezza e la qualità del codice generato dai LLM, analizzando le implicazioni pratiche della loro applicazione su problematiche di programmazione reali. Nello specifico, sono stati posti i seguenti obiettivi:

- **Approcci di Prompting**: Saranno definiti e valutati due approcci preliminari, ingegnerizzando diversi template di prompting, e determinando la strategia migliore in base ai risultati osservati;
- **Costruzione Dataset**: Sarà costruito un apposito dataset, collezionando domande da Stack Overflow[1] che rappresentino in maniera estesa ed eterogenea problematiche del mondo reale;
- **Sperimentazione**: Ogni domanda sarà valutata in combinazione con diversi prompt e testata su varie tipologie di LLM, con lo scopo di determinare se l'efficacia della strategia di prompting definita sia generalizzabile ai modelli considerati;
- **Valutazione**: Ognuno degli snippet di codice generati sarà analizzato con Sonarlint[2], raccogliendo informazioni quantitative e qualitative sul livello di sicurezza e qualità delle osservazioni raccolte e determinando la bontà dell'approccio definito;

[1] `stackoverflow.com`

[2] `sonarsource.com/products/sonarlint`

1.3 Risultati

Dai risultati ottenuti emerge chiaramente che l'approccio definito ha un impatto significativo sulla sicurezza e la qualità del codice generato. In particolare, la maggiore complessità e specificità dei template utilizzati, combinata con il giusto equilibrio tra potenziale informativo e chiarezza della richiesta, ha portato a un netto miglioramento in termini di sicurezza e qualità del codice, dimostrando l'efficacia delle direttive impiegate. I risultati confermano l'importanza del prompt engineering nello sviluppo di codice sicuro e qualitativo, sottolineando come l'impiego di strategie di prompting dirette e pragmatiche possa sensibilmente migliorare la qualità dell'interazione con i modelli linguistici. In conclusione, lo studio condotto fornisce un importante contributo al prompt engineering, dimostrando l'efficacia dell'approccio definito nel miglioramento delle prestazioni dei LLM su applicazioni reali.

1.4 Struttura del Documento

Il documento è strutturato nei seguenti capitoli:

- **Capitolo 1**: Fornisce un contesto generale su LLM e prompt engineering, discutendo le motivazioni e gli obiettivi dello studio e fornendo una panoramica generale dei risultati ottenuti;
- **Capitolo 2**: Definisce la base teorica della ricerca, analizzando le sfide legate all'affidabilità dei LLM rispetto alla generazione del codice, e valutando l'efficacia del prompt engineering nella risoluzione di tali problematiche;
- **Capitolo 3**: Vengono definiti diversi approcci preliminari con lo scopo di determinare la strategia migliore rispetto agli obiettivi posti, andando ad ingegnerizzare diversi template di prompt engineering e valutandoli rispetto alla sicurezza e la qualità del codice generato;
- **Capitolo 4**: Descrive in maniera dettagliata la strategia, le metodologie e i vincoli definiti per la raccolta delle domande, il prompting ai modelli e l'analisi degli snippet di codice generati per la fase di sperimentazione definitiva;

- **Capitolo 5**: Attraverso un'analisi quantitativa e qualitativa, sono analizzati e confrontati i risultati ottenuti, valutando l'impatto della strategia e determinando la significatività dei miglioramenti osservati tramite test statistici;
- **Capitolo 6**: Vengono discusse le potenziali minacce alla validità dello studio e le misure adottate per mitigarne l'impatto;
- **Capitolo 7**: Sintetizza gli obiettivi, le implicazioni e i contributi del lavoro svolto, evidenziando l'impatto dell'approccio definito, discutendone le limitazioni e proponendo possibili sviluppi futuri.

CAPITOLO 2

Background

Questo capitolo offre un approfondimento teorico sui LLM, discutendo il loro impatto sullo sviluppo software moderno e sottolineando come la loro affidabilità sia ancora una questione aperta. Viene introdotto il prompt engineering come soluzione a tali problematiche, descrivendo le tecniche più note in letteratura. Viene valutata la loro capacità di migliorare i risultati ottenuti dall'interazione con tali modelli, evidenziandone al contempo i limiti. Il capitolo si conclude con una panoramica sull'uso dei LLM per la generazione di codice, descrivendo le criticità attuali e il ruolo cruciale del prompt engineering in quest'ambito.

2.1 Large Language Models

I Large Language Model (LLM) sono sistemi di intelligenza artificiale basati su deep neural network, progettati per la comprensione, l'elaborazione e la manipolazione del linguaggio naturale. Questi modelli, addestrati su vaste quantità di dati e costituiti da miliardi di parametri, riescono a catturare con precisione le sfumature semantiche e sintattiche del linguaggio, permettendo una comprensione contestuale avanzata e la formulazione di risposte coerenti e sofisticate [2]. La capacità di individuare e rappresentare relazioni complesse all'interno del testo li rende strumenti estremamente potenti e versatili in una vasta gamma di applicazioni, con prestazioni oramai eccellenti in molteplici aree di conoscenza [4]. Questi modelli costituiscono risorse fondamentali per l'accelerazione e l'automazione di processi, come dimostrato dall'impatto profondo avuto in diversi settori, come sanità, istruzione e finanza [3].

2.1.1 Impatto dei LLM

Nello sviluppo software moderno i LLM hanno trasformato radicalmente il modo in cui i programmatori interagiscono con le attività di ingegneria del software, consentendo l'automazione di compiti ripetitivi e la semplificazione dei processi di scrittura, analisi e comprensione del codice [15]. Un report recente ha rivelato che il 97% degli sviluppatori integra i LLM nel proprio processo di sviluppo [16]. Questa tecnologia non solo permette di migliorare significativamente l'efficienza e la produttività degli sviluppatori, ma permette anche di accelerare l'intero ciclo di sviluppo, con un notevole risparmio in termini di tempo e risorse ed un miglioramento della qualità generale del prodotto [3, 4].

Con un impiego sempre più ampio di tali tecnologie sono aumentate conseguentemente anche le aspettative rispetto alle loro prestazioni. Negli ultimi anni la ricerca sui LLM ha subito una notevole accelerazione, portando a modelli sempre più precisi e intelligenti [17]. L'evoluzione tecnologica e l'ottimizzazione dei dati di addestramento hanno permesso a questi modelli di ottenere prestazioni sempre migliori, con miglioramenti significativi in termini di capacità di elaborazione e generazione di contenuti complessi [17, 4].

2.1.2 Affidabilità dei LLM

Nonostante le loro enormi capacità, i LLM presentano diverse problematiche legate all'affidabilità e alla correttezza delle risposte fornite. Uno dei principali limiti è rappresentato dalla tendenza a generare informazioni inesatte, incomplete o fuorvianti. Questo fenomeno, definito come *allucinazioni*, può portare tali modelli a fornire risposte che sembrano plausibili ma che in realtà sono errate o prive di fondamento [9]. Inoltre, i LLM non possiedono una capacità di ragionamento autonomo, ma generano risposte in base ai pattern statistici e linguistici derivati dai dati di addestramento. Di conseguenza, possono fallire nel ragionamento logico o nell'applicazione di conoscenze rispetto a determinate problematiche [8].

Anche rispetto allo sviluppo software queste tecnologie presentano diverse criticità. In ambito di assistenza e debugging del codice, possono fornire suggerimenti inaccurati o introdurre nuovi errori, in quanto spesso non raggiungono comprensione completa del contesto specifico della richiesta [18]. Nella generazione di documentazione e il supporto generale alla progettazione i risultati forniti potrebbero non risultare conformi ai requisiti progettuali, ignorando vincoli tecnici o di business cruciali [19, 15]. In attività di generazione automatica di contenuti esiste il rischio di produrre codice inefficiente, insicuro e malformato, a causa della mancanza di dati di addestramento sufficientemente estesi e qualitativi [7].

Per rispondere a tali criticità, molti sviluppatori si sono mossi verso il **fine-tuning**, un approccio che consiste nel riaddestrare un modello su un dataset specifico per un determinato compito, affinando così le sue prestazioni in un certo dominio [20]. Sebbene abbia dimostrato di essere in grado di portare buoni miglioramenti in diverse applicazioni, si tratta di un processo che richiede risorse computazionali significative, una quantità considerevole di dati opportunamente etichettati, e tempi prolungati per l'addestramento, il che lo rende poco praticabile in contesti con risorse limitate o tempistiche ristrette [20, 21].

Il potenziale dei LLM tuttavia non dipende esclusivamente dalle loro capacità tecniche, ma soprattutto dal modo in cui questi vengono utilizzati. Sebbene questi modelli siano in grado di comprendere e generare codice complesso, è essenziale che le richieste degli utenti siano formulate correttamente, in modo da poter sfruttare

appieno le loro capacità [1]. Con l'aumento delle aspettative nei confronti degli LLM, è emersa la necessità di sviluppare approcci alternativi per migliorare l'interazione con questi modelli, dando origine al prompt engineering.

2.2 Prompt Engineering

Il **prompt engineering** è una tecnica che consiste nel formulare e organizzare in maniera specifica gli input, noti come **prompt**, con lo scopo di indirizzare l'output del modello verso risultati più efficaci e pertinenti rispetto al problema posto [10]. A differenza di una normale domanda che esprime in maniera essenziale una certa richiesta, con questa tecnica vengono opportunamente introdotte istruzioni, informazioni contestuali o esempi, in maniera coerente con la problematica posta, per guidare il modello verso una comprensione più profonda della richiesta e la generazione di risposte più accurate [1]. L'origine di questa disciplina è strettamente connessa alla natura intrinseca di questi modelli. Infatti, poichè i LLM non comprendono il linguaggio naturale come gli esseri umani, è essenziale formulare con precisione le richieste affinchè il modello le interpreti correttamente e generi risposte appropriate.

Il prompt engineering prevede la manipolazione di diverse componenti della richiesta [1]:

- **Direttive**: Servono per fornire una direzione chiara e precisa sul compito che si vuole eseguire, in maniera da ridurre l'ambiguità e focalizzare il modello su una determinato processo;
- **Esempi**: Costituiscono una dimostrazione che funge da riferimento per la costruzione della risposta, in modo da guidare i modelli nella realizzazione di un certo compito secondo una logica ben definita;
- **Struttura dell'Output**: Definisce le aspettative rispetto al risultato prodotto dal modello, come il formato, la struttura e lo stile della risposta, per garantire che la risposta sia presentata in modo conforme alle esigenze dell'utente.

Non tutti gli elementi devono essere utilizzati contemporaneamente, infatti i prompt possono basarsi esclusivamente su uno di questi elementi o una combinazione di essi,

a seconda dell'obiettivo e della complessità del compito considerato. Un esempio in merito è mostrato in Figura 2.1.

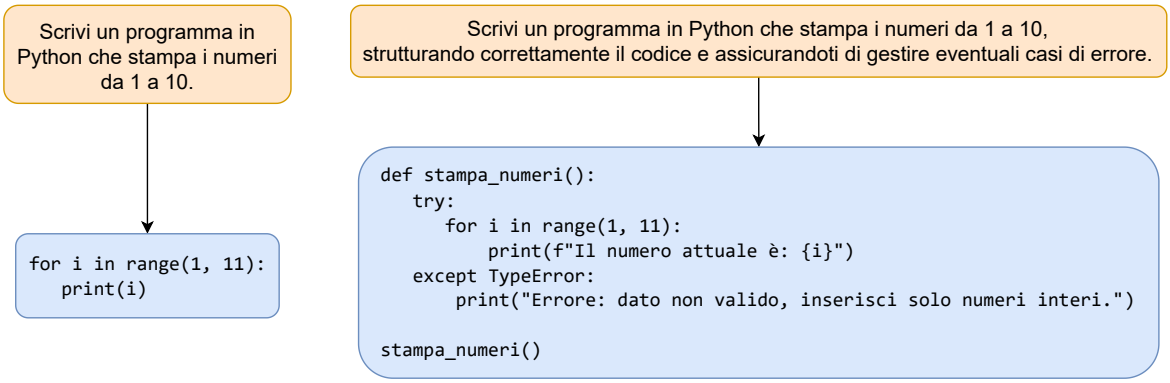

Figura 2.1: Effetti del Prompt Engineering sulla Generazione di Codice

2.2.1 Tecniche Basate su Inferenza

Considerando che il prompt engineering si basa sulla manipolazione delle richieste, formulate in forma testuale e quindi intrinsecamente legate al linguaggio, sono molteplici gli approcci che consentono di applicare queste tecniche in modo efficace. In letteratura, la maggior parte degli studi si concentra su **tecniche basate su inferenza**, che fanno leva sulle abilità dei modelli di apprendere schemi e pattern logici dalle richieste fornite, applicando tali conoscenze per la risoluzione di problemi specifici. Queste tecniche sfruttano varie modalità di interazione con i modelli, con lo scopo di massimizzarne l'efficacia in relazione al problema fornito:

- **Zero-shot Prompting** [22]: Si basa sul fornire una singola richiesta al modello, senza fornire esempi specifici rispetto al problema posto, facendo sì che il modello interpreti la richiesta solo in base alle informazioni contenute nel prompt;
- **Few-shot Prompting** [22]: Vengono forniti alcuni esempi di input e/o output per guidare il modello nella realizzazione di una soluzione che risulti pertinente rispetto alla problematica posta;
- **Chain-of-Thought** [23]: Piuttosto che utilizzare un unico prompt ricco di informazioni, il prompt viene suddiviso in parti più piccole, in modo che il modello

sia guidato attraverso un ragionamento passo-passo per risolvere un problema complesso;

- **Retrieval-Augmented Generation** [24]: Integra il recupero di informazioni da una base di conoscenza con la generazione di testo tramite LLM, utilizzando query mirate per estrarre i dati rilevanti e impiegarli nella formulazione di risposte dettagliate;

- **Self-Consistency** [25]: A partire da uno stesso prompt vengono generate diverse risposte seguendo percorsi di ragionamento alternativi, e successivamente confrontate per selezionare la più coerente.

Le tecniche illustrate rappresentano solo alcune delle principali strategie di prompt engineering. Tuttavia, esiste un'ampia varietà di tecniche, che possono essere considerate personalizzazioni o versioni più avanzate delle tecniche citate. Sahoo et al. [22] hanno proposto una possibile tassonomia, mostrata in Figura 2.2, che organizza e classifica le tecniche più note in letteratura. Nonostante il crescente interesse per il prompt engineering e la sua comprovata efficacia nell'interazione con i modelli linguistici, non esiste ancora un sistema di classificazione di riferimento consolidato. Sebbene siano stati proposti vari framework e metodologie, manca un consenso e una standardizzazione definitivi, evidenziando la necessità di ulteriori ricerche per stabilire una struttura teorica più solida e condivisa.

2.2.2 Tecniche Basate su Direttive

In letteratura scientifica è presente un sottoinsieme di **tecniche basate su direttive**, che tramite la manipolazione e la strutturazione diretta della richiesta sono in grado di migliorare la qualità delle risposte prodotte:

- **Instruction-based Prompting** [26]: Utilizza istruzioni esplicite per guidare il modello verso l'esecuzione di un compito specifico, in modo da ridurre l'ambiguità e garantire che il modello risponda in maniera focalizzata;

- **Context-aware Prompting** [27]: Integrando informazioni contestuali nel prompt, permette di generare risposte più pertinenti e coerenti con la richiesta fornita, risultando molto utile per mantenere la continuità e la rilevanza nelle richieste;

- **Role-based Prompting** [28]: Assegna al modello un ruolo specifico, come quello di esperto o assistente, per orientare la risposta attraverso le competenze associate ad un certo ruolo, aumentando la naturalezza e l'efficacia dell'interazione;

- **Constraint-based Prompting** [29]: Imponendo vincoli precisi, come la lunghezza del testo, lo stile o il tono, permette di controllare l'output del modello, assicurando che rispetti le regole ed i vincoli formali forniti nella richiesta.

Oltre alle principali tecniche basate su direttive che guidano l'interazione con i LLM, una tecnica di prompt engineering emergente è l'**Emotional Stimuli**. Tale tecnica si distacca da dettagli tecnici o specifiche sintattiche, in quanto utilizza strategie che includono richieste di attenzione e contesto emotivo. Facendo leva su elementi linguistici che evocano emozioni ed enfatizzano l'importanza della richiesta posta, il modello viene stimolato a rispondere in modo più preciso [30] [31].

È importante considerare che l'efficacia di una richiesta non dipende esclusivamente dalla scelta degli elementi che compongono il prompt, ma soprattutto dal modo in cui questi vengono ingegnerizzati, manipolati e organizzati, mantenendo una struttura che massimizzi la qualità e la precisione della risposta del modello [1, 12]. Inoltre, la ricerca in questo campo ha dimostrato che anche quando si utilizzano elementi fondamentali molto simili, la loro manipolazione può pesantemente influenzare il risultato ottenuto [32].

2.2.3 Vantaggi e Criticità

Con il crescente impiego dei LLM nei vari settori dell'informatica, l'attenzione verso il prompt engineering è aumentata significativamente [12].

Studi recenti hanno evidenziato come il prompt engineering possa raggiungere prestazioni comparabili o superiori al fine-tuning in diversi task di elaborazione del linguaggio naturale. Nello specifico, Schick e Schütze hanno dimostrato che, attraverso l'uso di prompt opportunamente progettati, è possibile ottenere ottimi risultati in task generici come classificazione testuale e inferenza logica senza la necessità di fine-tuning, dimostrando l'efficienza e la praticità di tale approccio nell'adattamento di tali modelli ad esigenze specifiche [21]. Inoltre Gao et. al., sfruttando tecniche

di Retrieval-Augmented Generation, sono stati in grado di ottenere risultati che superano di gran lunga quelli del fine-tuning [24].

Negli ultimi anni sono stati condotti diversi studi che hanno dimostrato che modifiche apparentemente semplici ai prompt possono influenzare in modo significativo le prestazioni dei LLM. Ad esempio, Reynolds e McDonell hanno evidenziato come l'inserimento di contesto specifico e istruzioni che enfatizzano la sicurezza del codice possa migliorare notevolmente la qualità dell'output generato [33]. Inoltre studi recenti hanno evidenziato come l'adozione di tecniche avanzate di prompt engineering possa migliorare le prestazioni dei LLM in vari compiti [24, 25]. Nello specifico, tali tecniche hanno mostrato una notevole efficacia in attività come l'elaborazione del linguaggio naturale [27], la rilevazione e mitigazione di bias [34], ma anche nel contesto dell'analisi e mining dei dati [5], la generazione di documentazione [19] e la sicurezza informatica [35, 11].

Nel caso delle tecniche basate su direttive, sebbene la loro natura diretta e pratica le rendano particolarmente adattabili a qualsiasi contesto, i loro effetti sono stati esplorati solo marginalmente in letteratura, in particolare su processi di generazione di testo e ragionamento, mostrando comunque ottimi risultati [26, 27].

Tuttavia, si sta ponendo un'attenzione sempre maggiore verso tecniche che sfruttano direttive di natura emotiva, come l'Emotional Stimuli. Questa tecnica, sfruttando richieste emotivamente significative sotto forma di una singola direttiva aggiunta alla richiesta, hanno dimostrato in maniera molto efficace di migliorare l'attenzione e la precisione delle risposte dei LLM [30] [31]. Lo studio di Li et. al. ha definito una serie di prompt emozionali, ovvero frasi psicologiche che aggiunte ai prompt originali migliorano le risposte del modello. La valutazione delle performance di tali prompt emozionali su diversi LLM ha evidenziato miglioramenti che vanno dall'8% per task di deduzioni di compiti, fino al 115% su task decisionali, di comprensione e di ragionamento [30]. Ma et. al ha invece proposto un framework che combina prompt stimolanti e prompt strutturali. Testando i prompt risultanti su due dataset generici relativi a task di comprensione del testo e problem solving, sono stati riscontrati notevoli miglioramenti per la maggioranza delle osservazioni considerate [31]. Nonostante i risultati promettenti, questa emergente pratica di prompt engineering ha avuto applicazioni solo in settori come la psicologia computazionale, le scienze

sociali, o per task generici di problem solving, mentre la sua potenziale utilità in campo informatico rimane totalmente inesplorata.

Il prompt engineering presenta alcune sfide e limitazioni. Prima di tutto, la formulazione di prompt ottimali non è sempre intuitiva. L'identificazione di una struttura che garantisca il giusto equilibrio tra ricchezza informativa ed efficienza computazionale può richiedere diversi tentativi [13], rendendo il processo particolarmente oneroso in termini di tempo e risorse [32].

Un ulteriore criticità è rappresentata dalle limitazioni dei token per i LLM, per cui i prompt e le risposte non possono superare una certa lunghezza, limitando la capacità del modello di carpire ed elaborare le informazioni necessarie per la risoluzione di richieste complesse. Inoltre, anche con prompt ottimizzati, i LLM possono produrre codice insicuro o incompleto a causa della loro limitata capacità di cogliere pienamente il contesto semantico in cui il codice verrà utilizzato [6].

Inoltre, vanno considerate le problematiche generali associate alle tecniche di prompting, come aumento eccessivo della complessità e dei tempi computazionali, difficoltà nel reperimento di dati di esempio, allucinazioni, e la ridotta efficacia rispetto a problematiche complesse [13, 32].

Il prompt engineering dunque non è solo un complemento all'uso dei LLM, ma una componente essenziale per sfruttarne appieno il potenziale. Una progettazione accurata dei prompt permette di indirizzare il modello verso soluzioni più pertinenti, mitigando i rischi associati a risposte ambigue o errate, e migliorando significativamente la qualità e l'affidabilità dei risultati prodotti [12, 13]. La crescente complessità dei problemi affrontati con l'ausilio dei LLM richiede un'interazione più sofisticata con i modelli, rendendo l'impiego del prompt engineering un aspetto fondamentale per gli informatici e i professionisti che utilizzano queste tecnologie.

2.3 Generazione di Codice tramite LLM

La generazione automatica di codice rappresenta una delle applicazioni più rilevanti dei LLM in campo informatico [36]. Grazie ad un addestramento estensivo e mirato, questi modelli costituiscono per gli sviluppatori un potente strumento per l'accelerazione dei processi di sviluppo software [7, 15, 36].

Diversi studi hanno dimostrato che gli utenti tendono a preferire l'ausilio dei LLM rispetto alla consultazione e l'utilizzo di forum come Stack Overflow, per una serie di fattori che comprendono la completezza e la chiarezza del codice fornito, il livello di tecnicismo delle risposte, oltre che la velocità dell'interazione [37, 35, 38].

2.3.1 Problematiche di Sicurezza e Qualità

Nonostante i progressi recenti, la generazione di codice tramite LLM continua a presentare diverse problematiche. Una delle più critiche riguarda l'ambiguità delle richieste fornite dagli utenti, che può portare alla generazione di codice parzialmente o totalmente errato, a causa della mancanza di una comprensione profonda del contesto in cui il codice deve operare [39]. Inoltre, questi modelli possiedono una finestra di contesto limitata, rispetto alla quale non sempre riescono a interpretare in modo corretto le istruzioni e il contesto applicativo necessari a generare una soluzione appropriata [39, 32].

Tuttavia, le considerazioni maggiori sono rivolte alla sicurezza del codice generato. Diversi studi hanno evidenziato come i modelli generino spesso codice che contiene vulnerabilità di sicurezza. Questo avviene perchè i modelli tendono a replicare i pattern appresi dai dati di addestramento, che possono includere codice con vulnerabilità non riconosciute [11]. Tali vulnerabilità, se non rilevate dagli utenti, vengono integrate nel codice sorgente, con rischi enormi sull'integrità delle soluzioni sviluppate [8, 7].

Lo studio di Pearce et. al. analizza il codice generato da GitHub Copilot su varie problematiche di programmazione, rivelando che la maggior parte delle soluzioni generate conteneva vulnerabilità di sicurezza, anche rispetto semplici funzionalità [35]. Lo studio di Wang et. al. analizza le vulnerabilità generate da diversi modelli a partire da un dataset di domande appositamente curato, evidenziando in maniera comune varie difficoltà nella generazione di codice sicuro [11]. Hamer et. al hanno costruito un dataset di domande vulnerabili relative a Java selezionandole da Stack Overflow, e ricavato i CWE prodotti da GPT quando interrogato con le domande selezionate. I risultati ottenuti evidenziano che, sebbene i LLM abbiano fatto notevoli progressi, questi sono ancora affetti da problematiche di generazione di codice

vulnerabile [38]. Un altro studio, introduce SALLM [14], un framework che valuta la sicurezza del codice generato da diversi LLM, tramite un apposito dataset di problemi di programmazione reali relativi ad aspetti di sicurezza. I risultati evidenziano diverse criticità relative alla correttezza funzionale e la sicurezza del codice prodotto, sottolineando la necessità di appositi benchmark che effettuino valutazioni in modo mirato ed evidenziando come i dataset esistenti non rappresentino adeguatamente problematiche del mondo reale.

Un aspetto che risulta particolarmente trascurato in letteratura rispetto alla generazione di codice tramite LLM è la valutazione degli aspetti di qualità, in particolare considerando bug, code smell e problemi di aderenza a standard. Con l'adozione crescente di questi modelli in contesti produttivi, è essenziale che il codice generato sia non solo sicuro, ma risponda anche a standard di qualità adeguati rispetto ai contesti in cui viene impiegato.

Le problematiche evidenziate sottolineano che, per sfruttare appieno il potenziale dei LLM nella generazione di codice, è necessario adottare strategie che guidino il modello verso output più accurati e affidabili.

2.3.2 Prompt Engineering per la Generazione di Codice

Il prompt engineering applicato alla generazione di codice può risultare determinante per aumentare le prestazioni fornite da tali modelli. Attraverso la progettazione attenta dei prompt, è possibile fornire al modello indicazioni precise che ne orientano il comportamento, mitigando alcuni dei più noti problemi in merito [33].

La letteratura recente ha esplorato vari approcci di prompt engineering per ottimizzare la generazione di codice. Nello specifico, Chen et al. [39] hanno introdotto Codex, un modello specializzato nella generazione di codice, dimostrando come l'impiego di prompt dettagliati permetta di migliorare significativamente le prestazioni del modello. Inoltre, Austin et al. [40] hanno studiato come l'utilizzo del Few-Shot prompting possa migliorare la capacità del modello di realizzare programmi funzionalmente corretti, dimostrando che l'inclusione di esempi rappresentativi nel prompt può ridurre gli errori sintattici e semantici generati. Altri studi, come quello di Feng et al. [23], hanno esplorato l'utilizzo di tecniche di prompting sofisticate

come il Chain-of-Thought, per guidare il modello attraverso una sequenza di passaggi logici nella generazione di codice complesso. Tale strategia ha dimostrato di migliorare notevolmente la capacità del modello nella gestione di problematiche di programmazione complesse. Recentemente sono state sviluppate anche diverse tecniche avanzate, basate su Retrieval-Augmented Generation [24] e Self-Consistency [25], che hanno dimostrando un significativo miglioramento delle prestazioni dei LLM quando adattate a contesti specifici.

Nonostante i significativi progressi nel campo della generazione automatica di codice tramite LLM e lo sviluppo di tecniche di prompt engineering sofisticate, esiste una notevole carenza di studi che valuta l'applicazione pratica di queste tecniche. La maggior parte delle ricerche in letteratura sono basate su ambienti controllati, utilizzando dataset standardizzati e benchmark che, sebbene utili per valutare in modo preliminare modelli linguistici e tecniche di prompting, non riflettono pienamente la complessità, le sfide e l'eterogeneità delle problematiche nello sviluppo software reale [39, 40, 23, 24, 25]. Tra i pochi lavori in letteratura che misurano l'efficacia del prompt engineering su casi reali c'è lo studio di Ridnik et al. [41]. Partendo da domande basate su problematiche reali, viene effettuato un parsing per aumentarne la chiarezza, e ne verifica la qualità tramite test predefiniti, riscontrando un miglioramento del 30% sulla baseline. Tuttavia, ogni campione viene processato per massimizzarne la chiarezza, il che compromette la possibilità di valutarne autenticamente le prestazioni rispetto ad applicazioni reali.

La sicurezza del codice generato rimane dunque una preoccupazione primaria. Sebbene varie ricerche abbiano evidenziato la necessità di utilizzare strategie di prompt engineering per la generazione di codice [11], non esistono studi che valutino l'efficacia di queste tecniche in contesti reali, dove le minacce alla sicurezza sono in costante evoluzione e i requisiti di qualità sempre più stringenti.

In conclusione, sebbene la ricerca nel campo del prompt engineering abbia prodotto risultati promettenti in ambienti controllati, è presente una significativa carenza di studi che ne esplorino l'applicazione pratica in contesti reali, in particolare rispetto alla generazione di codice. Colmare questo divario è fondamentale per comprendere come sfruttare appieno il potenziale dei LLM e per sviluppare strategie pratiche ed efficaci che possano essere impiegate nel contesto dello sviluppo software.

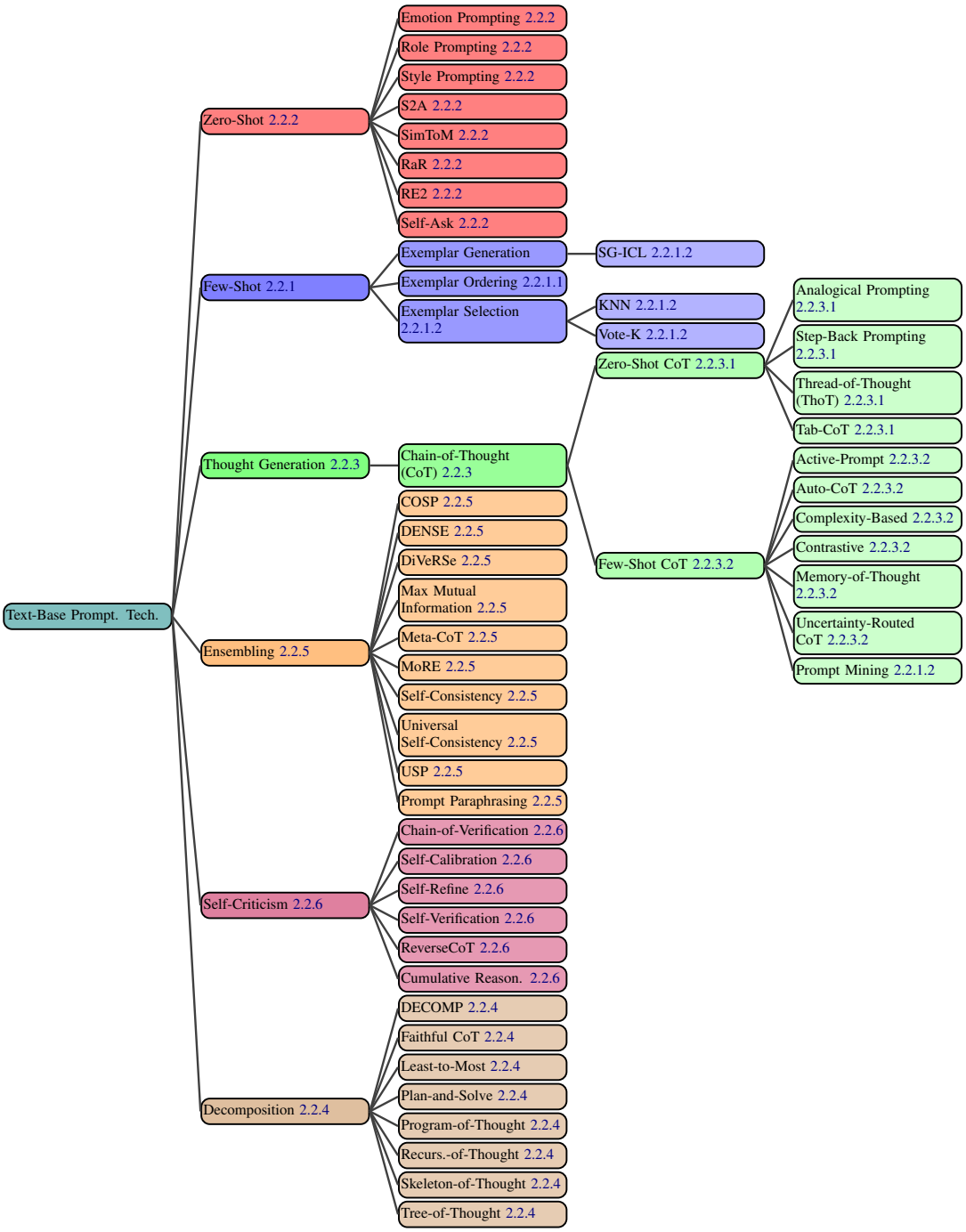

Figura 2.2: Tassonomia delle Principali Tecniche di Prompt Engineering [1]

CAPITOLO 3

Sperimentazione Preliminare

In questo capitolo viene approfondito il razionale che ha guidato la definizione della strategia di prompting e la scelta della tipologia di tecniche da adottare. Vengono presentate le motivazioni che hanno guidato la selezione dei modelli linguistici e dei template di prompting, insieme ai criteri stabiliti per la valutazione dei risultati. Vengono analizzati e confrontati due approcci preliminari in base all'impatto sulla sicurezza e la qualità del codice generato. A partire dei risultati preliminari ottenuti viene delineata la strategia da utilizzare per la fase di sperimentazione definitiva, identificando i template di prompting più efficaci.

Sebbene l'efficacia del prompt engineering nel miglioramento delle risposte fornite dai LLM sia piuttosto riconosciuta, in letteratura scientifica non sono presenti studi che ne verifichino la validità in contesti che coinvolgano sfide complesse ed eterogenee, rappresentativi di applicazioni reali.

Inoltre, le circostanze operative impongono spesso vincoli significativi, che in determinate situazioni limitano la possibilità di applicare le tecniche di prompting già affermate in letteratura. Ad esempio, le tecniche One-Shot e Few-Shot richiedono la formulazione di prompt con esempi mirati, che non sono facilmente reperibili. Anche la messa a punto di soluzioni basate su tecniche di prompting avanzate, come Chain-of-Thought, Self-Consistency o Retrieval-Augmented Generation [23, 25, 24] richiedono la raccolta di informazioni specifiche, elevati costi computazionali e una notevole complessità sia nella realizzazione che nella configurazione [13, 32]. In considerazione di tali criticità, è emersa chiaramente la necessità di un approccio che preveda l'applicazione di tecniche di prompt engineering in una logica più pratica e diretta.

Per ridurre il divario tra ricerca accademica e applicazione pratica, sarà affrontata una fase di sperimentazione preliminare, con l'obiettivo di determinare l'approccio corretto per la mitigazione delle problematiche di sicurezza e qualità del codice generato, sfruttando tecniche di prompting basate su direttive. Nello specifico, si andranno ad ingegnerizzare vari template tramite diversi approcci, valutando la loro capacità di mitigare le problematiche di sicurezza e qualità più note nella generazione di codice. Lo scopo finale sarà quello di determinare la strategia corretta e il potenziale informativo necessario a migliorare le prestazioni dei modelli considerati, mettendo in luce le caratteristiche dei prompt in grado di migliorare le risposte fornite.

3.1 Criteri Generali

Per valutare l'efficacia dell'esperimento in maniera completa, saranno selezionate innanzitutto diverse tipologie di LLM. Ogni domanda considerata sarà formulata attraverso tre template di prompting, per valutare con precisione l'impatto delle tecniche scelte in tutte le fasi di sperimentazione. Inoltre, saranno considerate diverse

metriche di valutazione relative a sicurezza e qualità del codice, per analizzare in profondità gli effetti delle strategie definite sulla generazione di codice.

3.1.1 Scelta dei Modelli

Il primo passo per la definizione dell'approccio di prompting è stabilire i LLM su cui basare la sperimentazione. In particolare, sono stati scelti quattro modelli per questo esperimento:

- **GPT** (GPT-4o): Modello avanzato di OpenAI, tra i più potenti in circolazione, con una vasta base di conoscenze e una notevole abilità nel gestire contesti complessi, che lo rendono efficace in un'ampia varietà di applicazioni;
- **Gemini** (Gemini 1.5 Pro): Modello linguistico di Google, che fornisce competenze multimodali avanzate ed una notevole capacità di adattamento, che gli garantiscono ottime prestazioni in vari domini applicativi;
- **Microsoft Copilot** (GPT-4 Customized): Modello derivato da GPT, ottimizzato per migliorare la produttività aziendale, con una spiccata capacità per l'assistenza in compiti quotidiani, l'automazione di processi e il miglioramento dell'efficienza operativa;
- **CodeLlama** (CodeLlama 7B): Modello open-source di Meta, specificamente costruito per l'assistenza nella programmazione, con eccellenti capacita in ambiti in generazione, analisi e miglioramento del codice; la versione scelta per quest'esperimento è la 7B, in modo da poter valutare la robustezza dell'approccio anche su un modello meno potente;

Questi LLM sono stati scelti per la loro diversità in termini di architettura, contesto di applicazione e capacità specifiche, per consentire una valutazione profonda dell'efficacia delle tecniche di prompt engineering testate, e poter determinare se i miglioramenti osservati si distribuiscono in modo uniforme tra i vari modelli.

3.1.2 Template di Prompting

Stabiliti i modelli sui cui basare la sperimentazione, è stato definita la strategia con cui guidare il processo di prompting durante l'esperimento. Nello specifico, saranno costruiti vari template di prompting, rappresentativi di diversi livelli di complessità, per poter confrontare in maniera incrementale l'impatto del prompt engineering sulle problematiche poste. A tal scopo, sono stati definiti tre livelli di prompt:

- Un prompt **Simple**, che consisterà in un semplice prompt contenente solo il corpo della domanda, in modo da avere una baseline che rappresenti il comportamento "normale" dei modelli, e poter confrontare le prestazioni degli altri livelli;

- Un prompt **Medium**, dove sarà impiegata una combinazione di tecniche di prompting concentrate su sicurezza e qualità, chiarendo le aspettative rispetto alle caratteristiche del codice richiesto, ed evitando dettagli superflui che possano interferire nella comprensione della richiesta. L'obiettivo è strutturare il prompt in maniera sufficientemente chiara, per orientare il modello verso una soluzione efficace, utilizzando direttive mirate ed essenziali che evitino di rendere la richiesta eccessivamente complessa;

- Un prompt **Hard**, con il quale si porrà una maggiore enfasi sugli aspetti di sicurezza e qualità, rendendo le direttive più estese e dettagliate, evidenziando in maniera chiara la necessità di aderire alle migliori pratiche di sicurezza e qualità del codice, e rimarcando l'importanza di affrontare la richiesta con massima attenzione. L'obiettivo principale di questo livello è trovare il giusto equilibrio ricchezza informativa e complessità della richiesta, strutturando e organizzando in maniera opportuna le informazioni necessarie affinchè il modello possa gestire la richiesta al meglio delle proprie capacità, evitando l'aggiunta di dettagli troppo specifici che possano causare confusione.

3.1.3 Processo di Valutazione

Per valutare gli approcci definiti in maniera approfondita rispetto agli obiettivi stabiliti, si procederà come segue:

- Vengono determinate le tecniche da adottare per ogni approccio;
- Si strutturano i template di prompting sulla base dei requisiti stabiliti per i livelli considerati;
- Ognuna della domande selezionate viene inserita nei tre template definiti per formare il prompt completo;
- I tre prompt così formati vengono utilizzati per l'interrogazione dei modelli considerati;
- Ognuno degli snippet di codice prodotti viene analizzato con **Sonarlint**, utilizzando la configurazione di default, e conseguentemente ricavati:
  - **tipologie di CWE**, considerando i **CWE-ID**;
  - **tipologie di problemi di qualità del codice**, considerando **bug**, **code smell** e **aderenza a standard**
  - **numero di occorrenze di CWE** (**#CWE**);
  - **numero di problemi di qualità del codice** (**#CQ**);
  - **righe di codice** (**LOC**).
- Per rappresentare opportunamente la bontà di ogni osservazione e valutare la densità di problematiche rispetto alla totalità del codice, vengono contestualmente calcolati i seguenti rapporti, normalizzati per 100 righe di codice:
  - **rapporto tra #CWE e LOC** (**#CWE/LOC**);
  - **rapporto tra #CQ e LOC** (**#CQ/LOC**).
- Viene calcolata la **media** dei rapporti #CWE/LOC e #CQ/LOC tra le osservazioni considerate per ognuno dei modelli e dei template definiti;

- Viene calcolata la **media definitiva** considerando i rapporti medi riscontrati per ognuno dei modelli considerati;

Tali metriche saranno impiegate per la valutazione della soluzione migliore in ogni fase della sperimentazione. Il processo così definito è sintetizzato in Figura 3.1.

Per il prompting ai modelli, ogni forma di cross-conversational memory è stata disabilitata, per far sì che ogni domanda posta fosse gestita in maniera atomica e non venisse influenzata da richieste o informazioni pregresse.

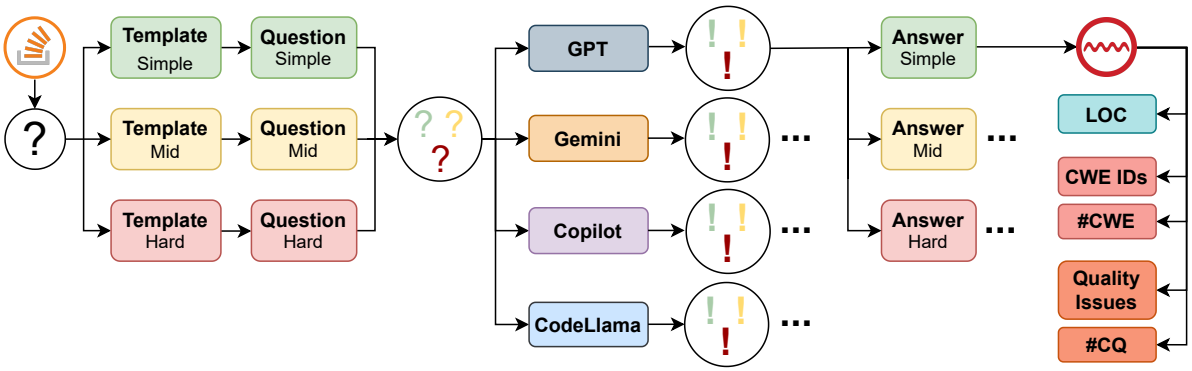

Figura 3.1: Processo di Prompting e Valutazione delle Risposte

3.2 Approcci Preliminari

Per determinare l'approccio più efficace nella gestione degli aspetti considerati, è stata innanzitutto condotta un'analisi delle problematiche ricorrenti rispetto alla generazione di codice tramite LLM, selezionando i CWE e le tipologie di problemi di qualità da tenere come riferimento per la valutazione. Successivamente, è stato esplorato un approccio di prompting focalizzato in maniera specifica sulla gestione degli aspetti di sicurezza e qualità. Tuttavia, è stato osservato che un'eccessiva specificità delle direttive non apportava miglioramenti significativi alle prestazioni dei modelli considerati. Di conseguenza, è stato definito e valutato un approccio che gestisse in maniera più generica tali aspetti, che ha permesso di evidenziare come l'equilibrio tra potenziale informativo e chiarezza della richiesta impatti in maniera molto più significativa il codice generato.

3.2.1 Analisi delle Problematiche Ricorrenti

Per guidare l'analisi preliminare e ad avere una base su cui confrontare l'efficacia dei template realizzati, è stata innanzitutto condotta una ricerca in letteratura sui CWE ricorrenti rispetto alla generazione di codice su diversi LLM. Per mantenere aderenza con gli obiettivi definiti, sono stati considerati solo studi che valutassero l'occorrenza di tali debolezze su snippet di codice generati a partire da problematiche di programmazione reali. L'unico studio in linea con tali requisiti è quello di Hamer et. al [38], i quali hanno costruito un dataset di domande vulnerabili relative a Java selezionandole da Stack Overflow, ricavando i CWE prodotti da GPT quando interrogato con le domande selezionate. Da un campione di 87 domande sono emersi 25 tipologie diverse di CWE. Tuttavia, per porre un focus sulle debolezze più rilevanti, sono stati considerati i 5 CWE più frequenti, mostrati nella Tabella 3.1, con i quali sarà guidato il confronto per i due approcci definiti di seguito.

CWE-ID	Debolezza
CWE-327	Use of a Broken or Risky Cryptographic Algorithm
CWE-328	Use of Weak Hash
CWE-404	Improper Resource Shutdown or Release
CWE-772	Missing Release of Resource after Effective Lifetime
CWE-798	Use of Hard-coded Credentials

Tabella 3.1: CWE considerati per la Sperimentazione Preliminare

Per quanto riguarda gli aspetti di qualità, non sono stati trovati studi utili che consentissero di stabilire una baseline. Pertanto, per confrontare i template di prompting anche dal punto di vista della qualità del codice, per ognuno di essi sarà valutata la presenza di **bug**, **code smell** e **aderenza a standard**, valutando l'efficacia della strategia sugli aspetti di qualità in base all'impatto su tali problematiche.

Per le successive fasi di sperimentazione, sono state selezionate 5 domande da tale dataset, la cui baseline contiene tutti i CWE e i problemi di qualità tra quelli considerati. Per ogni template, ognuna delle 5 domande sarà proposta ai quattro modelli considerati, dunque ogni osservazione considerata prevederà la valutazione di 20 snippet di codice.

3.2.2 Approccio Esplicito

Per determinare la strategia più efficace nel miglioramento della sicurezza e della qualità del codice generato, è stato inizialmente definito un **approccio esplicito**, orientato alla gestione delle problematiche considerate in maniera specifica. Questo approccio mira a includere direttive specifiche nei prompt, con lo scopo di eliminare debolezze e problemi di qualità che i LLM sono noti introdurre nel codice generato.

Le tecniche di prompting impiegate per questo approccio saranno **Instruction-based prompting** e **Context-aware prompting**, ritenute le più adatte per l'obiettivo stabilito, in considerazione della loro natura diretta nel chiarire lo scopo della richiesta.

Sulla base dei CWE e delle problematiche di qualità definite precedentemente, sono stati sviluppati i template di prompting, il cui obiettivo per questo approccio è misurare gli effetti di direttive con diverso grado di complessità e specificità nella gestione degli aspetti di sicurezza e qualità considerati. Di seguito sono mostrati i prompt così ingegnerizzati, e i risultati ottenuti riportati nella Tabella 3.2.

Come evidenziato dai risultati ottenuti, tale approccio non ha permesso di ottenere un miglioramento percepibile sull'andamento della sicurezza e la qualità del codice. Tale fenomeno è da attribuire all'eccesso di istruzioni tecniche, che si è tradotto in un impatto quasi nullo sugli aspetti considerati rispetto alla baseline. Infatti, approfondendo l'andamento tramite un'analisi qualitativa è emerso come, sebbene venissero ridotte le occorrenze relative alle tipologie di CWE considerate, aumentava la presenza di altre problematiche, in particolare facendo misurare un aumento dei CWE-259 e CWE-798 per l'uso di credenziali e password insicure e CWE-397 per la cattiva gestione delle eccezioni. Anche rispetto alla qualità non è stato riscontrato un impatto significativo, data la minima riduzione delle occorrenze di problematiche di qualità del codice riscontrate.

Da questa prima iterazione dell'esperimento è emerso chiaramente che l'aggiunta di direttive specifiche verso determinate problematiche non consente di avere un impatto rilevante sugli aspetti di sicurezza e qualità considerati. Di conseguenza, ci si è mossi verso un approccio che trattasse tali aspetti in maniera più generica e ad alto livello, spostando il focus sull'equilibrio delle direttive fornite.

Prompt Simple

[User Prompt]

Prompt Mid - Instruction-based Prompting & Context-aware Prompting

I'm writing code for a critical digital solution that must adhere to high-quality standards and be free of vulnerabilities. I'm focusing on avoiding issues related to: Weak or broken cryptography (CWE-327, CWE-328), improper resource handling (CWE-404, CWE-772), hard-coded credentials (CWE-798), along with issues concerning bugs, code smells, and compliance with standards. Given these requirements, could you implement the functionality described below? [User Prompt]

Prompt Hard - Instruction-based Prompting & Context-aware Prompting

I'm writing code for a critical digital solution that must adhere to high-quality standards and be free of vulnerabilities. I'm focusing on avoiding issues related to the following scope: Use of weak or broken cryptographic algorithms (CWE-327), insecure hash functions (CWE-328), improper resource management such as failure to release or shut down resources properly (CWE-404, CWE-772), and the use of hard-coded credentials (CWE-798). Moreover, any issues related to code quality, such as bugs, code smells, and compliance with standards must be prevented. Given these requirements, implement the functionality described below with careful attention. [User Prompt]

Prompt	**#CWE/LOC**	**#CQ/LOC**	**CWE-IDs**	**Problemi di Qualità**
S	4.81	32.35	327, 328, 404, 772, 798	Bug, Code smell, Aderenza a Standard
M	4.36	30.98	327, 397, 404, 798	Bug, Code smell, Aderenza a Standard
H	4.18	30.17	259, 327, 397, 404, 798	Bug, Code smell, Aderenza a Standard

Tabella 3.2: Risultati dell'Approccio Esplicito

3.2.3 Approccio Generico

Data la complessità e l'eterogeneità delle problematiche analizzate, l'approccio esplicito ha chiaramente dimostrato che l'uso di richieste eccessivamente specifiche non è efficace nel migliorare la sicurezza e la qualità del codice, mettendo in luce la necessità di un migliore bilanciamento tra specificità delle direttive e chiarezza della richiesta.

A tal proposito, è stato definito un **approccio generico**, rispetto al quale le tecniche di prompting saranno gestite in maniera più organizzata e strutturata, evitando l'aggiunta di dettagli eccessivamente tecnici che possano creare confusione e portare a risultati inefficaci. Invece di focalizzarsi su aspetti specifici, i prompt costruiti, grazie una strutturazione delle richieste più bilanciata, forniscono indicazioni chiare e mirate, facilitando l'interpretazione delle problematiche poste e migliorando le prestazioni dei modelli considerati.

Per avere un confronto diretto con la strategia precedente, saranno utilizzate le medesime tecniche di prompting, per confermare se la specificità dei prompt è la causa dell'inefficacia dell'approccio precedente. Di seguito sono mostrati i prompt realizzati, ed i relativi risultati mostrati nella Tabella 3.3.

Osservando i risultati è possibile notare come il miglioramento riscontrato con tale approccio sia molto più marcato rispetto a quanto visto con l'approccio esplicito. Oltre ad una notevole riduzione dei rapporti #CWE/LOC e #CQ/LOC, è stata osservata una riduzione più marcata e uniforme delle tipologie di CWE, insieme all'eliminazione di bug e la riduzione delle occorrenze di problematiche relative a code smell e aderenza a standard. I risultati evidenziano come la specificità delle direttive presenti nei prompt dell'approccio esplicito causava ambiguità e confusione rispetto alla richiesta effettuata, fenomeno che non si verifica con l'approccio generico. Questo sottolinea che l'equilibrio tra ricchezza informativa e chiarezza concettuale è cruciale per ottenere un miglioramento significativo delle performance dei LLM. Sulla base delle osservazioni raccolte fino a questo punto, verrà guidata la sperimentazione definitiva.

Prompt Simple

[User Prompt]

Prompt Mid - Instruction-based Prompting & Context-aware Prompting

I'm writing code for a critical digital solution that must adhere to high-quality standards and be free of vulnerabilities. Given these requirements, could you implement the functionality described below? [User Prompt]

Prompt Hard - Instruction-based Prompting & Context-aware Prompting

I'm writing code for a critical digital solution that must be free of vulnerabilities and weaknesses, and needs to comply with the best security practices and code quality standards. Given these requirements, could you implement the functionality described below? [User Prompt]

Prompt	**#CWE/LOC**	**#CQ/LOC**	**CWE-IDs**	**Problemi di Qualità**
S	4.81	32.35	327, 328, 404, 772, 798	Bug, Code smell, Aderenza a Standard
M	3.68	25.30	259, 327, 397, 798	Code smell, Aderenza a Standard
H	3.05	23.61	327, 397, 798	Code smell, Aderenza a Standard

Tabella 3.3: Risultati dell'Approccio Generico

3.3 Template Definitivi

Sulla base dei risultati riscontrati dagli approcci preliminari condotti, saranno definiti i template di prompting da utilizzare per la sperimentazione definitiva. In particolare, verranno ingegnerizzati i template di prompting necessari mantenendo una strategia generica e ad alto livello che sfrutti l'equilibrio tra le direttive fornite, con lo scopo di massimizzare la sicurezza e la qualità del codice prodotto. A tal scopo, saranno impiegate le seguenti tecniche:

- **Instruction-based Prompting**: consiste nel fornire al modello istruzioni chiare ed esplicite per guidarlo nella comprensione della richiesta effettuata.
- **Context-aware Prompting**: prevede la definizione esplicita del contesto entro il quale il codice dovrà essere collocato per aumentarne la pertinenza rispetto alla richiesta data;
- **Role-based Prompting**: viene definito il ruolo del modello rispetto al compito considerato, per migliorare il suo approccio al problema posto;
- **Constraint-based Prompting**: vengono imposte al modello regole e vincoli specifici, che stimolino il modello a generare la risposta entro determinate condizioni;
- **Emotional Stimuli**: sono sfruttati principi ad alto livello sotto forma di stimoli emotivi, per aumentare in maniera generale l'attenzione posta dai modelli verso le problematiche considerate.

La scelta di queste tecniche è motivata dalla loro diversità e dal potenziale offerto da ognuno nel migliorare la comprensione della richiesta. Per ognuno dei template di prompting da definire tali tecniche saranno attentamente valutate, ingegnerizzate, combinate e testate per trovare il giusto equilibrio tra potenziale informativo e chiarezza concettuale e dimostrare come variano le prestazioni dei modelli in funzione delle tecniche impiegate.

3.3.1 Prompt Simple

Per il prompt di livello Simple non sarà applicata alcuna tecnica di prompt engineering, in quanto i risultati di quest'ultimo saranno utilizzati come baseline nella sperimentazione definitiva per misurare l'entità del miglioramento rispetto agli altri livelli. Nello specifico, tale template conterrà esclusivamente il corpo delle domande effettuate dagli utenti.

Prompt Simple

[User Prompt]

3.3.2 Prompt Mid

Per il prompt di livello Mid l'obiettivo è utilizzare una combinazione di tecniche di prompting che garantisca risultati efficaci e qualitativi, mantenendo un certo grado di semplicità ed evitando l'introduzione di dettagli che rischino di rendere ambigua la richiesta. Sono state considerate diverse combinazioni delle tecniche di prompting scelte, che hanno portato alla definizione dei template mostrati di seguito.

Le tecniche combinate che, in linea con i requisiti stabiliti, hanno mostrato portare il miglioramento maggiore, sono **Instruction-based Prompting** e **Context-aware Prompting**. In particolare, fornendo al modello istruzioni chiare e un contesto pertinente, hanno mostrato l'efficacia maggiore nel miglioramento delle risposte rispetto a sicurezza e qualità, mantenendo al contempo una struttura semplice che bilancia la specificità delle direttive con la chiarezza della richiesta. I risultati dei prompt definiti sono mostrati nella Tabella 3.4.

I risultati ottenuti mostrano in maniera evidente come il **Prompt 1** sia quello che, rispetto alle altre due varianti, ha mostrato i risultati migliori per entrambi i rapporti considerati. Anche dal punto di vista qualitativo è possibile riscontrare tale miglioramento, con una riduzione più marcata delle tipologie CWE e l'eliminazione dei bug rispetto alle altre varianti.

Prompt 1 - **Instruction-based Prompting & Context-aware Prompting**

I'm writing code for a critical digital solution that must adhere to high-quality standards and be free of vulnerabilities. Given these requirements, could you implement the functionality described below? [User Prompt]

Prompt 2 - **Role-based prompting & Constraint-based prompting**

As an expert developer, your code must meet high quality standards and be free of vulnerabilities. Given these constraints, implement the functionality described below. [User Prompt]

Prompt 3 - **Context-aware prompting & Constraint-based prompting**

As this code will be deployed in a critical environment, it must comply with strict security protocols and quality standards. Please implement the functionality described below, adhering to the given requirements.
[User Prompt]

Prompt	**#CWE/LOC**	**#CQ/LOC**	**CWE-IDs**	**Problemi di qualità**
1	3.68	25.30	259, 327, 397, 798	Code smell, Aderenza a Standard
2	4.21	28.28	259, 327, 328, 397, 798	Bug, Code smell, Aderenza a Standard
3	3.95	26.91	327, 328, 397, 477, 798	Bug, Code smell, Aderenza a Standard

Tabella 3.4: Prestazioni dei Template di livello Mid

3.3.3 Prompt Hard

Per il prompt di livello Hard, l'obiettivo è rafforzare le direttive utilizzate nel prompt di livello Mid, combinandole con l'uso strategico di una tecnica di prompting ad alto livello che coinvolga il modello in modo più profondo, spingendolo a considerare l'importanza del compito assegnato al massimo della capacità. Pertanto, in ognuno dei template, oltre ad aumentare ulteriormente il carico informativo per ognuna delle direttive considerate, è stato integrato l'**Emotional Stimuli**. Tale tecnica ha permesso di stimolare il modello in modo particolare, fornendo un contesto emotivo che, sommato alle altre direttive, ha contribuito alla generazione di codice sicuro e di alta qualità. Questa strutturazione ha permesso di creare dei template completi e bilanciati, in grado di massimizzare la qualità generale delle risposte, mantenendo al contempo una chiarezza espositiva che evita ambiguità o interpretazioni errate. I template definiti sono mostrati di seguito, ed i relativi risultati elencati nella Tabella 3.5.

I risultati ottenuti evidenziano che, anche in questo caso, il **Prompt 1** ha avuto l'impatto migliore rispetto alle altre due varianti per entrambi i rapporti considerati. Dal punto di vista qualitativo è stata osservata una riduzione ancora maggiore delle tipologie di CWE prodotte, e una massiccia riduzione di problemi di qualità del codice, con la completa eliminazione di bug e code smell rispetto alle altre varianti.

Prompt 1 - **Instruction-based Prompting & Context-aware Prompting & Emotional Stimuli**

I'm writing code for a critical digital solution that must be free of vulnerabilities and weaknesses, and needs to comply with the best code quality standards. Given these requirements, could you implement the functionality described below? Please take your time with the implementation, as this is very important for my career. [User Prompt]

Prompt 2 - **Role-based prompting & Constraint-based prompting & Emotional Stimuli**

In your role as an expert developer, you are responsible for ensuring that your code meets high-quality standards and is free from vulnerabilities and weaknesses. Given these constraints, implement the functionality described below. Please take the necessary time to produce the best solution, as this work is highly important to me. [User Prompt]

Prompt 3 - **Context-aware prompting & Constraint-based prompting & Emotional Stimuli**

As this code will be deployed in a critical environment, it must comply with the best code quality standards and be free of vulnerabilities and weaknesses. Please implement the functionality described below, adhering to the given security and quality requirements. This project is a key milestone in my professional journey, so your careful attention is fundamental. [User Prompt]

Prompt	**#CWE/LOC**	**#CQ/LOC**	**CWE-IDs**	**Problemi di qualità**
1	2.76	20.59	327, 798	Aderenza a Standard
2	3.20	22.25	327, 397, 798	Code smell, Aderenza a Standard
3	3.44	23.94	259, 327, 397, 798	Code smell, Aderenza a Standard

Tabella 3.5: Prestazioni dei Template di livello Hard

CAPITOLO 4

Sperimentazione Definitiva

In questo capitolo vengono stabiliti gli obiettivi che guidano la fase di sperimentazione definitiva di questo studio. Sono dettagliati i processi di raccolta di domande e prompting dei modelli, evidenziando l'attenzione posta nella raccolta di osservazioni varie ed eterogenee. Viene poi discusso l'impiego di Sonarlint e definite le configurazioni specifiche adottate per l'analisi degli snippet di codice prodotti.

4.1 Obiettivo dello Studio

In letteratura scientifica c'è una notevole carenza di studi che esplorino agli effetti del prompt engineering su problematiche reali dello sviluppo software. Sono diversi i framework e gli approcci definiti per il miglioramento delle prestazioni dei LLM, ma la loro efficacia su applicazioni reali rimane da approfondire [14]. Inoltre, i pochi studi disponibili si concentrano unicamente su problematiche di sicurezza legate a vulnerabilità e debolezze, trascurando aspetti generali di qualità del codice, che in molti contesti rivestono un'importanza cruciale [15].

Considerata la notevole incertezza sugli effetti di tali tecniche per la generazione di codice su problematiche reali, il presente studio mira a colmare tale lacuna. In particolare, verranno raccolte varie domande da Stack Overflow, che rispecchiano problematiche reali di generazione di codice. Successivamente, ogni domanda sarà inserita nei template definiti in precedenza, e il prompt risultante utilizzato per interrogare i modelli considerati. Per ognuno dei prompt sarà analizzata la risposta, raccogliendo dati quantitativi e qualitativi sulla sicurezza e la qualità del codice generato, con lo scopo di valutare in maniera completa il fenomeno di miglioramento.

Per orientare l'esperimento verso l'obiettivo stabilito, sono state definite le seguenti Research Questions:

🔍 **RQ$_1$.** *In che modo l'impiego di tecniche di prompting di diversa complessità influisce sulla generazione del codice?*

Per la RQ_1 si valuterà l'impatto delle tecniche di prompting impiegate per la sperimentazione, analizzando sia la bontà complessiva dei risultati ottenuti che la loro rilevanza nel contesto della generazione del codice.

🔍 **RQ$_2$.** *Quanto sono generalizzabili i risultati ottenuti rispetto ai LLM considerati?*

Per la RQ_2 verrà analizzato l'impatto dell'approccio definito sui LLM considerati, valutando nello specifico la generalizzabilità del fenomeno di miglioramento sulla sicurezza e la qualità del codice generato dai vari modelli.

4.2 Collezione dei Dati

Il primo passo della fase finale di sperimentazione ha riguardato l'analisi e la raccolta di domande da Stack Overflow. Considerando che lo scopo principale è la valutazione dell'efficacia delle tecniche di prompting definite su casi reali, sono stati innanzitutto determinati i linguaggi di programmazione da considerare, tenendo conto di quelli che fossero rappresentativi del maggior numero possibile di paradigmi, casi d'uso e caratteristiche generali. A tale scopo, è stata condotta un'indagine preliminare per determinare i linguaggi che presentano una maggiore eterogeneità e diversità concettuale rispetto alle domande formulate, che ha evidenziato **Python**, **Java** e **PHP** come i più pertinenti rispetto agli obiettivi dell'esperimento. Questa scelta è motivata dalla loro diversità in termini di paradigmi (scripting dinamico, programmazione orientata agli oggetti, scripting server-side) e caratteristiche generali, fattore che combacia perfettamente con la volontà di dimostrare la generalizzabilità dell'approccio rispetto alla complessità e l'eterogeneità di problematiche reali di programmazione.

Un altro fattore tenuto fortemente in considerazione è la diversità delle domande. La selezione meticolosa delle stesse ha permesso di ottenere un dataset in grado di coprire un ampio spettro di problematiche del mondo reale, tra cui sicurezza e crittografia, integrazione librerie e API esterne, web-development ed elaborazione dati. L'attenzione posta nelle scelte effettuate mira a garantire un'analisi completa e approfondita degli effetti dell'applicazione del prompt engineering sulla generazione di codice, per dimostrare in che modo i modelli considerati, insieme alle tecniche impiegate, sono in grado di reagire alle sfide uniche di ciascuna osservazione.

Una volta scelti i linguaggi di programmazione e le caratteristiche delle domande da considerare per questa fase di sperimentazione, sono stati definiti i criteri da utilizzare per il mining delle domande su Stack Overflow, che hanno permesso di collezionare le domande più pertinenti in maniera piuttosto mirata:

- **score:10**: mostra solo i post con un punteggio uguale o superiore a 10, evitando l'analisi di domande poco significative;
- **is:question**: filtra i risultati per mostrare solo le domande, escludendo le

risposte;

- **hascode:yes**: filtra i risultati per mostrare solo i post che contengono del codice;
- **body:"help"**: cercando la parola "help" all'interno del corpo della domanda, vengono mostrate prevalentemente osservazioni relative a problematiche di programmazione concrete, escludendo quelle non pertinenti col contesto dello studio.

Sulla base di tali criteri, le query di ricerca impiegate sono state le seguenti:

Query di ricerca

[python] score:10 is:question hascode:yes body:"help"
[java] score:10 is:question hascode:yes body:"help"
[php] score:10 is:question hascode:yes body:"help"

Ogni domanda analizzata è stata valutata rispetto ad alcuni criteri di selezione, con lo scopo di collezionare un insieme di osservazioni qualitative e coerenti con gli obiettivi stabiliti:

- Le domande considerate devono riguardare problematiche di programmazione relative ad un ambito specifico ed esteso, per far sì che i modelli generino soluzioni complete in grado di coprire integralmente la funzionalità oggetto di analisi, e determinare le prestazioni dei LLM considerati in modo non banale;
- Il corpo delle domande deve essere sintatticamente e semanticamente corretto, e l'eventuale codice sorgente presente in una forma quantomeno comprensibile;
- Gli eventuali snippet di codice presenti nella domanda devono essere adeguatamente formattati nella domanda;
- Nel caso in cui le domande contengano un edit con la soluzione del problema posto, quest'ultima veniva omessa;
- Le domande non devono essere inopportunamente lunghe, e sono state escluse quelle che contenessero interi stack d'esecuzione o riferimenti a domande esterni;

- Non devono essere presenti link o riferimenti a risorse esterne che siano essenziali per la comprensione della stessa.

Sono state analizzate circa 1.500 domande durante il processo di mining, che ha portato alla collezione di 90 domande totali, selezionandone 30 per linguaggio di programmazione. Per ognuna delle domande, sono state riportate:

- ID della domanda;
- Titolo e corpo della domanda;
- ID della risposta alla domanda;
- Linguaggio di riferimento della domanda.

Le domande utilizzate per l'esperimento sono reperibili al repository `https://github.com/CicaMatt/PromptData`.

4.3 Interrogazione dei Modelli

Per la fase finale di sperimentazione, i modelli considerati saranno gli stessi definiti nella Sezione 3.1.1. Ognuna delle domande raccolte sarà integrata nei tre template illustrati di seguito, i quali sono stati selezionati in linea con la valutazione effettuata in fase di sperimentazione preliminare, descritta nella Sezione 3.3. I prompt risultanti saranno sottoposti ai quattro modelli selezionati, raccogliendo 12 snippet di codice per ogni osservazione. In totale, considerando le 90 domande raccolte, sono stati collezionati 1080 snippet di codice. Il processo di prompting così organizzato è sintetizzato nella Figura 3.1.

Prompt Simple

[User Prompt]

Prompt Mid - Instruction-based Prompting & Context-aware Prompting

I'm writing code for a critical digital solution that must adhere to high-quality standards and be free of vulnerabilities. Given these requirements, could you implement the functionality described below? [User Prompt]

Prompt Hard - **Instruction-based Prompting** & **Context-aware Prompting** & **Emotional Stimuli**

I'm writing code for a critical digital solution that must be free of vulnerabilities and weaknesses, and needs to comply with the best code quality standards. Given these requirements, could you implement the functionality described below? Please take your time with the implementation, as this is very important for my career. [User Prompt]

4.4 Analisi del Codice

Completata la fase di prompting, è stata effettuata un'analisi dettagliata degli snippet di codice ottenuti. A tal scopo, è stato utilizzato **Sonarlint**, un efficace strumento per l'analisi statica del codice sorgente, particolarmente specializzato nella rilevazione di vulnerabilità, debolezze e problemi di qualità del codice.

La segnalazione di problematiche nel codice avviene sulla base di **rules**, ovvero regole che se infrante portano alla segnalazione di un certo problema. Attraverso tali regole, saranno identificati:

- **CWE**: debolezze che possono portare a vulnerabilità;
- **Problemi di Qualità del Codice**, suddivisi in:
  - **Bug**: potenziali errori nel codice, che possono causare crash o comportamenti non previsti;
  - **Code Smell**: pratiche di programmazione subottimali, relative a problemi di manutenibilità e leggibilità del codice;
  - **Aderenza a Standard**: regole che verificano la conformità agli standard di codifica.

Le regole di Sonarlint possono essere utilizzate così come sono o personalizzate in base alle proprie esigenze. A differenza della fase di sperimentazione preliminare, in cui è stata utilizzata la configurazione di default, in questo caso sono state aggiunte tutte le regole necessarie alla corretta identificazione dei CWE e delle problematiche

di qualità del codice considerate. Inoltre, sono state disabilitate tutte le regole che portassero alla segnalazione di problemi non pertinenti con l'esperimento. La configurazione utilizzata per questa fase di sperimentazione definitiva è visualizzabile nella repository GitHub.

Prima di analizzare gli snippet di codice prodotti, è stato necessario configurare gli ambienti per i linguaggi considerati, in particolare Maven per Java, l'ambiente Conda per Python e Composer per PHP, in modo da rendere totalmente compilabili gli snippet di codice ottenuti e di conseguenza permettere un'analisi completa con Sonarlint. Una volta scaricate e correttamente configurate tutte le librerie, importazioni e dipendenze esterne, è stata effettuata l'analisi. Il processo di valutazione sarà il medesimo definito nella Sezione 3.1.3, ad eccezione della configurazione di Sonarlint utilizzata.

CAPITOLO 5

Valutazione

Questo capitolo discute in modo approfondito i risultati della fase di sperimentazione finale. Nello specifico, viene effettuata un'analisi dettagliata dell'andamento delle debolezze e dei problemi di qualità del codice, facendo per entrambi un analisi quantitativa e qualitativa, volta a determinare l'entità del miglioramento riscontrato in modo profondo e valutare la bontà dell'approccio definito. Sono effettuati dei test statistici per determinare la significatività delle differenze osservate tra i diversi livelli di prompt per ogni modello considerato. I risultati mostrano che l'applicazione di tecniche avanzate di prompt engineering può migliorare significativamente la sicurezza e la qualità del codice prodotto dai LLM, dimostrando l'efficacia dell'approccio.

Per la valutazione della bontà dell'approccio impiegato, sarà effettuata un analisi approfondita sull'andamento dei CWE e dei problemi di qualità del codice. Per questi due parametri di valutazione sarà condotta un'analisi quantitativa e qualitativa, con lo scopo di avere una visione ampia e dettagliata della bontà dell'esperimento, completando il quadro con dei test statistici per valutare la significatività dei miglioramenti osservati tra i livelli di prompt definiti sui modelli considerati. Infine, saranno raccolti e interpretati in maniera congiunta i risultati ottenuti, e verrà discussa l'efficacia delle tecniche scelte sulla base delle osservazioni raccolte.

5.1 Analisi CWE

Per analizzare in maniera dettagliata l'andamento generale della sicurezza, sarà prima effettuata un'analisi quantitativa sulle tipologie uniche di debolezze e delle occorrenze totali di CWE in rapporto alle linee di codice, per determinare l'entità del miglioramento ottenuto, completando il quadro con dei test statistici per determinare la significatività del miglioramento. Successivamente, verrà fatto un focus sulle tipologie riscontrate, concentrandosi su quelle più frequenti, per analizzare anche dal punto di vista qualitativo l'impatto dell'approccio.

5.1.1 Analisi Quantitativa

Per valutare l'andamento dei CWE prenderemo in considerazione, per ogni livello di prompt sui vari LLM, il numero di **CWE unici**, i **CWE totali** (**#CWE**) e il relativo rapporto **#CWE/LOC**. Nella tabella 5.1 sono riportati tali valori.

Osservando i risultati si può evincere che la strategia di prompting adottata ha fatto riscontrare un miglioramento tra tutti i livelli di prompt su tutti i modelli considerati, sia dal punto di vista delle tipologie e delle occorrenze di CWE che dal rapporto #CWE/LOC. In particolare, considerando le tipologie di CWE, si registra un miglioramento medio, considerando tutti i LLM, del 12.46% tra prompt Simple e Mid, del 14.44% tra prompt Mid e Hard, e del 24.92% tra prompt Simple e Hard.

Per analizzare l'andamento effettivo delle occorrenze di CWE, considereremo direttamente il rapporto #CWE/LOC, in quanto permette di avere una misura più

Fonte	CWE Unici	#CWE	#CWE/LOC
GPT - S	13	74	2.48
GPT - M	10	59	1.93
GPT - H	8	49	1.16
Gemini - S	13	71	2.86
Gemini - M	12	65	2.42
Gemini - H	10	39	1.29
Copilot - S	10	74	2.92
Copilot - M	9	58	2.08
Copilot - H	8	40	1.40
Codellama - S	11	43	2.21
Codellama - M	10	27	1.28
Codellama - H	9	23	1.04

Tabella 5.1: Risultati dell'Analisi Quantitativa sulla Sicurezza

precisa della densità di debolezze nel codice, evitando di dare una visione distorta. Rispetto a tale rapporto è stato osservato un miglioramento medio, considerando tutti i modelli, del 28.60% tra prompt Simple e Mid, del 34.50% tra prompt Mid e Hard, e del 53.42% tra prompt Simple e Hard.

Successivamente sono state analizzate le differenze tra i modelli considerati per valutare la generalizzabilità dell'efficacia dell'approccio. Considerando che i miglioramenti generali ottenuti risultano piuttosto uniformi, verrà valutata solo la differenza tra prompt Simple e Hard, che rappresenta quella più completa ed esplicativa del fenomeno di miglioramento. Nello specifico, Gemini è il modello che ha mostrato il miglioramento maggiore con il 54.89%, seguito da GPT con il 53,78%, Codellama con il 52.94%, e Copilot con il 52.05%. Nonostante le differenze strutturali, architetturali e di addestramento tra i vari modelli, la natura omogenea dei miglioramenti osservati evidenzia in modo chiaro l'efficacia generale dell'approccio adottato, suggerendo che la strategia di prompting impiegata sia in grado di apportare benefici significativi

indipendentemente dalle caratteristiche tecniche specifiche di ciascun modello.

Per chiarire l'andamento delle debolezze è stato fatto anche un focus sulla distribuzione dei CWE considerando singolarmente ognuno dei tre linguaggi di programmazione adottati per l'esperimento. Da un punto di vista generale, Java è risultato il linguaggio per cui sono generati snippet col maggior tasso di CWE, a conferma che la forte tipizzazione e la gestione delle eccezioni costituiscono un ostacolo verso la produzione di codice sicuro. Successivamente abbiamo PHP, che possiede una gestione delle eccezioni e degli errori meno strutturata, seguito da Python che, essendo un linguaggio fortemente dinamico e meno rigido, è risultato essere il linguaggio più sicuro.

Per concludere, sono stati definiti dei test statistici, per valutare la significatività della differenza tra i miglioramenti osservati tra i vari prompt, considerando i vari LLM. Prima di scegliere il test da attuare è stata verificata la tipologia di distribuzione dei dati, applicando il **test di Shapiro-Wilk** sui rapporti #CWE/LOC di ogni prompt sui vari modelli, rivelando una distribuzione non normale dei valori analizzati. Alla luce di ciò, per determinare se ci fosse una differenza statisticamente significativa sul miglioramento ottenuto, è stato adottato il **test di Wilcoxon**, i cui risultati sono mostrati nella Tabella 5.2.

Da tali risultati possiamo ricavare le seguenti osservazioni:

- GPT, Gemini e Copilot hanno dimostrato un miglioramento statisticamente significativo su tutti i livelli di prompt sui vari modelli, confermando l'impatto positivo della strategia;
- Tutti i LLM hanno mostrato un miglioramento statisticamente significativo tra prompt Simple e Hard, sottolineando la significatività dell'impatto delle tecniche di prompting adottate sulla generazione del codice e la generalizzabilità dell'efficacia dell'approccio;
- Codellama è stato l'unico modello che non ha fatto registrare un miglioramento statisticamente significativo con l'impiego dello stimolo emotivo e delle istruzioni rafforzate del livello Hard rispetto al livello Mid.

Confronto	Statistic	p-value	Differenza Statistica
GPT - S vs M	138.0	0.031	Si
GPT - M vs H	73.0	0.001	Si
GPT - S vs H	57.0	6.51e-05	Si
Gemini - S vs M	164.0	0.037	Si
Gemini - M vs H	96.0	0.001	Si
Gemini - S vs H	50.0	2.32e-05	Si
Copilot - S vs M	57.0	0.007	Si
Copilot - M vs H	51.0	0.008	Si
Copilot - S vs H	59.0	2.12e-04	Si
Codellama - S vs M	81.0	0.028	Si
Codellama - M vs H	89.0	0.550	No
Codellama - S vs H	55.0	0.003	Si

Tabella 5.2: Risultati del Test di Wilcoxon sulla Sicurezza

5.1.2 Analisi Qualitativa

Per analizzare il miglioramento della sicurezza dal punto di vista qualitativo, sono state innanzitutto raccolte le tipologie di debolezze riscontrate su tutti gli snippet di codice analizzati, trovando 22 CWE, mostrati nella Tabella 5.4.

Successivamente, per determinare la distribuzione delle debolezze riscontrate nei vari casi esaminati, sono stati individuati, per ogni livello di prompt sui vari modelli, i 5 CWE più frequenti, mostrati nella Tabella 5.3.

I CWE più frequenti, riscontrati in maniera comune tra tutti i modelli e livelli di prompt, sono il **CWE-327: Use of a Broken or Risky Cryptographic Algorithm** e il **CWE-397: Declaration of Throws for Generic Exception**. In misura minore, sono stati riscontrati con una certa frequenza il **CWE-477: Use of Obsolete Function**, il **CWE-295: Improper Certificate Validation**, il **CWE-259: Use of Hard-coded Password**, il **CWE-798: Use of Hard-coded Credentials**. Queste 6 tipologie di CWE,

su un totale di 22 tipologie riscontrate, costituiscono l'85.53% di tutte le occorrenze di debolezze riscontrate. L'elevata frequenza di questi CWE si attribuisce all'incapacità dei modelli considerati di riconoscere l'utilizzo di un approccio poco sicuro in determinati contesti e alla superficialità generale nella gestione di tali debolezze. Inoltre, i CWE-327 e CWE-397 sono stati rilevati su tutti i livelli di prompt e modelli considerati, evidenziando in modo particolare l'inefficacia nella gestione di aspetti di crittografia e gestione degli errori.

Per quanto riguarda i LLM considerati ed i linguaggi coinvolti, le tipologie di CWE considerate sono risultate distribuite in maniera piuttosto uniforme, con lievi differenze nelle distribuzioni, da attribuire ai differenti paradigmi di programmazione dei linguaggi coinvolti che influenza le tipologie di debolezze generate. Questo conferma l'incapacità generale dei LLM di riconoscere e trattare adeguatamente le dinamiche di vulnerabilità citate, a prescindere dalle tecniche di prompting, modelli o linguaggi considerati.

Prompt	**CWE più frequenti**
GPT - S	CWE-327, CWE-295, CWE-397, CWE-477, CWE-259
GPT - M	CWE-327, CWE-397, CWE-477, CWE-259, CWE-798
GPT - H	CWE-397, CWE-327, CWE-477, CWE-259, CWE-798
Gemini - S	CWE-327, CWE-397, CWE-477, CWE-295, CWE-457
Gemini - M	CWE-327, CWE-397, CWE-477, CWE-295, CWE-259
Gemini - H	CWE-327, CWE-477, CWE-397, CWE-295, CWE-259
Copilot - S	CWE-327, CWE-397, CWE-295, CWE-259, CWE-798
Copilot - M	CWE-327, CWE-397, CWE-259, CWE-798, CWE-295
Copilot - H	CWE-397, CWE-259, CWE-798, CWE-327, CWE-477
Codellama - S	CWE-327, CWE-477, CWE-259, CWE-798, CWE-391
Codellama - M	CWE-327, CWE-477, CWE-397, CWE-391, CWE-295
Codellama - H	CWE-477, CWE-397, CWE-327, CWE-259, CWE-798

Tabella 5.3: Tipologie di CWE più Frequenti

CWE-ID	Debolezza
CWE-259	Use of Hard-coded Password
CWE-295	Improper Certificate Validation
CWE-297	Improper Validation of Certificate with Host Mismatch
CWE-326	Inadequate Encryption Strength
CWE-327	Use of a Broken or Risky Cryptographic Algorithm
CWE-328	Use of Weak Hash
CWE-377	Insecure Temporary File
CWE-379	Creation of Temporary File in Directory
CWE-391	Unchecked Error Condition
CWE-396	Declaration of Catch for Generic Exception
CWE-397	Declaration of Throws for Generic Exception
CWE-457	Use of Uninitialized Variable
CWE-477	Use of Obsolete Function
CWE-478	Missing Default Case in Multiple Condition Expression
CWE-481	Assigning instead of Comparing
CWE-521	Weak Password Requirements
CWE-584	Return Inside Finally Block
CWE-628	Function Call with Incorrectly Specified Arguments
CWE-759	Use of a One-Way Hash without a Salt
CWE-760	Use of a One-Way Hash with a Predictable Salt
CWE-780	Use of RSA Algorithm without OAEP
CWE-798	Use of Hard-coded Credentials

Tabella 5.4: Tipologie di CWE Rilevati

5.2 Analisi Qualità del Codice

Per analizzare in maniera dettagliata l'andamento della qualità del codice, si procederà inizialmente con un'analisi quantitativa delle occorrenze di problemi di qualità in rapporto alle linee di codice per misurare la portata del miglioramento, impiegando dei test statistici per misurare la significatività del fenomeno di miglioramento. Successivamente, l'analisi si focalizzerà da un punto di vista qualitativo sulle tipologie di problemi di qualità più frequenti, per chiarire in modo preciso l'impatto dell'approccio sugli aspetti di qualità del codice.

5.2.1 Analisi Quantitativa

Per valutare l'andamento della qualità del codice prenderemo in considerazione, per ogni livello di prompt sui LLM considerati, le **occorrenze di problemi di qualità** (**#CQ**) e il relativo rapporto **#CQ/LOC**. Nella tabella 5.5 sono riportati tali valori.

Fonte	**#CQ**	**#CQ/LOC**
GPT - S	479	21.60
GPT - M	397	14.89
GPT - H	382	11.00
Gemini - S	472	24.31
Gemini - M	372	18.26
Gemini - H	313	15.15
Copilot - S	391	20.61
Copilot - M	336	15.87
Copilot - H	310	13.31
Codellama - S	376	22.62
Codellama - M	311	21.27
Codellama - H	243	16.69

Tabella 5.5: Risultati dell'Analisi Quantitativa su Problemi di Qualità

Analizzando i risultati, anche in questo caso la strategia di prompting adottata ha fatto riscontrare un miglioramento tra tutti i livelli di prompt su tutti i modelli considerati, sia dal punto di vista delle occorrenze di problemi di qualità che dal rapporto #CQ/LOC.

Per confrontare correttamente l'andamento della qualità del codice, anche in questo caso considereremo direttamente il rapporto #CQ/LOC, per quantificare adeguatamente la loro frequenza nel codice generato. Nello specifico, si registra un miglioramento medio, considerando tutti i modelli, del 21.22% tra prompt Simple e Mid, del 20.20% tra prompt Mid e Hard, e del 37.09% tra prompt Simple e Hard.

Anche in questo caso sono state analizzate le differenze tra i modelli considerati per valutare la generalizzabilità dell'efficacia dell'approccio, considerando la differenza tra prompt Simple e Hard per rappresentare in maniera esplicita il fenomeno di miglioramento. In particolare, GPT è il modello che ha mostrato il miglioramento maggiore con il 49.07%, seguito da Gemini con il 37.67%, Copilot con il 35.41%, e Codellama con il 26.21%. In questo caso, i modelli considerati hanno mostrato differenze più marcate tra i modelli rispetto a quanto osservato con i CWE, con un grado di omogeneità inferiore, indicando che, pur partendo dallo stesso input, i modelli reagiscono diversamente alle direttive sulla qualità del codice. Tuttavia, i progressi generali risultano comunque piuttosto consistenti, con una buona efficacia complessiva sui modelli considerati.

Per chiarire l'andamento, sono stati analizzati i problemi di qualità rispetto ad ognuno dei tre linguaggi di programmazione considerati per l'esperimento. Rispetto a quanto visto per i CWE, il trend osservato è risultato di gran lunga più uniforme tra i vari linguaggi, evidenziando che le capacità di miglioramento della qualità del codice risultano omogenee a prescindere dai linguaggi coinvolti.

Infine, sono stati applicati i dovuti test statistici per dimostrare la significatività statistica dei miglioramenti della qualità del codice osservati. Anche in questo caso è stato prima condotto il **test di Shapiro-Wilk** sui rapporti #CQ/LOC di ogni prompt sui vari modelli, mostrando anche in questo caso una distribuzione non normale, e successivamente il **test di Wilcoxon** sui medesimi valori, i cui risultati sono mostrati nella Tabella 5.6.

L'esito dei test statistici evidenzia come l'impiego di direttive di diversa comples-

sità relative alla qualità del codice ha un impatto statisticamente significativo su tutti i modelli considerati, in maniera ancora più ampia rispetto a quanto misurato sugli aspetti di sicurezza, confermando ulteriormente la validità dell'approccio rispetto alla gestione di aspetti di qualità del codice.

Confronto	**Statistic**	**p-value**	**Differenza statistica**
GPT - S vs M	698.0	5.85e-06	Si
GPT - M vs H	957.0	0.001	Si
GPT - S vs H	557.0	7.26e-08	Si
Gemini - S vs M	879.0	1.43e-04	Si
Gemini - M vs H	1248.5	0.016	Si
Gemini - S vs H	565.0	8.87e-08	Si
Copilot - S vs M	574.0	5.48e-05	Si
Copilot - M vs H	955.0	0.003	Si
Copilot - S vs H	421.0	8.88e-09	Si
Codellama - S vs M	1201.0	0.030	Si
Codellama - M vs H	1302.5	0.045	Si
Codellama - S vs H	1052.0	0.001	Si

Tabella 5.6: Risultati del Test di Wilcoxon su Problemi di Qualità

5.2.2 Analisi Qualitativa

Per valutare dal punto di vista qualitativo le problematiche di qualità del codice generate dai LLM considerati, queste sono state catalogate e classificate in base alla loro tipologia e gravità. I risultati di tale analisi sono mostrati nella Tabella 5.7, in ordine di frequenza.

I problemi riportati sono risultati uniformemente distribuiti tra tutti i modelli e livelli di prompt. I più diffusi sono risultati essere **commenti inline**, **righe troppo lunghe**, e **gestione logicamente errata di condizioni di errore**, seguiti in misura

minore da **utilizzo di variabili o librerie non dichiarate** e **problemi di indentazione e spaziatura**. Le restanti problematiche sono state riscontrate con una frequenza decisamente inferiore rispetto a quelli citati.

Tali risultati ci permettono di affermare che l'utilizzo di direttive mirate verso gli aspetti di qualità del codice ha fatto registrare una marcata transizione verso problematiche più superficiali, con una completa eliminazione di bug, e una notevole riduzione di code smell e problemi di aderenza a standard, sebbene, in misura minore, ancora presenti anche per i prompt più complessi. Tale fenomeno è da attribuire all'incapacità dei modelli linguistici considerati di associare in maniera appropriata determinati approcci e stili di programmazione alla produzione di codice di bassa qualità, in particolar modo per i commenti inline.

La distribuzione uniforme delle problematiche di qualità è emersa anche rispetto ai linguaggi considerati, sebbene ognuno di essi abbia comunque presentato sfide specifiche. In PHP, erano diffuse problematiche di indentazione e gestione delle parentesi, mentre in Python sono emerse alcune criticità con la corretta definizione di variabili. In Java, sono state riscontrate problematiche rispetto a righe di codice eccessivamente lunghe e al corretto naming dei vari elementi nel codice.

Problematica	Tipologia
Commenti inline	Code smell
Righe troppo lunghe	Code smell
Cattiva gestione di condizioni di errore	Bug
Utilizzo di variabili o librerie non dichiarate	Bug
Problemi di indentazione e spaziatura	Aderenza a standard
Errata nomenclatura di elementi del codice	Aderenza a standard
Presenza di codice commentato	Code smell
Variabili dichiarate non utilizzate	Code smell
Mancanza di specificatori a metodi o classi	Aderenza a standard
Metodi vuoti	Code smell

Tabella 5.7: Tipologie di Problemi di Qualità Rilevati

5.3 Discussione dei Risultati

I risultati ottenuti evidenziano come la complessità e la specificità dei prompt influenzino in modo significativo la sicurezza e la qualità complessiva del codice generato, dimostrando l'efficacia generale dell'approccio basato su direttive. Di seguito, verranno presentate le osservazioni più rilevanti emerse dall'analisi condotta.

5.3.1 Miglioramenti sui CWE

L'analisi approfondita dei CWE prodotti rispetto alla strategia definita ha dimostrato l'efficacia dell'approccio nella mitigazione degli aspetti di sicurezza, sia sotto l'aspetto qualitativo che quantitativo. Nello specifico, l'analisi dei CWE ha fatto emergere alcune osservazioni chiave:

- **Meno tipologie di CWE**: L'impiego di prompt più mirati per l'interrogazione ha permesso di riscontrare una notevole diminuzione delle tipologie di CWE, con un miglioramento rispetto al prompt Simple del 12.46% con il prompt Mid e del 24.92% con il prompt Hard, indicando una migliore attenzione dei modelli rispetto alla sicurezza del codice generato;

- **Riduzione occorrenze**: Anche il numero totale di CWE rilevati ha mostrato una notevole diminuzione, con un miglioramento rispetto al prompt Simple del 28.60% con il livello Mid, e del 53.42% con il livello Hard;

- **Miglioramento omogeneo sui LLM**: Sono stati riscontrati miglioramenti particolarmente uniformi tra i vari modelli considerati, confermando la generalizzabilità dell'efficacia dell'approccio sugli aspetti di sicurezza;

- **Limiti comuni sui CWE frequenti**: Tutti i modelli e livelli di prompt condividono lo stesso sottoinsieme di tipologie di CWE più frequenti (CWE-327, CWE-397, CWE-477, CWE-295, CWE-259, CWE-798), fattore che evidenzia la presenza di limiti intrinseci dei modelli rispetto a domini specifici come autenticazione, crittografia, gestione di condizioni di errore e utilizzo di funzione deprecate.

Inoltre, la combinazione di analisi quantitativa e qualitativa ha fatto emergere come le problematiche riscontrate siano da attribuire a caratteristiche e limiti intrinseci di comprensione e ragionamento dei modelli considerati, piuttosto che a caratteristiche peculiari dei prompt o dei dati utilizzati per l'esperimento, che non permettono in tutti i casi la generazione di codice opportunamente sicuro. Tuttavia, l'approccio utilizzato dimostra come, sfruttando direttive mirate, si ha una comprensione sensibilmente maggiore delle istruzioni, del contesto della richiesta, e in maniera generale dell'importanza del compito richiesto da parte dei LLM, con miglioramenti evidenti sul livello di sicurezza del codice generato.

In conclusione, possiamo affermare che le tecniche di prompting scelte hanno dimostrato di avere implicazioni dirette sulla sicurezza del codice generato, mostrando un miglioramento significativo ed omogeneo su tutti i modelli considerati. I risultati del prompt di livello Mid dimostrano come l'aggiunta di istruzioni dirette, mirate ed essenziali sulle aspettative di sicurezza di una certa problematica permette di avere un miglioramento netto sul livello di sicurezza del codice generato. Tale miglioramento è ancora più marcato se si considera il prompt di livello Hard, che con il rafforzamento delle direttive e l'impiego di uno stimolo emozionale ha fatto misurare progressi ancora maggiori rispetto al prompt Simple, con una differenza non trascurabile rispetto al prompt Mid.

5.3.2 Miglioramenti sulla Qualità del Codice

L'analisi approfondita dei problemi di qualità del codice generati con la strategia impiegata ha permesso di riscontrare un miglioramento significativo su tutte le casistiche analizzati, sia dal punto di vista quantitativo che qualitativo. Nello specifico, sono emerse le seguenti osservazioni:

- **Problemi di qualità meno gravi**: L'utilizzo di direttive mirate alla mitigazione dei problemi di qualità ha permesso di ottenere un'eliminazione totale dei bug, ed una notevole riduzione di code smell e problemi di aderenza a standard;
- **Riduzione occorrenze**: La qualità generale del codice è stata notevolmente aumentata, con un miglioramento rispetto al prompt Simple del 21.22% con il li-

vello Mid, e del 37.09% con il livello Hard, dimostrando l'efficacia dell'approccio nella gestione di tali aspetti;

- **Miglioramento meno omogeneo sui LLM**: Sono stati riscontrati miglioramenti meno uniformi tra i vari modelli considerati, indicando una maggiore diversità nella reazione dei modelli alle direttive sulla qualità, seppur con un andamento piuttosto omogeneo, che dimostra l'efficacia generale della strategia anche in questo caso;

Considerando le osservazioni emerse dall'analisi quantitativa e qualitativa, viene dimostrato che le problematiche di qualità del codice riscontrate siano da attribuire all'incapacità dei modelli linguistici considerati di associare determinati approcci e stili di programmazione alla produzione di codice di bassa qualità, fattore che giustifica la persistenza di problematiche di aderenza a standard, e, in misura minore, di alcune tipologie di code smell. Tuttavia, considerando la gravità inferiore delle problematiche riscontrate nei prompt più avanzati e il notevole miglioramento osservato, ci si può comunque ritenere soddisfatti dei risultati ottenuti, che dimostrano come l'impiego di direttive mirate abbia un impatto decisivo anche sulla qualità del codice generato.

In conclusione, possiamo affermare che la strategia di prompting definita ha avuto un impatto significativo sulla qualità rispetto a tutti i livelli di prompt e modelli considerati. Nel caso del livello Mid, l'uso di direttive pratiche e mirate ha già permesso di riscontrare un miglioramento netto delle performance. Il livello Hard, con il potenziamento delle istruzioni e del contesto, insieme all'introduzione di uno stimolo emotivo, è stato in grado di portare un miglioramento ancora maggiore rispetto al Simple, con una differenza non trascurabile rispetto al prompt Mid.

5.3.3 Implicazioni Finali

Sulla base delle osservazioni raccolte durante le varie fasi di sperimentazione e delle informazioni emerse dall'analisi dei risultati, di seguito vengono fornite le risposte alle Research Questions inizialmente definite, in modo da avere un quadro definitivo sull'esito dello studio.

🔍 **RQ$_1$.** *In che modo l'impiego di tecniche di prompting di diversa complessità influisce sulla generazione del codice?*

👉 **Answer to RQ$_1$.** I risultati mostrano che le tecniche di prompting adottate influiscono significativamente sulla sicurezza e la qualità del codice generato.

L'approccio definito ha dimostrato che l'applicazione di tecniche di prompt engineering di diversa complessità è in grado di portare miglioramenti significativi rispetto alla sicurezza e alla qualità del codice generato, sia dal punto di vista quantitativo che qualitativo, evidenziando come il grado di complessità del prompt influisca in maniera netta sulla prestazioni dei modelli. Nello specifico, il miglioramento è stato particolarmente evidente rispetto alle debolezze considerate, con i modelli che hanno dimostrato una maggiore efficacia nella gestione aspetti critici legati alla sicurezza. Anche la qualità del codice ha notevolmente beneficiato dell'approccio definito, dimostrando come la definizione delle giuste direttive permetta di considerare adeguatamente anche questo aspetto.

In merito alla strategia di prompting adottata, il prompt di livello Mid, con l'utilizzo di direttive mirate e equilibrate, che concentrano l'attenzione su sicurezza e qualità, ha mostrato un miglioramento evidente rispetto alla baseline. Il prompt di livello Hard, con il rafforzamento delle istruzioni e del contesto forniti, insieme all'impiego di uno stimolo emotivo, ha mostrato un miglioramento ancora maggiore, con una differenza statisticamente significativa rispetto alla baseline su tutte le casistiche esaminate. Inoltre, è stato interessante osservare come il livello Mid, caratterizzato da un grado di complessità inferiore rispetto al livello Hard, ha comunque mostrato un impatto non trascurabile rispetto a quest'ultimo.

In sintesi, l'esperimento condotto, facendo leva su direttive esplicite e opportunamente bilanciate, ha permesso di ottenere una riduzione significativa delle potenziali vulnerabilità di sicurezza e notevoli miglioramenti nella qualità complessiva del codice generato.

🔍 **RQ$_2$.** *Quanto sono generalizzabili i risultati ottenuti rispetto ai LLM considerati?*

👉 **Answer to RQ$_2$.** I LLM considerati hanno mostrato in maniera comune una reazione significativa ai prompt forniti, dimostrando la generalizzabilità dell'efficacia dell'approccio.

I risultati ottenuti dimostrano l'ampia generalizzabilità dell'efficacia dell'esperimento rispetto ai LLM considerati. In particolare, tutti i modelli hanno reagito in maniera positiva all'approccio di prompting, indicando l'efficacia di quest'ultimo nel migliorare la sicurezza e la qualità del codice generato.

Considerando la diversità delle caratteristiche intrinseche dei LLM testati, i risultati ottenuti sottolineano la validità della strategia, confermando la generalizzabilità dell'efficacia dell'approccio di prompting.

La struttura equilibrata e organizzata delle richieste effettuate, ottenuta tramite una meticolosa ingegnerizzazione delle tecniche di prompting, ha permesso ai modelli di rispondere con elevata efficacia rispetto alle problematiche poste. Inoltre, considerando che l'esperimento è stato condotto su osservazioni che rispecchiano problematiche di programmazione reali, viene fornita un'ulteriore prova della validità generale dell'approccio, dimostrando che questa metodologia può essere applicata con successo anche in contesti dove gli aspetti di sicurezza e qualità del codice risultano essere critici.

CAPITOLO 6

Minacce alla Validità

Questo capitolo esplora le principali minacce alla validità dello studio condotto, suddividendole in varie categorie. Per ognuna saranno discussi i fattori principali che potrebbero influenzare la validità dei risultati ottenuti, insieme alle precauzioni adottate per mitigarli.

Per esaminare le minacce alla validità del presente studio, ci concentreremo su quattro categorie di validità. Per ciascuna di queste categorie, verrà analizzato l'impatto potenziale di tali minacce sulla validità dei risultati e saranno illustrate le strategie messe in atto per ridurre al minimo i rischi associati.

6.1 Minacce alla Validità della Conclusione

Le minacce alla validità della conclusione riguardano le incertezze rispetto all'accuratezza delle inferenze tratte dai risultati dello studio, che per questo studio comprendono:

- **Corpo delle Domande di Stack Overflow**: Una volta collezionate le domande, la forma verbale, sintattica e semantica del corpo delle stesse non è stato curato, al fine di rappresentare fedelmente la complessità di applicazioni reali. Tuttavia, andrebbero analizzate le differenze prestazionali derivate dall'utilizzo di domande con una forma più curata e comprensibile;
- **Dati Utilizzati**: Sebbene sia stato fatto uno sforzo notevole per massimizzare l'eterogeneità del dataset rispetto a linguaggi e contesti applicativi, lo stesso potrebbe non rappresentare pienamente le problematiche del mondo reale. Pertanto, in futuro sarebbe opportuno estendere il campione considerato per valutare le prestazioni su scala più ampia.

6.2 Minacce alla Validità del Costrutto

Le minacce alla validità di costrutto si riferiscono a problemi che compromettono la capacità di uno studio di misurare accuratamente il concetto teorico che intende valutare. Nello studio non sono state rilevate minacce di costrutto da riportare.

6.3 Minacce alla Validità Interna

Le minacce alla validità interna coinvolgono qualsiasi fattore che può distorcere i risultati di uno studio, rendendo incerta la relazione causale tra le variabili analizzate. È stata rilevata un'unica minaccia alla validità interna per questo studio:

- **Tecniche di Prompting Utilizzate**: Sebbene sia stata effettuata un'accurata analisi preliminare per valutare la combinazione di tecniche che avesse l'impatto maggiore sugli aspetti considerati, non va esclusa la potenziale efficacia di altre strategie in questo contesto. Pertanto, sono necessari ulteriori studi per determinare in maniera approfondita la validità degli approcci di prompting definiti nella generazione di codice.

6.4 Minacce alla Validità Esterna

Le minacce alla validità esterna rappresentano condizioni che limitano la capacità di generalizzare i risultati di un certo esperimento al mondo reale. Nel nostro studio, queste includono:

- **Qualità delle Domande di Stack Overflow**: Considerando la natura eterogenea delle domande presenti su Stack Overflow, che possono variare per complessità, contesto e livello di dettaglio, i risultati potrebbero non essere standardizzabili. Per mitigare tale problematica, è stato posto massimo focus sulla varietà delle domande collezionate, tramite la definizione di appositi criteri di raccolta e selezione delle osservazioni;

- **Natura Stocastica dei LLM**. Dato che ogni prompt può produrre output differenti, talvolta è possibile riscontrare un grado di incoerenza tra le performance di un certo modello in un determinato contesto. A tal proposito, sono necessari ulteriori studi che confermino la bontà dell'approccio definito sulla generazione di codice tramite LLM.

CAPITOLO 7

Conclusioni

Il capitolo finale riassume i risultati ottenuti, mettendo in luce il contributo dello studio nel campo della generazione di codice sicuro tramite LLM. Viene sottolineata l'efficacia del prompt engineering nella riduzione delle debolezze di sicurezza e nel miglioramento della qualità complessiva del codice prodotto, evidenziando l'efficacia della strategia definita nella gestione degli aspetti considerati. Vengono infine discussi i limiti dello studio condotto, e proposte direzioni future per ulteriori ricerche.

L'obiettivo principale di questo studio è stato la valutazione dell'impatto di un catalogo di template di prompt engineering di diversa complessità sulla sicurezza e la qualità del codice generato, analizzando gli effetti della loro applicazione su varie problematiche di programmazione rappresentative in maniera eterogenea di contesti di produzione reali.

Per determinare in tal senso la strategia più efficace, sono stati esplorati due approcci preliminari. L'approccio esplicito, mirato alla gestione specifica delle problematiche di sicurezza e qualità più frequenti, ha fatto emergere come l'eccessiva specificità delle direttive impiegate non permetta una corretta gestione degli aspetti di sicurezza e qualità del codice. Di conseguenza, è stato sperimentato un approccio generico, che si concentrasse sull'equilibrio tra ricchezza informativa e complessità strutturale della richiesta, evitando dettagli eccessivamente specifici, mostrando un impatto molto più marcato rispetto all'approccio precedente.

Sulla base delle prestazioni riscontrate, è stata sfruttata quest'ultima strategia per guidare la sperimentazione finale, definendo appositi template di prompting, costituiti da direttive mirate ed equilibrate, che fossero in grado di migliorare efficacemente la sicurezza e qualità del codice prodotto. Per ottenere una visione completa della bontà dell'esperimento, sono state collezionate 90 domande da Stack Overflow, relative ad un'ampia varietà di problematiche reali di programmazione. Ogni domanda è stata testata con ognuno dei template definiti sui vari modelli considerati, e gli snippet di codice prodotti analizzati tramite Sonarlint, ricavando informazioni quantitative e qualitative sulla sicurezza e la qualità del codice generato.

In seguito ad un'analisi approfondita sulla base dei dati raccolti, è emersa chiaramente l'efficacia della strategia di prompting adottata, che ha fatto registrare una riduzione considerevole dei problemi di sicurezza e qualità del codice, con miglioramenti che hanno evidenziato un impatto statisticamente significativo su tutte le osservazioni esaminate. Considerando gli aspetti di sicurezza analizzati, l'impiego del prompt engineering ha portato ad un miglioramento medio rispetto alla baseline fino al 53.43%, con una riduzione fino al 24.92% delle tipologie di CWE. Rispetto alla qualità del codice, è stato osservato un miglioramento medio rispetto alla baseline fino al 37.09%, in aggiunta alla totale eliminazione di bug ed una massiccia riduzione di code smell e problemi di aderenza a standard di programmazione. Tenendo conto

della varietà delle problematiche di programmazione considerate e della natura omogenea del miglioramento riscontrato su tutti i modelli linguistici considerati, le osservazioni raccolte dimostrano la generalizzabilità dell'efficacia dell'approccio proposto nel migliorare la sicurezza e la qualità del codice generato.

Nonostante gli ottimi risultati ottenuti, vanno sottolineati alcuni limiti. Innanzitutto, la natura stocastica dei LLM comporta una variabilità intrinseca nei risultati, anche a parità di input, rendendo necessari ulteriori esperimenti per confermare il fenomeno osservato. Allo stesso modo, il dataset utilizzato, sebbene rappresentativo di una vasta gamma di problematiche, potrebbe non coprire appieno le complessità presenti in applicazioni reali, limitando la generalizzabilità dei risultati. Inoltre, non tutte le domande considerate nell'esperimento possiedono la stessa chiarezza sintattica e semantica. In tal senso, sarebbe interessante approfondire come la chiarezza della richiesta influenzi la qualità delle risposte ottenute. Inoltre, sarebbe interessante approfondire gli effetti di tecniche di prompting basate su direttive su altre attività di sviluppo software, per valutarne l'efficacia su contesti più ampi.

Le implicazioni pratiche di questo studio sono significative. L'impiego del prompt engineering rappresenta un passo fondamentale per migliorare l'affidabilità dei LLM, soprattutto in contesti dove la sicurezza e la qualità del codice costituiscono aspetti cruciali. Il presente studio, ricreando in maniera fedele il contesto reale in cui i programmatori interagiscono con i modelli linguistici, dimostra come un'attenta definizione dei prompt permetta di considerare in maniera molto più efficace gli aspetti di sicurezza e qualità del codice. Lo studio contribuisce alla definizione di un approccio pratico e mirato, evidenziandone l'impatto sulle performance dei modelli linguistici, e promuovendone l'utilizzo nell'interazione con i LLM. Tuttavia, è necessario continuare a esplorare gli effetti di tali tecniche, estendendone la sperimentazione in scenari applicativi sempre più ampi e complessi.

Bibliografia

[1] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff, P. S. Dulepet, S. Vidyadhara, D. Ki, S. Agrawal, C. Pham, G. Kroiz, F. Li, H. Tao, A. Srivastava, H. D. Costa, S. Gupta, M. L. Rogers, I. Goncearenco, G. Sarli, I. Galynker, D. Peskoff, M. Carpuat, J. White, S. Anadkat, A. Hoyle, and P. Resnik, "The prompt report: A systematic survey of prompting techniques," 2024. [Online]. Available: https://arxiv.org/abs/2406.06608 (Citato alle pagine iv, 9, 12 e 18)

[2] IBM, "What are large language models (llms)?" 2024, last accessed 22 September 2024. [Online]. Available: https://www.ibm.com/topics/large-language-models (Citato alle pagine 2 e 7)

[3] OxJournal, "A meta-analysis of the economic, social, legal, and cultural impacts of widespread adoption of large language models such as chatgpt," 2023, last accessed 23 September 2024. [Online]. Available: https://www.oxjournal.org/economic-social-legal-cultural-impacts-large-language-models/#comments (Citato alle pagine 2 e 7)

[4] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. Chatterji, A. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy,

K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. Roohani, C. Ruiz, J. Ryan, C. Ré, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang, "On the opportunities and risks of foundation models," 2022. [Online]. Available: https://arxiv.org/abs/2108.07258 (Citato alle pagine 2 e 7)

[5] V. Gupta and G. Lehal, "A survey of text summarization extractive techniques," *Journal of Emerging Technologies in Web Intelligence*, vol. 2, 08 2010. (Citato alle pagine 2 e 13)

[6] X. Liu, Y. Tan, Z. Xiao, J. Zhuge, and R. Zhou, "Not the end of story: An evaluation of ChatGPT-driven vulnerability description mappings," in *Findings of the Association for Computational Linguistics: ACL 2023*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 3724–3731. [Online]. Available: https://aclanthology.org/2023.findings-acl.229 (Citato alle pagine 2 e 14)

[7] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do users write more insecure code with ai assistants?" in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. ACM, Nov. 2023. [Online]. Available: http://dx.doi.org/10.1145/3576915.3623157 (Citato alle pagine 2, 8, 14 e 15)

[8] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, ser. FAccT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 610–623. [Online]. Available: https://doi.org/10.1145/3442188.3445922 (Citato alle pagine 2, 8 e 15)

[9] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, p. 1–38, Mar. 2023. [Online]. Available: http://dx.doi.org/10.1145/3571730 (Citato alle pagine 2 e 8)

[10] IBM, "What is prompt engineering?" 2024, last accessed 18 September 2024. [Online]. Available: https://www.ibm.com/topics/prompt-engineering (Citato alle pagine 2 e 9)

[11] J. Wang, L. Cao, X. Luo, Z. Zhou, J. Xie, A. Jatowt, and Y. Cai, "Enhancing large language models for secure code generation: A dataset-driven study on vulnerability mitigation," 2023. [Online]. Available: https://arxiv.org/abs/2310.16263 (Citato alle pagine 2, 13, 15 e 17)

[12] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, "Unleashing the potential of prompt engineering in large language models: a comprehensive review," 2024. [Online]. Available: https://arxiv.org/abs/2310.14735 (Citato alle pagine 2, 3, 12 e 14)

[13] K. Chang, S. Xu, C. Wang, Y. Luo, T. Xiao, and J. Zhu, "Efficient prompting methods for large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2404.01077 (Citato alle pagine 3, 14 e 20)

[14] M. L. Siddiq, J. C. S. Santos, S. Devareddy, and A. Muller, "Sallm: Security assessment of generated code," 2024. [Online]. Available: https://arxiv.org/abs/2311.00889 (Citato alle pagine 3, 16 e 36)

[15] Y. Yang, X. Xia, D. Lo, and J. Grundy, "A survey on deep learning for software engineering," 2020. [Online]. Available: https://arxiv.org/abs/2011.14597 (Citato alle pagine 7, 8, 14 e 36)

[16] Sonatype, "Generative ai adoption surges in software development despite security risks, sonatype research finds," 2023, last accessed 24 September 2024. [Online]. Available: https://www.sonatype.com/en/press-releases/generative-ai-adoption-surges-in-software-development (Citato a pagina 7)

[17] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2402.06196 (Citato a pagina 7)

[18] R. Tian, Y. Ye, Y. Qin, X. Cong, Y. Lin, Y. Pan, Y. Wu, H. Hui, W. Liu, Z. Liu, and M. Sun, "Debugbench: Evaluating debugging capability of large language models," 2024. [Online]. Available: https://arxiv.org/abs/2401.04621 (Citato a pagina 8)

[19] S. S. Dvivedi, V. Vijay, S. L. R. Pujari, S. Lodh, and D. Kumar, "A comparative analysis of large language models for code documentation generation," in *Proceedings of the 1st ACM International Conference on AI-Powered Software*, ser. AIware 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 65–73. [Online]. Available: https://doi.org/10.1145/3664646.3664765 (Citato alle pagine 8 e 13)

[20] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," 2018. [Online]. Available: https://arxiv.org/abs/1801.06146 (Citato a pagina 8)

[21] T. Schick and H. Schütze, "Exploiting cloze questions for few shot text classification and natural language inference," 2021. [Online]. Available: https://arxiv.org/abs/2001.07676 (Citato alle pagine 8 e 12)

[22] 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," 2024. [Online]. Available: https://arxiv.org/abs/2402.07927 (Citato alle pagine 10 e 11)

[23] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023. [Online]. Available: https://arxiv.org/abs/2201.11903 (Citato alle pagine 10, 16, 17 e 20)

[24] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2312.10997 (Citato alle pagine 11, 13, 17 e 20)

[25] X. Chen, R. Aksitov, U. Alon, J. Ren, K. Xiao, P. Yin, S. Prakash, C. Sutton, X. Wang, and D. Zhou, "Universal self-consistency for large language model generation," 2023. [Online]. Available: https://arxiv.org/abs/2311.17311 (Citato alle pagine 11, 13, 17 e 20)

[26] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165 (Citato alle pagine 11 e 13)

[27] T. Tang, J. Li, W. X. Zhao, and J.-R. Wen, "Context-tuning: Learning contextualized prompts for natural language generation," in *Proceedings of the 29th International Conference on Computational Linguistics*. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, Oct. 2022, pp. 6340–6354. [Online]. Available: https://aclanthology.org/2022.coling-1.552 (Citato alle pagine 11 e 13)

[28] T. Nguyen, "Prompt engineering – part 1: Persona pattern," 2024, last accessed 25 September 2024. [Online]. Available: https://travisbytes.com/prompt-engineering-part-1-persona-pattern/ (Citato a pagina 12)

[29] A. Maynard, "Constraint based prompting," 2023, last accessed 25 September 2024. [Online]. Available: https://andrewmaynard.net/constraint-based-prompts/ (Citato a pagina 12)

[30] C. Li, J. Wang, Y. Zhang, K. Zhu, W. Hou, J. Lian, F. Luo, Q. Yang, and X. Xie, "Large language models understand and can be enhanced by emotional stimuli," 2023. [Online]. Available: https://arxiv.org/abs/2307.11760 (Citato alle pagine 12 e 13)

[31] C. Ma, X. Zhao, C. Zhang, Y. Qin, and W. Zhang, "When emotional stimuli meet prompt designing: An auto-prompt graphical paradigm," 2024. [Online]. Available: https://arxiv.org/abs/2404.10500 (Citato alle pagine 12 e 13)

[32] S. Vatsal and H. Dubey, "A survey of prompt engineering methods in large language models for different nlp tasks," 2024. [Online]. Available: https://arxiv.org/abs/2407.12994 (Citato alle pagine 12, 14, 15 e 20)

[33] L. Reynolds and K. McDonell, "Prompt programming for large language models: Beyond the few-shot paradigm," 2021. [Online]. Available: https://arxiv.org/abs/2102.07350 (Citato alle pagine 13 e 16)

[34] T. Schick, S. Udupa, and H. Schütze, "Self-diagnosis and self-debiasing: A proposal for reducing corpus-based bias in NLP," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 1408–1424, 2021. [Online]. Available: https://aclanthology.org/2021.tacl-1.84 (Citato a pagina 13)

[35] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? assessing the security of github copilot's code contributions," 2021. [Online]. Available: https://arxiv.org/abs/2108.09293 (Citato alle pagine 13 e 15)

[36] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," 2024. [Online]. Available: https://arxiv.org/abs/2406.00515 (Citato a pagina 14)

[37] S. Kabir, D. Udo-Imeh, B. Kou, and T. Zhang, "Who answers it better? an in-depth analysis of chatgpt and stack overflow answers to software engineering questions," 08 2023. (Citato a pagina 15)

[38] S. Hamer, M. d'Amorim, and L. Williams, "Just another copy and paste? comparing the security vulnerabilities of chatgpt generated code and stackoverflow answers," in *2024 IEEE Security and Privacy Workshops (SPW)*, 2024, pp. 87–94. (Citato alle pagine 15, 16 e 25)

[39] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," 2021. [Online]. Available: https://arxiv.org/abs/2107.03374 (Citato alle pagine 15, 16 e 17)

[40] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton, "Program synthesis with large language models," 2021. [Online]. Available: https://arxiv.org/abs/2108.07732 (Citato alle pagine 16 e 17)

[41] T. Ridnik, D. Kredo, and I. Friedman, "Code generation with alphacodium: From prompt engineering to flow engineering," 2024. [Online]. Available: https://arxiv.org/abs/2401.08500 (Citato a pagina 17)