UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Analisi di Comprensibilità del Codice Generato da ChatGPT

RELATORE

Prof. Fabio Palomba

Dott.ssa Giulia Sellitto

CANDIDATO

Annamaria Basile

Matricola: 0512108137

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

Abstract

L'utilizzo diffuso di ChatGPT nella generazione di codice, sia tra gli sviluppatori che tra gli studenti, è notevole. Questo modello di linguaggio avanzato, basato su GPT di OpenAI, permette interazioni naturali, ma va sottolineato che il codice prodotto non è soggetto a un controllo rigoroso, compromettendo la sua qualità.

Per questo motivo, è essenziale focalizzarsi sulla comprensibilità del codice generato, considerando la leggibilità e la presenza di antipattern linguistici.

L'obiettivo di questo lavoro di tesi è analizzare la qualità del codice generato da ChatGPT dal punto di vista della comprendibilità.

Per studiare queste caratteristiche è stato ideato e realizzato un questionario somministrato agli studenti. In questo questionario sono stati raccolti dei prompt usati per generare del codice Java. A partire da questi snippet di codice è stata effettuata un'analisi attraverso due tool proposti in letteratura. Il primo tool calcola una serie di metriche di leggibilita su diversi modelli proposti, mentre il secondo tool è capace di indivduare fino a 19 Linguistic Antipattern in uno snippet di codice.

L'analisi ha evidenziato che il codice prodotto da ChatGPT non è ottimale dal punto di vista della comprensione del codice, sottolineando l'importanza di non fare affidamento esclusivo sulle generazioni automatiche di codice da parte del modello.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

Negli attuali contesti di sviluppo, l'uso di ChatGPT per generare codice si è diffuso notevolmente, sia tra gli sviluppatori che tra gli studenti. Questo modello di linguaggio avanzato, sviluppato da OpenAI sulla base dell'architettura GPT (Generative Pre-trained Transformer), è progettato per interpretare e generare testo in modo naturale, consentendo interazioni significative e fluide con gli utenti.

Tuttavia è importante notare che il codice prodotto attraverso questo processo non è sottoposto ad un adeguato controllo, quindi la sua qualità non è garantita.

Pertanto, è necessario prestare attenzione alla comprensibilità del codice generato. Alcuni attributi di cruciale importanza includono la leggibilità del codice e l'assenza di Antipattern Linguistici, ovvero soluzioni scadenti a problemi di progettazione ricorrenti. Assicurarsi che il codice sia chiaramente comprensibile rappresenta una priorità, verificando quindi che il codice sia leggibile e non contenga Antipatterns.

1.2 Motivazione ed Obiettivi

La motivazione che ha spinto alla realizzazione di questa tesi è dato dal fatto che lavorare con codici di alta qualità rappresenta un fattore significativo per i processi di sviluppo e di manutenzione dei progetti software. Tale qualità nel contesto dei codici generati da ChatGPT va misurata in termini di comprensibilità. L'obiettivo di questa tesi è di testare se i codici generati da ChatGPT siano leggibili e privi di Antipattern attraverso dei tool che si occupano di riconoscere se il codice generato sia leggibile o se presenta degli antipattern.

◎ **Obiettivo.** Analizzare la qualità del codice generato da ChatGPT dal punto di vista della comprensibilità.

1.3 Risultati Ottenuti

Per raggiungere l'obiettivo sono stati studiati la leggibilità e i Linguistic Antipattern degli snippet di codice generati automaticamente a partire da prompt ottenuti tramite un questionario.

Il tool IDEAL[1] ha mostrato come su 27 snippet di codice, 11 presentano dei Linguistic Antipattern, ovvero circa il 40,74%, e 2 di essi ne presentano più di uno. C'è da notare che nonostante quasi la metà dei codici generati presenta antipattern, questi sono sempre gli stessi, ovvero D.1, E.1, ed G.2 (descritti nelle Tabelle 3.4 e 3.5), il che è una nota positiva in quanto ChatGPT ha problemi con pochi antipattern.

Il tool di Scalabrino et al.[2], ha identificato problemi di leggibilita del codice generato riscontrando una carenza di commenti per parola, ovvero la proporzione tra codice e relativi commenti. Parallelamente si è notata anche una bassa leggibilità di tali commenti. Inoltre sono stati riscontrati altri problemi di leggibilità in cui gli snippet di codice possono risultare ambigui per l'alto livello di polisemia, ovvero la falcoltà di una parola di assumere più significati.

In generale, l'analisi ha riportato che il codice generato da ChatGPT non è perfetto dal punto di vista della code comprehension, concludendo che non ci si può affidare completamente al codice generato da esso.

Il materiale completo di risultati e dataset è disponibile nell'appendice online. [1]

1.4 Struttura della Tesi

In questa sezione viene presentata la struttura della tesi.

- **Capitolo 2:** in questo capitolo si delinea lo stato dell'arte, in cui viene spiegato dettagliatamente cos'è la Code Generation con l'utilizzo di ChatGPT e le sue applicazioni nello sviluppo del software. Inoltre viene introdotto il concetto di Program Comprehension, esaminando come questa competenza si colleghi con la Code Generation.
- **Capitolo 3:** questo capitolo si concentra sulla scelta dell'obiettivo basato sulla leggibilità del codice generato, esemaninando, inoltre, la scelta dei tool necessari per raggiungerlo.
- **Capitolo 4:** in questo capitolo, verrà condotta un'analisi approfonfita dei risultati ottenuti attraverso i tool utilizzati.
- **Capitolo 5:** in questo capitolo sono presenti le conclusioni rispetto al lavoro di tesi, e dei possibili sviluppi futuri.

[1] `https://github.com/Annamariabasile/Comprehension_analysis`

CAPITOLO 2

Stato dell'Arte

2.1 Code Generation

La generazione del codice è il processo di scrittura del codice sorgente basato su una descrizione del problema o un modello antologico come un modello ed è realizzato con uno strumento di programmazione.

In generale, la generazione del codice è una pratica della programmazione automatica, ovvero un tipo di programmazione per computer in cui un meccanismo genera un programma per computer per consentire ai programmatori umani di scrivere il codice ad un livello di astrazione più elevato.

L'obiettivo è migliorare la produttività del programmatore. La generazione del codice presenta dei vantaggi che la rendono interessante per gli utenti, tra i quali:

- **Risparmio di tempo**: la generazione del codice può avere un tempo di rilascio più rapido. Poichè i computer sono dispositivi automatizzati, la scrittutra del codice generato può essere più efficiente in termini di tempo.
- **Riutilizzo del codice**: la generazione del codice può essere adattato a diverse applicazioni, risparmiando tempo. Il riuso del codice può aiutare ad aumentare la produttività e a mantenere la coerenza delle applicazioni.

- **Meno errori umani**: essendo generato da una macchina, il codice può avere meno errori in quanto non è condizionato dal programmatore. Ciononostante, non è scontato che esso sia privo di difetti.

Nonostante la generazione automatica del codice sia molto utile, c'è bisogno di fare attenzione e controllare attentamente il risultato finale. Infatti, queste tecniche non sono assolutamente prive di difetti, e possono presentare dei possibili svantaggi.

È necessario quindi effettuare un'analisi sulla qualità del codice, in termini soprattutto di (1) code smell e (2) leggibilità.

Nell'ambito della ricerca, sono stati effettuati degli studi con l'obiettivo di verificare la presenza di code smell nel codice generato automaticamente [3], dimostrando come queste porzioni di codice non siano affatto prive di problemi.

Per quanto riguarda la leggibilità del codice generato, anche in termini di code comprehension, non sono stati svolti sufficienti esperimenti. Per questo motivo, l'obiettivo del presente lavoro è effettuare la suddetta analisi.

2.1.1 Transformer-based Code Generation

I transformers sono modelli di apprendimento basati sull'attenzione, introdotti nel 2017 per la prima volta da Vaswani et al. [4]. Questi modelli hanno rivoluzionato il campo dell'eleaborazione del linguaggio naturale, fornendo un'alternativa alle architetture di reti neurali ricorrenti (RNN). I transformers utilizzano l'attenzione per elaborare sequenze di input. L'attenzione permette al modello di dare maggiore importanza a determinate parti dell'input, in modo da ignorare informazioni meno importanti. La capacità di prestare attenzione a parti specifiche dell'input rende i transformers adatti all'elaborazione del linguaggio naturale, dove l'informazione rilevante può disperdersi all'interno di una sequenza di parole.

I transformers hanno rivoluzionato il campo dell'elaborazione del linguaggio naturale, fornendo un'alternativa efficiente alle architetture di reti neurali ricorrenti (RNN) utilizzate precedentemente. Questi modelli sono stati applicati con successo in molte aree della NLP (Natural Language Processing).

In particolare, tra le varie aree di applicazione, una delle più esplorate recentemente è quella della generazione di testo. I modelli di generazione di testo basati sui

transformers sono in grado di generare testo in modo autonomo, senza la necessità di input umani. Questi modelli sono stati utilizzati in molte applicazioni, e la generazione di codice menzionata nella sezione precedente è una di queste. Tra i modelli di transformes più conosciuti troviamo:

- **BERT** (Bidirectional Encoder Representations from Transformers) [5]: applica il suo meccanismo di auto-attenzione per apprendere informazioni da un testo in modo bidirezionale, sostituendo la consueta analisi di una porzione di testo da sinistra a destra o da destra a sinistra. Di conseguenza, acquisisce una profonda comprensione del testo.

- **GPT-3**(Generative Pre-trained Transformer): [6] è in grado di generare testo coerente e facilmente comprensibile. È stato addestrato su una vasta quantità di testo proveniente da Internet, il che gli conferisce una notevole capacità di comprendere il linguaggio e generare testo in diversi contesti. GPT-3 può essere utilizzato in diversi ambiti, come traduzione, generazione di testo creativo, risposta a domande ed elaborazione del linguaggio naturale. Grazie alle sue caratteristiche avanzate, GPT-3 ha attirato grande interesse e attenzione nella comunità di ricerca e sviluppo nel campo del linguaggio naturale.

2.1.2 ChatGPT

ChatGPT è nato come un progetto di ricerca di OpenAI, partendo da un modello di linguaggio naturale chiamato InstructGPT [6], in modo tale da ottenere un modello più specifico per la conversazione.

È stato sviluppato attraverso il miglioramento di GPT-3 [6] utilizzando una base di dati supervisionata, raccolta tramite OpenAI API e il contributo di persone che hanno fornito risposte adeguate alle richieste degli utenti.

ChatGPT è stato progettato per svolgere una varietà di task nel contesto della conversazione e dell'interazione con gli utenti. È in grado di svolgere diverse funzioni, come: fornire assistenza virtuale, rispondere a domande, creare conversazioni realistiche e generare testo coerente. Lo scopo principale è quello di creare un'eperienza di conversazione naturale e fornire risposte e informazioni utili agli utenti.

2.2 Program Comprehension

La Program Comprehension si riferisce alla capacità di un lettore di comprendere il codice e le sue funzionalità, e può essere considerata oggettiva in certa misura. In letteratura è stata studiata dal punto di vista cognitivo, fornendo diversi metodi e teorie per descriverla ed analizzarla. [7]

I programmatori che hanno esperienza e affrontano diversi progetti nel corso della loro carriera tendono ad essere più abili nel leggere codice software di vario tipo. Il principio del codice pulito è proprio la leggibilità, il che significa che il codice dovrebbe essere comprensibile anche per chi ha poca familiarità con il sistema.

La comprensione del programma avviene principalmente prima di apportare modifiche al codice. Molto spesso in progetti molto grandi alcune parti non vengono modificate per molto tempo, e gli sviluppatori devono esplorare il codice sorgente e altri artefatti per identificare e comprendere la porzione di codice rilevante per il cambiamento previsto.

Precedentemente è stato parlato di code generation, di quanto fosse importante generare codice e di tutti i vantaggi che porterebbe. Ma non ci sono abbastanza studi per stabilire se il codice generato automaticamente sia comprensibile o meno.

CAPITOLO 3

Metodologia di ricerca

3.1 Obiettivo e domande di ricerca

Dall'approfondimento della letteratura emerge che il codice generato in maniera automatica presenta alcune imperfezioni e limitazioni. Recentemente ChatGPT ha guadagnato notevole popolarità sia in contesti aziendali che universitari.

In un contesto aziendale, è molto importante che il codice sia ben manutenibile, ovvero di essere modificabile o riparabile in modo efficace, efficiente e rapido. Avere un codice manutenibile significa risparmiare in costi e tempi. Dato che molto spesso il codice generato viene usato da professionisti, e dato che un codice ben comprensibile risulta più facile da manutenere, è necessario studiare questo attributo nel contesto del codice generato.

In un contesto universitario, molto spesso gli studenti utilizzano strumenti di generazione di codice per studiare e fare esercizi. Un codice poco comprensibile può portare a problemi di apprendimento. Per questo motivo è importante che per gli studenti il codice sia comprensibile facilitando così sia l'uso che la condivisione.

Queste problematiche portano ad un'ulteriore questione di ricerca: la mancanza di studi concreti sull'affidabilità del processo di generazione del codice di alta qualità

attraverso l'utilizzo di ChatGPT. Pertanto, l'obiettivo del presente lavoro di tesi è di studiare aspetti di comprensibilità del codice generato da ChatGPT:

◎ **Obiettivo.** Analizzare la qualità del codice generato da ChatGPT dal punto di vista della comprensibilità.

Definiamo delle Research Questions per raggiungere l'obiettivo preposto.

Dal punto di vista della comprensibilità, il codice potrebbe risultare difficile da leggere poichè possono presentarsi termini ambigui all'interno di esso. Definiamo così la prima Research Question:

🔍 **RQ$_1$.** *Il codice generato da ChatGPT presenta problemi di leggibilità?*

Un impatto negativo sulla comprensione del codice generato è dato dalla presenza di Antipatterns, ovvero soluzioni scadenti a problemi di progettazione ricorrenti [8, 9]. Diversi studi dimostrano come i Linguistic Antipattern possano causare problemi di incompresione agli sviluppatori che interpretano in maniera errata il codice, spendendo più tempo del dovuto nella sua comprensione e peggiorando, di conseguenza, la manutenibilità. Definiamo quindi la seconda Research Question:

🔍 **RQ$_2$.** *Il codice generato da ChatGPT contiene Antipatterns linguistici?*

3.2 Metodo di ricerca

In Figura 3.1 vengono mostrati i vari step. Nello Step 1 è stato sviluppato un questionario con lo scopo di ottenere prompt per ChatGPT che permettono di generare del codice. Questo questionario è stato poi somministrato a studenti di informatica iscritti ad un corso di laurea triennale o magistrale. Con le risposte ricevute è stato creato un dataset di snippet di codice in Java.

Per quanto riguarda lo Step 2, gli snippet di codice sono stati elaborati da un tool capace di identificare le metriche di leggibilità del codice [2]. Successivamente i risultati sono stati analizzati.

Lo stesso approccio è stato usato nello Step 3, facendo analizzare il codice dal tool IDEAL [1], un tool capace di identificare i Linguistic Antipatterns.

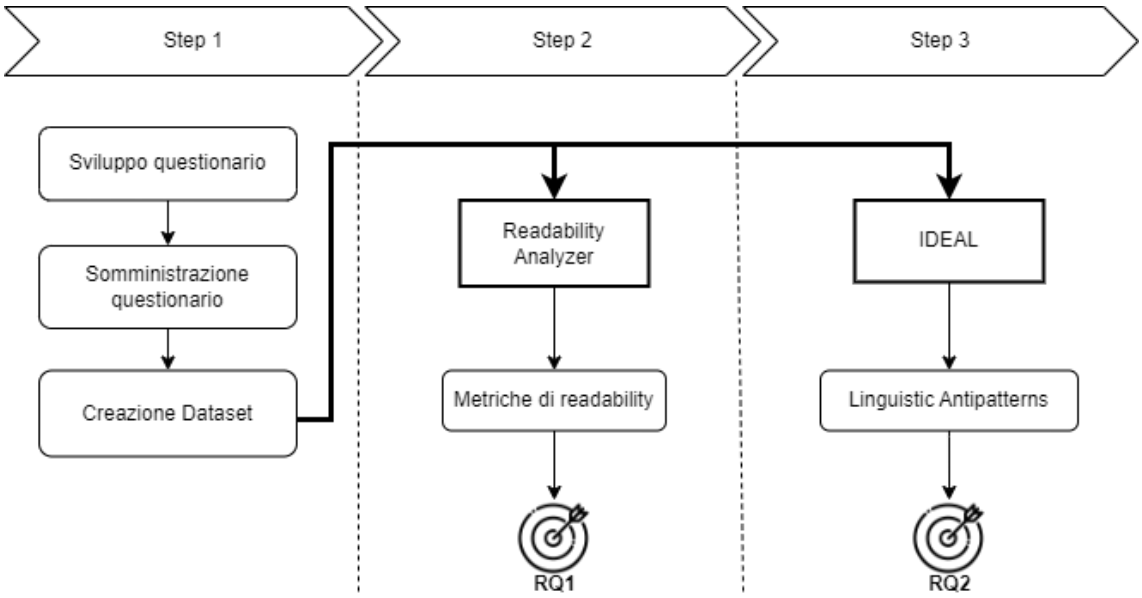

Figura 3.1: Rappresentazione grafica della metodologia di ricerca.

3.2.1 Costruzione del Dataset

È stato ideato e implementato un questionario con l'obiettivo di esplorare e comprendere i prompt usati dagli studenti per generare snippet di codice attraverso l'interazione con ChatGPT.

Lo scopo della creazione di questo questionario e della sua somministrazione agli studenti iscritti al corso di laurea in informatica è ottenere prove concrete. Ciò è motivato dal fatto che gli studenti hanno richiesto a ChatGPT codici per supportare l'apprendimento e la realizzazione di progetti universitari. Grazie alle risposte degli studenti, questi prompt hanno permesso di ottenere un dataset di codici Java realmenete generati e utilizzati in progetti universitari. In questo modo lo studio di attributi di comprensibilità in tali snippet risulta attendibile.

La struttura del questionario è stata attentamente elaborata per fornire una panoramica approfondita delle scelte e delle preferenze degli studenti in merito ai codici che vogliono far generare.

Quest'ultimo si compone di cinque domande mostrate nella Tabella 3.1, progettate in modo da catturare una varietà di informazioni significative. Tra queste, due domande sono strutturate in formato a risposta multipla, offrendo agli studenti diverse opzioni tra cui scegliere in base alle loro esperienze e percezioni. Le restanti

Tabella 3.1: Risposte questionario

#	Domanda
1	Hai sostenuto l'esame di Ingnegneria del Software?
2	Qual era l'ambito del tuo progetto?
3	Quali sono stati i task più difficili durante l'implementazione? Puoi elencare più risposte.
4	Hai utilizzato ChatGPT/Stack Overflow per farti suggerire codice per l'implementazione?
5	Se hai risposto di "Si" alla domanda precedente, cosa hai chiesto/cercato? Cerca di ricordare quante più domande possibili.

tre domande sono formulate in modo aperto, consentendo agli studenti di esprimere liberamente le proprie opinioni e dettagliare le loro esperienze nel processo di generazione di snippet di codice.

Il questionario è stato distribuito agli studenti universitari iscritti al corso di laurea in informatica, sia triennale che magistrale, al fine di raccogliere le risposte fornite dagli studenti e generare un dataset di prompt da sottoporre a ChatGPT.

3.2.2 Calcolo Readability

Il dataset creato precedentemente contenenti gli snippet di codice, è stato analizzato dal tool di Scalabrino et al. [2] calcolando le metriche di readability.

Questo tool calcola diversi tipi di metriche divisi sulla base di framework definiti da precedenti lavori in letteratura.

Modello Buse e Weimer

Il modello di Buse e Weimer[10] funziona come un addestratore binario, addestrato e testato su snippet di codice annotati manualmente. Le caratteristiche utilizzate da Buse e Weimer sono riportate nella Tabella 3.2. Il modello riesce a classificare i frammenti di codice come "leggibili" o "non leggibili" in oltre l'80% dei casi.

Tra le diverse caratteristiche, quelle risultate più utili per la distinzione tra il codice leggibile o non leggibile sono: il numero medio degli identificatori, la lunghezza media delle righe e il numero medio di parentesi.

Tabella 3.2: Caratteristiche utilizzate dal modello di leggibilità di Buse e Weimer

#	Caratteristiche	AVG	MAX
1	Lunghezza della riga (caratteri)	N/A	N/A
2	N. di identificatori	N/A	N/M
3	Indentazioni (spazi bianchi precedenti)	N/M	N/M
4	N. di parole chiavi	N/M	N/B
5	Lunghezza degli identificatori (caratteri)	N/B	N/M
6	N. di numeri	N/B	N/B
7	N. di parentesi	N/A	
8	N. di periodi	N/A	
9	N. di righe vuote	P/M	
10	N. di commenti	P/M	
11	N. di virgole	N/M	
12	N. di spazi	N/B	
13	N. di incarichi	N/B	
14	N. di filiale (se)	N/B	
15	N. di cicli (for, while)	N/B	
16	N. di operatori aritmetici	P/B	
17	N. di operatori a confronto	N/B	
18	N. di occorrenze e di qualsiasi carattere		N/M
19	N. di occorrenze di qualsiasi identificatore		N/B

***Note**: I valori per ogni metrica sono indicati solo con l'iniziale. Con N si intende "Negativamente correlata con l'alta leggibilità", con P si intende "Positivamente correlata con l'alta leggibilità". La seconda lettera dopo lo slash indica il potere predittivo(A="Alto", M="Medio", B="Basso")*

Modello Dorn

Dorn [11] introduce un modello per la valutazione della leggibilità del codice (vedi Tabella 3.3), organizzate in 4 categorie: visiva, spaziale, di allineamento e linguistica. L'obiettivo è catturare come il codice è letto dagli esseri umani sugli schermi. Le caratteristiche includono aspetti come evidenziazione della sintassi, standard di denominazione delle variabili e allineamento degli operatori. Il modello utilizza features visive, spaziali, di allineamento e basate sul linguaggio naturale.

- **Visive:** mirano a catturare aspetti visivi del codice, considerando la distribuzione dei caratteri per colore e regione, nonché la larghezza di banda media delle caratteristiche visuali, come l'indentazione, attraverso la trasformata di Fourier.
- **Spaziali:** coinvolgono la creazione di matrici per rappresentare le caratteristiche in base a determinati ruoli, come commenti.
- **Allineamento:** valutano la frequenza di allineamento di elementi sintattici.

Per la prima volta, il modello di Dorn introduce un fattore basato sul linguaggio naturale, contando il numero relativo di identificatori composti da parole presenti in un dizionario inglese.

Il modello di Dorn considera sia le proprietà strutturali del codice che il lessico del codice, differenziandosi da molti modelli esistenti che si concentrano principalmente sulle proprietà strutturali.

Modello New

Scalabrino et al. [2] hanno proprosto 7 proprietà testuali del codice sorgente che possono aiutare a caratterizzarne la leggibilità. Queste si basano sull'analisi sintattica del codice sorgente, cercando i termini presenti nel codice sorgente e nei commenti.

Comments and identifiers consistency Questa caratteristica mira ad analizzare la coerenza tra identificatori e commenti. In particolare, si calcola la Consistenza tra Commenti e Identificatori (CIC) misurando l'overlap tra i termini utilizzati in un commento di metodo e i termini utilizzati nel corpo del metodo.

Tabella 3.3: Caratteristiche definite da Dorn

#	Caratteristica	Direzione	Visivo	Spazio	Allineamento	Testuale
1	Lunghezza della linea	N	✔			
2	Lunghezza indentazione	P	✔			
3	Assegnamenti		✔			
4	Virgole		✔			
5	Confronti		✔			
6	Cicli		✔			
7	Parentesi		✔			
8	Periodi		✔			
9	Spazi	N	✔			
10	Commenti		✔	✔		
11	Parole chiavi	P	✔	✔		
12	Identificatori	N	✔	✔		✔
13	Numeri	P	✔	✔		
14	Operatori	N	✔	✔	✔	
15	Stringhe	P		✔		
16	Letterali			✔		
17	Espressioni				✔	

***Note**: La tabella mappa le categorie(esempio: percezione visiva, percezione spaziale, allineamento o analisi del linguaggio naturale) alle singole caratteristiche. La colonna direzione indica se è correlato positivamente(P) oppure negativamente (N).*

Identifier terms in dictionary Nella seguente caratteristica si ipotizzi che maggiore sia il numero di termini negli identificatori del codice sorgente che sono anche presenti in un dizionario, maggiore sarà la leggibilità del codice. Così, data una riga di codice l, si misura la feature Identifier terms in dictionary (ITID)

Narrow meaning identifiers Si ipotizzoa che un codice leggibile dovrebbe contenere più iponimi, cioè termini con un significato specifico, rispetto a iperonimi, cioè termini generici che potrebbero essere fuorvianti. Quindi, data una riga di codice l, si misura la feature Narrow meaning identifiers (NMI).

Comments readability Questo testo affronta l'importanza dei commenti nel codice e come la qualità di essi possa influenzare la comprensione del codice stesso. Sebbene molti commenti possano sicuramente aiutare a comprendere il codice, potrebbero avere l'effetto opposto se la loro qualità è bassa. Per affrontare questo problema, è stata introdotta una feature che calcola la leggibilità dei commenti (CR) utilizzando l'indice Flesch-Kincaid, comunemente usato per valutare la leggibilità dei testi in linguaggio naturale. Tale indice considera tre tipi di elementi: parole, sillabe e frasi.

Number of meanings In questa caratteristica si misura il numero di significati (NM), o il livello di polisemia, di uno snippet. La polisemia è la facoltà di una parola di assumere più significati. Per ogni termine nel codice sorgente, si misura il numero di significati derivati da WordNet.

Textual coherence La coerenza testuale dello snippet può essere utilizzata per stimare il numero di "concetti" implementati da uno snippet di codice sorgente. Nello specifico, vengono considerati i blocchi sintattici di uno snippet come documenti, creando un albero sintattico astratto (AST) per rilevare i blocchi sintattici. Viene quindi calcolata la sovrapposizione di vocabolario tra tutte le possibili coppie di blocchi sintattici distinti. La coerenza testuale (TC) di uno snippet può essere calcolata come la sovrapposizione massima, minima o media tra ciascuna coppia di blocchi sintattici.

Number of concepts La misura "Number of concepts" valuta il numero di concetti implementati in uno snippet a livello di riga. La Coerenza Testuale cerca di

catturare il numero di argomenti implementati in uno snippet a livello di blocco, ma può avere limitazioni quando ci sono pochi blocchi sintattici. Se uno snippet contiene un solo blocco, questa caratteristica non è calcolabile. Inoltre, la coerenza testuale è una caratteristica a grana grossa e funziona con l'assunzione che i blocchi sintattici siano autoconsistenti. Per questo motivo, viene introdotta una misurazione che cattura direttamente il numero di concetti implementati in uno snippet a livello di riga. Questa misurazione può essere calcolata anche su snippet che potrebbero non essere sintatticamente corretti.

3.2.3 Individuazione Linguistic Antipatterns

Allo stesso modo, gli snippet di codice sono stati analizzati dal tool IDEAL di Arnaoudova et al. [1], analizzando i Linguistic Antipatterns contenenti all'interno di essi. I Linguistic Antipattern sono stati suddivisi in categorie che vanno dalla "A" alla "G., riportati nelle Tabelle 3.4 e 3.5.

Tabella 3.4: Linguistic Antipatter Parte 1

Id	Pattern	Detection Strategy
A.1	"Get more than accessor"	Il nome inizia con 'get', lo specificatore di accesso è pubblico/protetto, il nome contiene il nome di un attributo, il tipo restituito è lo stesso del tipo di attributo e il corpo contiene istruzioni condizionali
A.2	"Is" returns more than a Boolean	Il nome inizia con un termine relativo a predicato/affermazione e il tipo restituito non è booleano
A.3	"Set" method returns	l nome inizia con 'set' e il tipo restituito non è void
A.4	Expecting but not getting single instance	L'ultimo termine nel nome è singolare e il nome non contiene termini che sono un tipo di raccolta e il tipo restituito è una raccolta
B.1	Not implemented condition	Il nome contiene termini correlati condizionali nel nome o nel commento e il corpo non contiene istruzioni condizi
B.2	Validation method does not confirm	Il nome inizia con un termine correlato alla convalida, non ha un tipo restituito e non genera un'eccezione
B.3	"Get" method does not return	Il nome inizia con un termine correlato a 'get' e il tipo restituito è void
B.4	Not answered question	Il nome inizia con un termine relativo a predicato/affermazione e il tipo restituito è void
B.5	Transform method does not return	Il nome inizia con o un termine interno contiene un termine di trasformazione e il tipo restituito è void
B.6	Expecting but not getting a collection	Il nome inizia con un termine correlato a 'get', il nome contiene un termine che è plurale o un tipo di raccolta e il tipo restituito non è un tipo basato su raccolta

Tabella 3.5: Linguistic Antipattern Parte 2

Id	Pattern	Detection Strategy
C.1	Method name and return type are opposite	Esiste una relazione di antonimi tra i termini in un nome di identificatore e il tipo di dati
C.2	Method signature and comment are opposite	Esiste una relazione di antonimo tra i termini nel nome di un identificatore o nel tipo di dati e nei commenti
D.1	Says one but contains many	L'ultimo termine nel nome è singolare e il tipo di dati è una raccolta
D.2	Name suggests Boolean but type does not	Il termine iniziale deve essere correlato a predicato/affermazione e il tipo di dati non è booleano
E.1	Says many but contains one	L'ultimo termine nel nome è plurale e il tipo di dati non è una raccolta
F.1	Attribute name and type are opposite	Esiste una relazione di antonimi tra i termini in un nome di identificatore e il tipo di dati
F.2	Attribute signature and comment are opposite	Esiste una relazione di antonimo tra i termini nel nome di un identificatore o nel tipo di dati e nei commenti
G.1	Name contains only special characters	Il nome dell'identificatore è composto solo da caratteri non alfanumerici
G.2	Redundant use of "test" in method name	Il nome inizia con il termine 'test'

CAPITOLO 4

Analisi dei risultati

Nel presente capitolo, verrà condotta un'analisi dei risultati ottenuti attraverso la creazione del dataset, del tool di scalabrino et al. [2] e del tool IDEAL [1].

4.1 Risultati Dataset

Questa sezione si concentra sull'ampliamento delle informazioni per la creazione del Dataset. Le domande del questionario inviato agli studenti sono riportate nella Tabella 3.1 del capitolo precedente. La domanda 5 presente in questa tabella è stata fondamentale per l'avvio del processo di creazione del dataset.

Nella Tabella 4.1, sono elencate le risposte correlate a questa domanda, che sono state successivamente utilizzate per generare snippet di codice mediante ChatGPT. Di seguito, sono forniti alcuni esempi di snippet di codice generati da ChatGPT.

Il dataset è disponibile nell'appendice online [1]

[1] https://github.com/Annamariabasile/Comprehension$_a$*nalysis*

Tabella 4.1: Risposte questionario

#	Risposte
1	Come posso usare in java la funzione collections.sort con una lambda expression
2	Come faccio ad ordinare una hashmap in base ai valore in java
3	Come faccio ad usare il k fold cross validation di scikit learn
4	Come faccio a calcolare la precision del mio modello di machine learning
5	Come implementare push notifications in java
6	Come implementare adapter pattern in java
7	Come fare richieste http in java
8	Come tradurre responde json in java
9	Come usare i mock mockito java spring
10	Come testare junit
11	Come implementare diverse parti del codice: token e notifiche(nei token, codici riguardanti i cors policy)
12	How to send push notification java spring?
13	How to implement token jwt java spring 2023?
14	Come passare le misurazioni da Java a JSON", "Come rappresentare questi JSON in oggetto
15	Come implementare adapter pattern in java

```java
public class Measurement {
    private String name;
    private double value;
    private String unit;
    // Costruttore, getter e setter
    // Esempio di costruttore:
    public Measurement(String name, double value, String unit) {
        this.name = name;
        this.value = value;
        this.unit = unit;
    }
}
```

4.2 Analisi Readability

RQ$_1$. *Il codice generato da ChatGPT presenta problemi di leggibilità?*

L'obiettivo di questa RQ è di analizzare gli snippet di codice generato da ChatGPT verificando se essi presentano problemi di leggibilità. Come riportato nel capitolo 3.2, gli snippet di codice sono stati analizati dal tool di Scalabrino et al. [2].

4.2.1 Risultati Modello New

Nelle seguenti Figure 4.1, 4.2, 4.3 sono riportati i risultati del modello New sottoforma di grafici. Si può notare per quanto riguarda la metrica **New Commented words AVG** nella Figura 4.1 che per la maggior parte dei codici è zero, quindi le parole commentate all'interno degli snippet di codici generati da ChatGPT sono bassi, il che può risultare un problema di leggibilità.

Inoltre, si puo notare come nella Figura 4.3 la metrica **New Comments readability** per almeno la metà degli snippet di codici analizzati risulta essere zero, quindi i commenti generati da ChatGPT spesso sono illegibili. Queste informazioni evidenziano che nei codici generati i commenti possono portare ad un problema di leggibilità.

Per quanto riguarda la metrica **New Expression complexity AVG**, nella Figura 4.1, un numero sinificativo di snippet presentano un valore sopra la media.

È interessante notare che nella Figura 4.2 il valore la metrica **New Number of senses AVG**, è abbastanza alto per la maggior parte dei codici. Questa metrica indica il livello di polisemia di uno snippet, ovvero la facoltà di una parola di assumere più significati. Quindi ciò significa che le parole usati negli snippet generati da ChatGPT possono risultare ambigue.

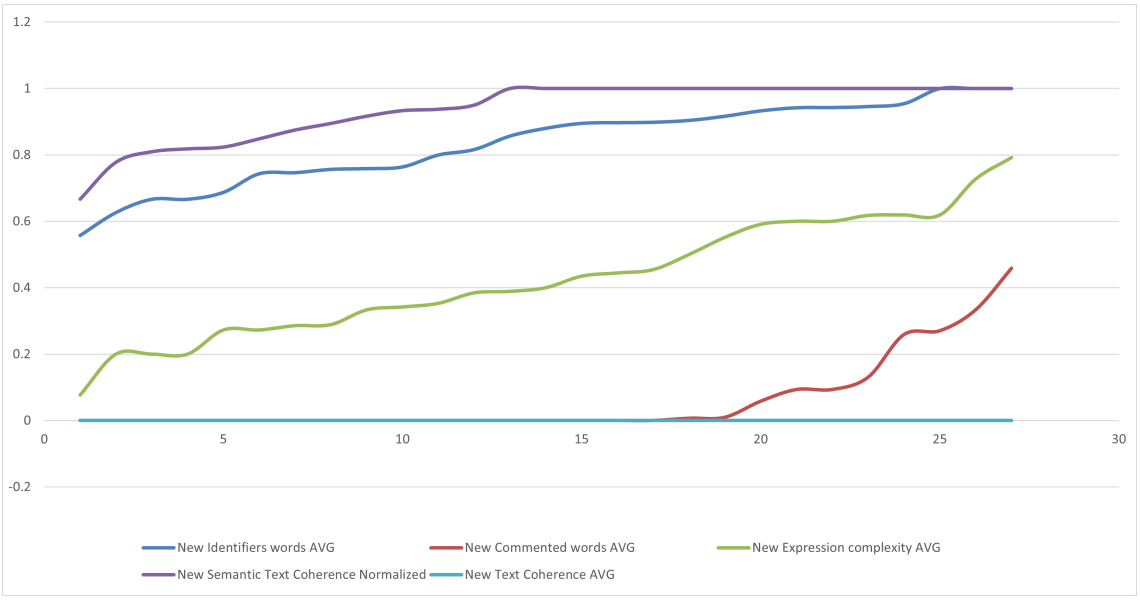

Figura 4.1: Grafico Modello New Parte 1.

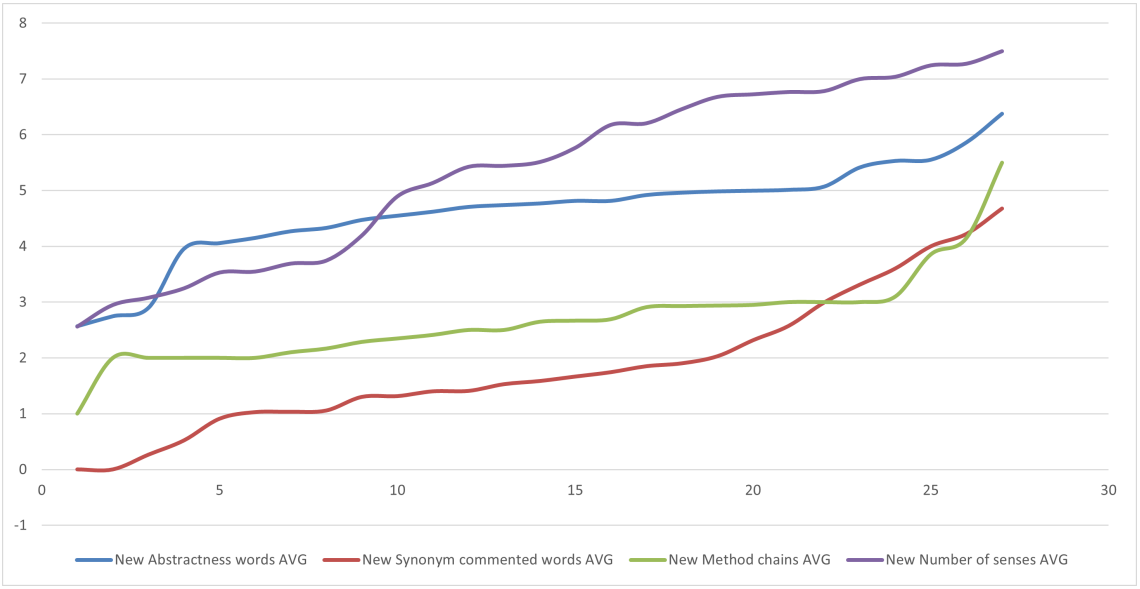

Figura 4.2: Grafico Modello New Parte 2.

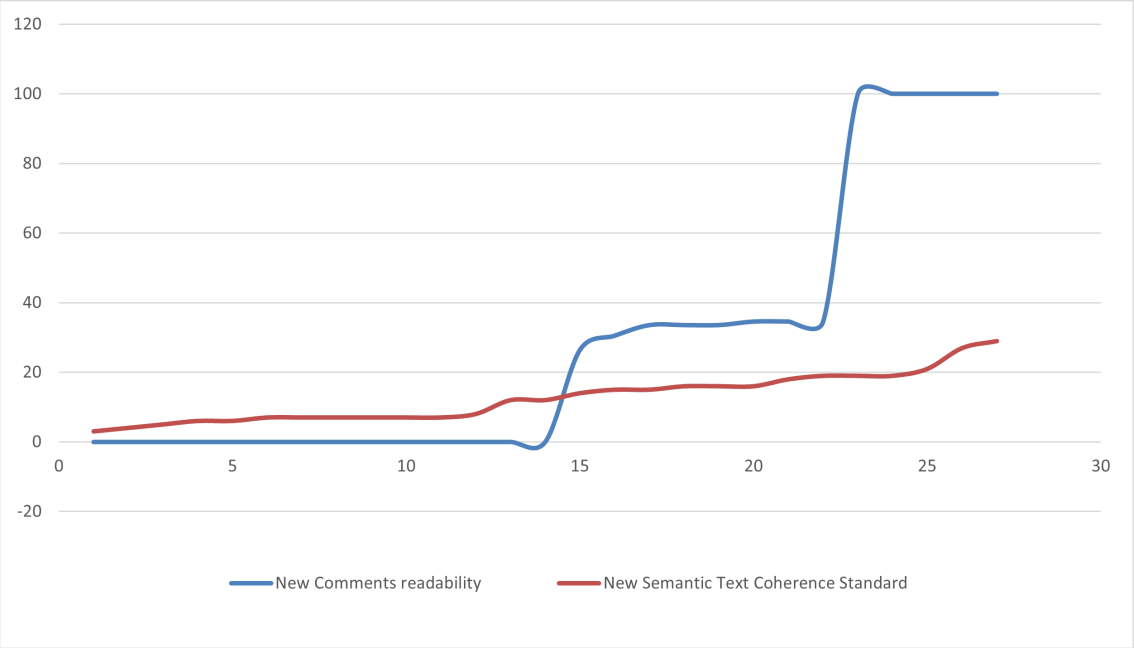

Figura 4.3: Grafico Modello New Parte 3.

4.2.2 Risultati Modello Buse e Weiner

Nelle seguenti Figure 4.4, 4.5, 4.6 sono riportati i risultati del modello di Buse e Weiner sottoforma di grafici. Come mostrato nella Tabella 3.2, si può notare che ci sono metriche che hanno potere predittivo alto e metriche con potere predittivo basso. Notiamo che le metriche con potere predittivo alto sono correlate negativamenete con l'alta leggibilità, quindi più è alto il valore più il codice e difficile da leggere. Analizzando i grafici si può notare che:

- i valori della metrica **BW Avg line length**, in Figura 4.6, sono vicini alla media per tutti gli snippet di codice generati da ChatGPT, quindi per questa metrica non risultano avere problemi di leggibilità;
- i valori della metrica **BW Avg number of identifiers**, in Figura 4.4, non risultano essere vicini alla media per gli snippet, quindi il codice generato da ChatGPT può presentare problemi di leggibilità a causa di un numero troppo elevato di identificatori diversi;

- i valori della metrica **BW Avg parenthesis**, in Figura 4.4, risultano essere omogenei se non per qualche eccezione, quindi i codici generati da ChatGPT per questa metrica non presentano problemi di leggibilità;
- i valori della metrica **BW Avg periods**, in Figura 4.4, non risultano cosi omogenei per alcuni codici, quindi alcuni di essi possono presentarsi problemi di leggibilità.

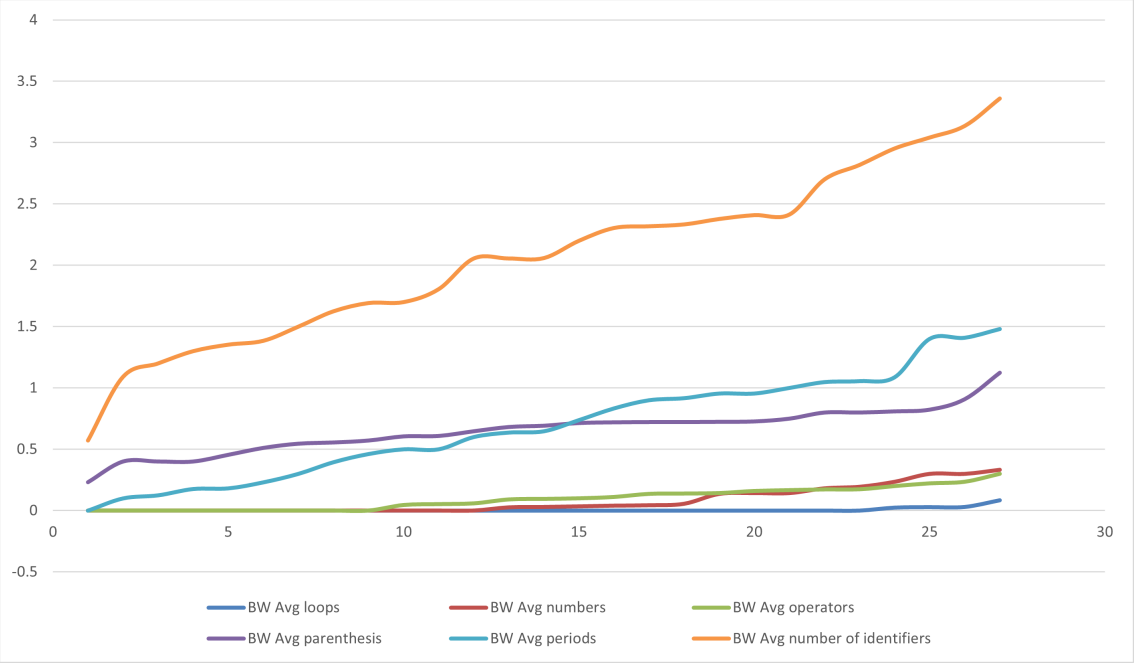

Figura 4.4: Grafico Modello BW Parte 1.

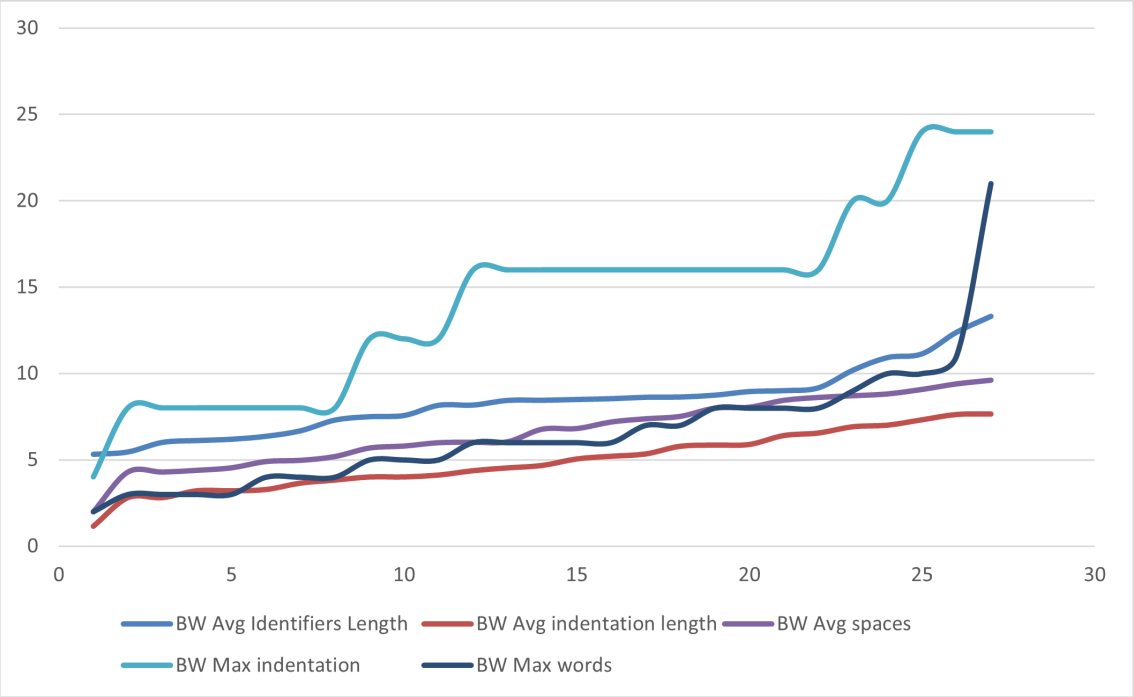

Figura 4.5: Grafico Modello BW Parte 2.

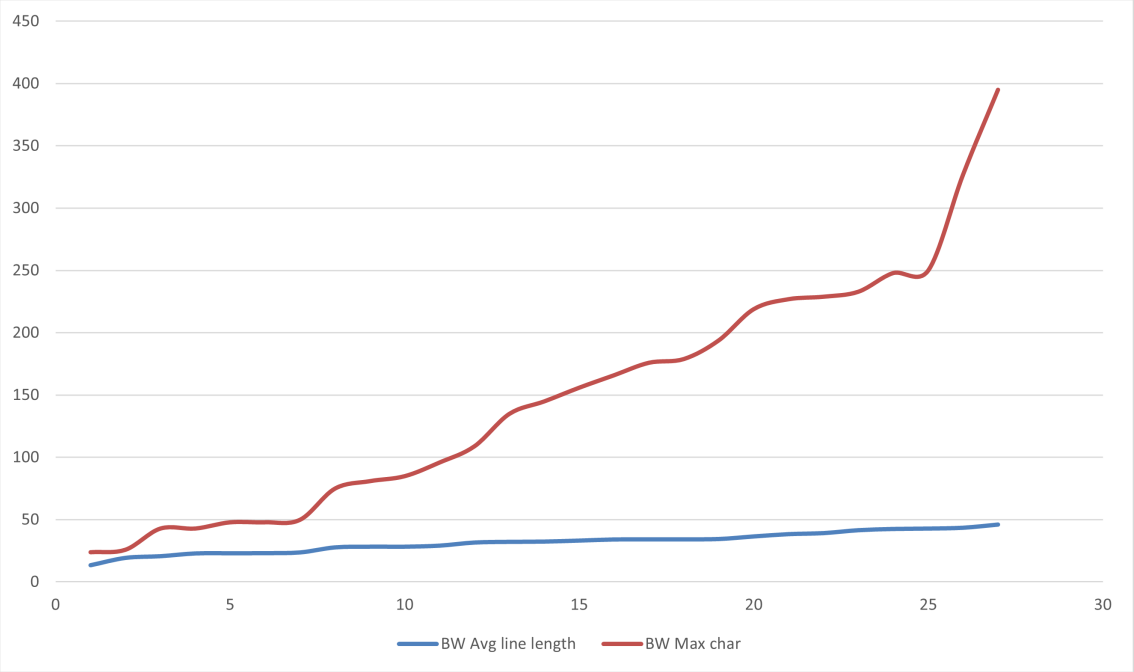

Figura 4.6: Grafico Modello BW Parte 3.

4.2.3 Risultati Modello Dorn

Nelle seguenti Figure 4.7, 4.8, 4.9, 4.10, 4.11 sono riportati i risultati del modello di Dorn sottoforma di grafici. Anche per il modello Dorn le metriche è possibile osservare diverse correlazioni con l'alta leggibilità [11].

Come mostrato nella Tabella 3.3 le metriche sono suddivise per features: visive, spaziali, di allineamento e testuali. Nei grafici mostrati in questa sezione, oltre a queste features vengono riportate delle metriche DFT, ovvero dei valori ottenuti trasformando la frequenza delle varie informazioni in segnali e prendendo la trasformata discreta di Fourier (DFT) del segnale.

L'analisi delle Figure 4.7, 4.9, focalizzate sulle metriche riguardarti i commenti, ovvero **Dorn DFT Comments** e **Dorn Visual Y Comments**, rivela una proporzione elevata tra la presenza dei commenti e le righe di codice. Questo risultato suggerisce una valutazione positiva in termini di leggibilità.

Delle figure 4.7, 4.8, emerge che anche per quanto riguarda le metriche **Dorn DTF Identifiers** e **Dorn Visual Y Keywords**, la presenza degli identificatori e di parole chiavi è significativamente alta. Questo indica un'eccessiva presenza di identificatori e parole chiavi utilizzati come nomi di variabili, generando potenzialmente confusione per chi utilizza il codice compromettendo la leggibilità.

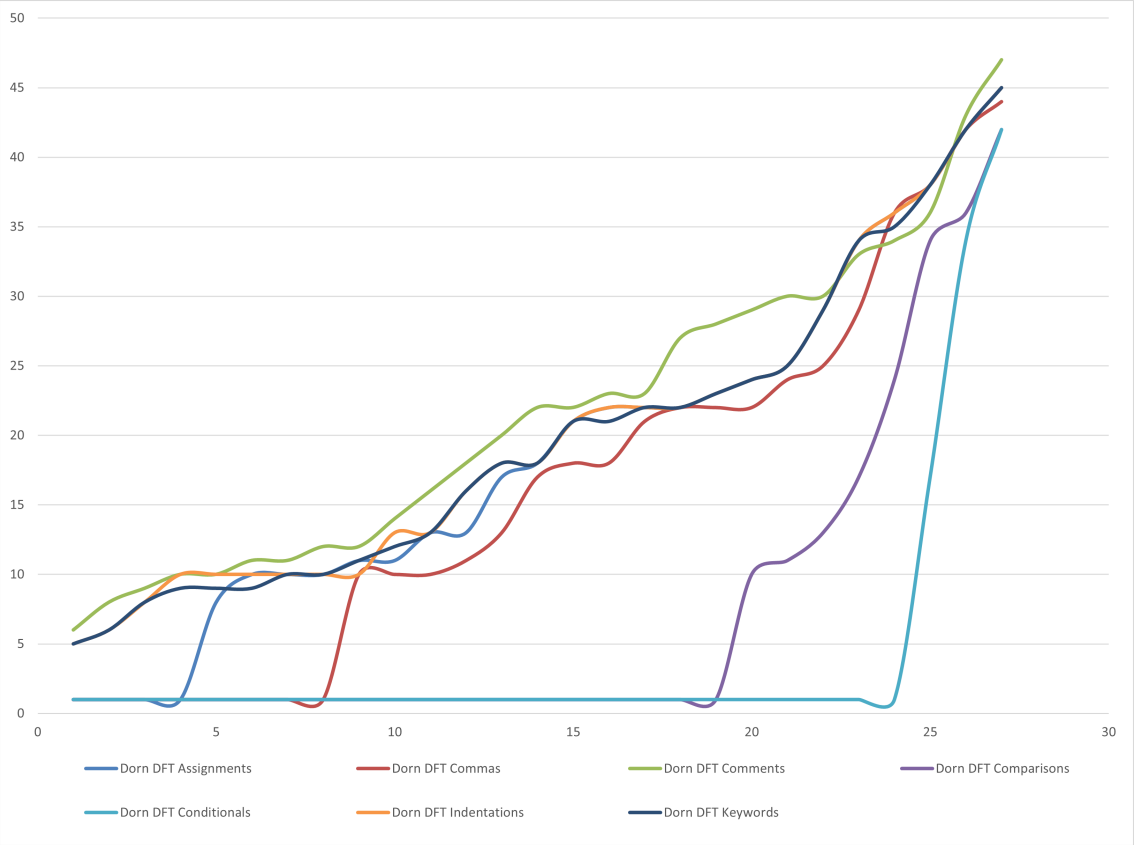

Figura 4.7: Grafico Modello Dorn Parte 1.

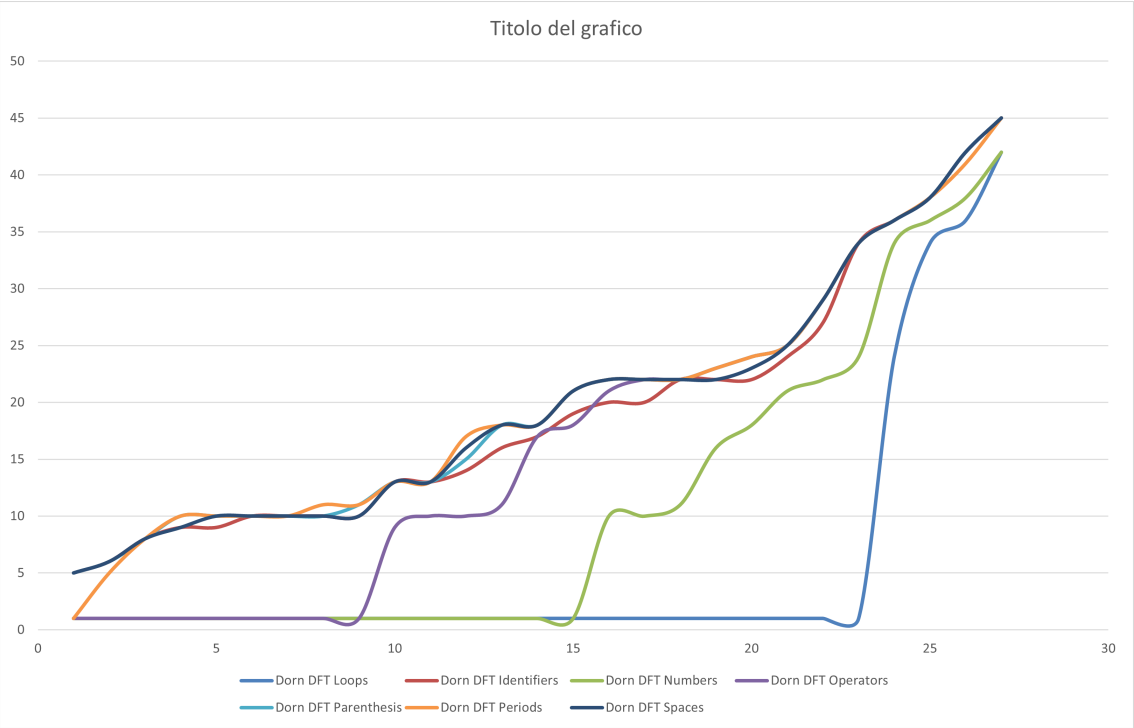

Figura 4.8: Grafico Modello Dorn Parte 2.

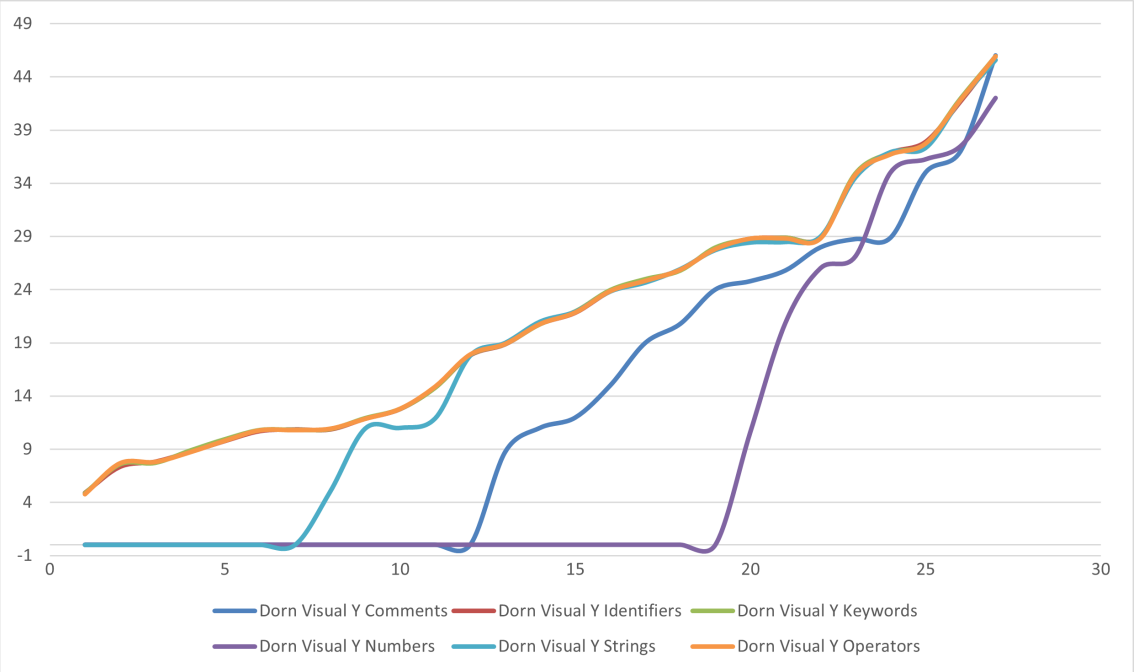

Figura 4.9: Grafico Modello Dorn Parte 3.

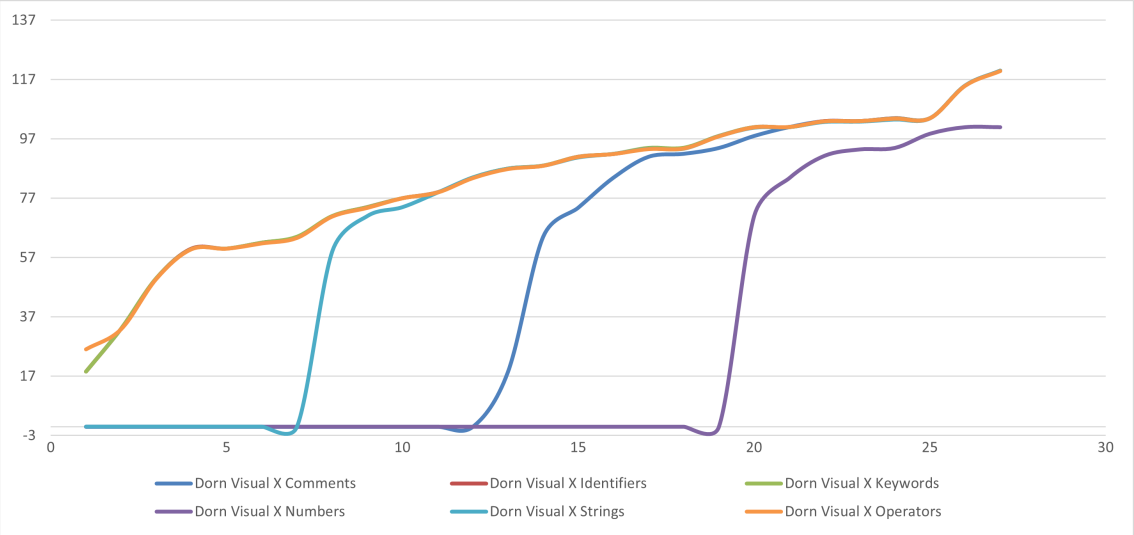

Figura 4.10: Grafico Modello Dorn Parte 4.

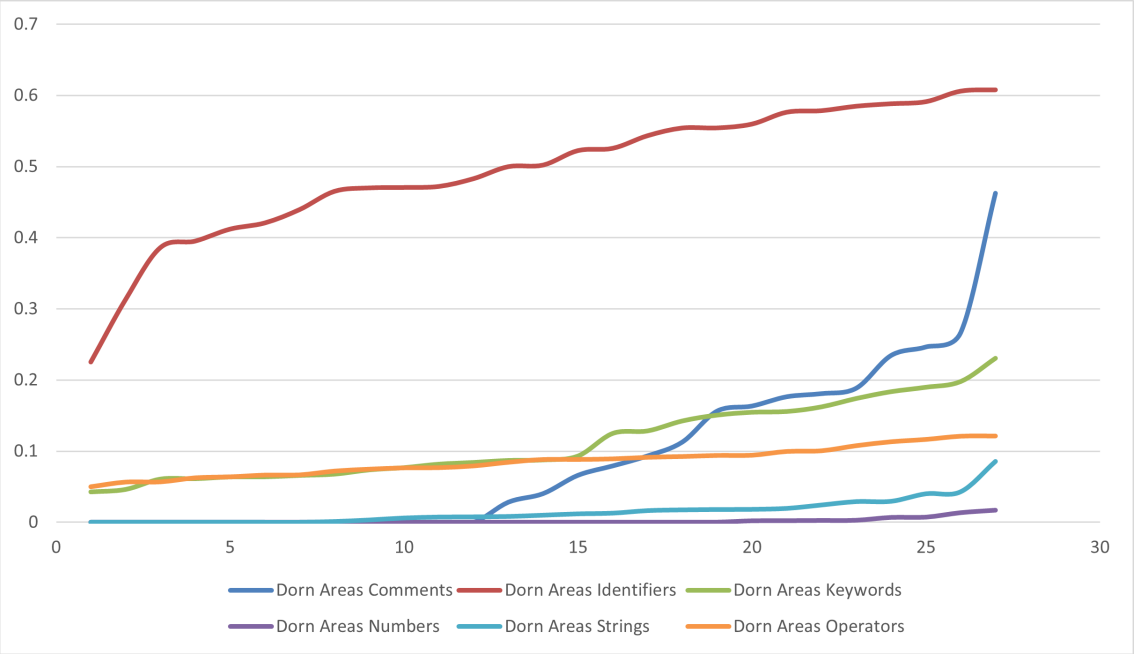

Figura 4.11: Grafico Modello Dorn Parte 5.

4.3 Analisi Linguistic Antipatterns

RQ$_2$. *Il codice generato da ChatGPT contiene Antipatterns linguistici?*

L'obiettivo di questa RQ è di è analizzare gli snippet di codice generati da ChatGpt al fine di verificare la presenza dei Linguistic Antipatterns. Come riportato nel capitolo della metodologia 3.2, gli snippet di codice presenti nel datset costruito sono stati analizzati dal tool IDEAL [1]. Nella presente sezione verranno riportati i risultati divisi per modello di leggibilità.

Il diagramma, visibile nella Figura 4.12, fornisce una rappresentazione visiva dettagliata dei Linguistic Antipattern identificati all'interno dei 27 snippet di codice analizzati. Risulta evidente che alcuni di questi antipattern siano prevalenti rispetto ad altri nel contesto degli snippet generati da ChatGPT. I dettagli specifici relativi a ciascun antipattern sono disponibili nella Tabella 3.4 e 3.5 del capitolo precedente, offrendo una panoramica esaustiva delle caratteristiche individuate in ciascun codice. Questa rappresentazione grafica è fondamentale per capire in modo approfondito la presenza e la distribuzione di tali antipattern nel contesto particolare dello studio.

👉 **Risposta alla RQ$_2$.** Il tool IDEAL ha mostrato come su 27 snippet di codice, 11 presentano dei Linguistic Antipattern. 2 di essi ne presententano più di uno

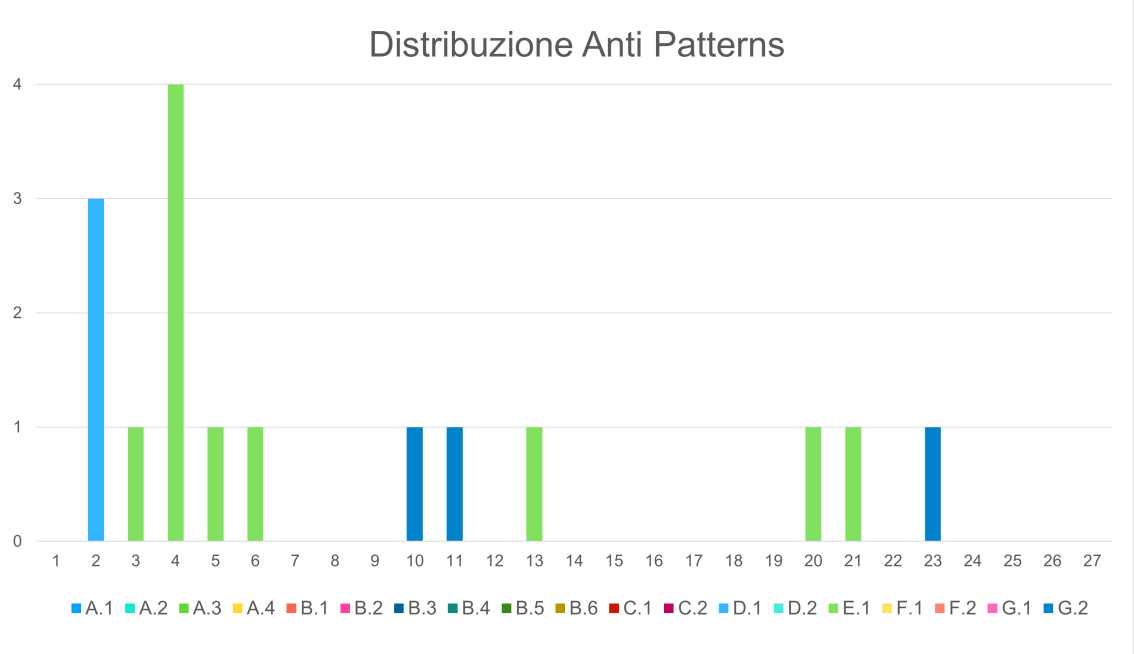

Figura 4.12: Risultati Linguistic Antipattern.

4.4 Discussioni

Dai risultati dei grafici che mostrano le metriche del modello di readability New, nella Figura 4.1, si riscontra una carenza di commenti per parola, ovvero la proporzione tra codice e relativi commenti. Parallelamente, nella Figura 4.3, si può notare una bassa leggibilità di tali commenti. Ciò significa che il codice prodotto è poco leggibile dal punto di vista dei commenti generati.

È interessante osservare che, secondo le metriche di Dorn, la valutazione dei commenti per riga risulta elevata, suggerendo una buona leggibilità del codice. Questo dato, tuttavia, risulta contraddittorio, poiché le specifiche metriche del modello New indicano una mancanza generale di leggibilità nei commenti generati da ChatGPT, nonostante la loro frequente presenza. Questo fenomeno solleva interrogativi sulla coerenza e l'effettiva chiarezza del testo commentato da ChatGPT, evidenziando una disaccordo interessante tra le diverse valutazioni metriche.

Nonostante ciò sono stati riscontrati altri problemi di leggibilità come nella Figura 4.2, per quanto riguarda la metrica **New Number of senses AVG**, in cui gli snippet di codice possono risultare ambigui per l'alto livello di polisemia, ovvero la falcoltà di una parola di assumere più significati.

Anche per le metriche del modello Dorn e del modello BW, nei Grafici 4.7 e 4.4, risultano esserci un'eccessiva presenza di identificatori diversi, generando confusione per chi utilizza il codice.

Analizzando il Grafico 4.12, si può notare come su 27 snippet di codici 11 di questi presentano Antipattern linguistici. Gli antipattern presenti sono: D.1, E.1 e G.2 descritti nelle Tabelle 3.4 3.5 del capitolo precedente. Inoltre si può notare che 2 degli 11 codici presentano piu volte lo stesso Antipattern D.1 e E.1. Si può dire quindi che il codice generato da ChatGPT su 27 codici generati da ChatGPT che circa il 40,74% presenta dei Linguistic Antipattern.

Tra i linguistic antipattern rilevati possiamo notare che ChatGPT non associa bene i plurali o i singolari ai nomi delle variabili che sono liste o che non sono liste.

Un altro antipattern trovato fa riferimento all'utilizzo ridondante della parola test, causando possibili incomprensioni in quanto non sono effettivamente metodi di test.

C'è da notare il fatto che nonostante quasi la metà dei codici generati presenta antipattern, questi sono sempre gli stessi, ovvero D.1, E.1, G.2, il che è una nota positiva in quanto ChatGPT ha problemi con pochi antipattern.

CAPITOLO 5

Conclusioni

Questo lavoro di tesi si propone di approfondire l'analisi della qualità del codice generato da ChatGPT, focalizzandosi principalmente sulla sua comprensibilità. Per rendere questa valutazione più accurata e significativa, è stato ideato e implementato un questionario somministrato agli studenti iscritti ai corsi di laurea triennale o magistrale in informatica.

L'obiettivo di coinvolgere gli studenti mira ad ottenere prompt autentici che riflettano le reali esigenze e prospettive degli utenti finali. Successivamente, questi prompt, sono stati sottoposti a ChatGPT, consentendo la generazione di un dataset di snippet di codice.

Il dataset generato con gli snippet di codice è stato sottoposto a un'analisi mediante l'impiego di strumenti specifici. In particolare, è stato adoperato il tool proposto da Scalabrino et al. [2], il quale è progettato per esaminare la leggibilità del codice generato. Parallelamente, è stato impiegato il tool IDEAL [1] al fine di identificare la presenza di antipattern linguistici negli snippet creati da ChatGPT.

Dopo l'analisi degli snippet di codice, sono emersi alcuni problemi di leggibilità, tra cui una carenza di commenti e la difficoltà nel comprenderli. Inoltre, è stata notata un'eccessiva presenza di identificatori e parole chiave, il che potrebbe causare confusione agli utenti che utilizzano il codice.

Oltre ai problemi di leggibilità, sono stati individuati alcuni Linguistic Antipatter, in particolare sono emersi i seguenti: D.1, E.1 e G.2. La presenza di tali Antipattern determina una nota positiva poiché nel codice generato da ChatGPT non vengono indivuati altre tipologie di Antipatter Linguistici se non quelli individuati.

Considerando che il dataset di prompt sottoposto a ChatGPT è stato fornito dagli studenti, un potenziale sviluppo futuro potrebbe riguardare l'ottimizzazione delle modalità di porre le domande a ChatGPT. Una possibile strategia potrebbe essere quella di formulare domande più dettagliate, al fine di valutare se ciò possa influire sulla leggibilità del risultato generato o sulla presenza di antipattern linguistici. Esaminare dettagliatamente questo aspetto potrebbe offrire conoscenze utili per migliorare l'interazione con ChatGPT e ottimizzare la qualità delle risposte ottenute.

Un altro possibile sviluppo futuro potrebbe essere di effettuare lo stesso studio ma con altri attributi che garantiscono la code comprehension.

Bibliografia

[1] A. Peruma, V. Arnaoudova, and C. D. Newman, "Ideal: An open-source identifier name appraisal tool," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 599–603. (Citato alle pagine 2, 9, 16, 19, 29 e 32)

[2] S. Scalabrino, M. Linares-Vásquez, R. Oliveto, and D. Poshyvanyk, "A comprehensive model for code readability," *Journal of Software: Evolution and Process*, vol. 30, no. 6, p. e1958, 2018, e1958 smr.1958. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1958 (Citato alle pagine 2, 9, 11, 13, 19, 21 e 32)

[3] M. L. Siddiq, S. H. Majumder, M. R. Mim, S. Jajodia, and J. C. S. Santos, "An empirical study of code smells in transformer-based code generation techniques," in *Proceedings of the 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Oct 2022. (Citato a pagina 5)

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. (Citato a pagina 5)

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. (Citato a pagina 6)

[6] 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. (Citato a pagina 6)

[7] M.-A. Storey, "Theories, methods and tools in program comprehension: past, present and future," in *13th International Workshop on Program Comprehension (IWPC'05)*, 2005, pp. 181–191. (Citato a pagina 7)

[8] F. Palomba, A. De Lucia, G. Bavota, and R. Oliveto, "Chapter four - anti-pattern detection: Methods, challenges, and open issues," ser. Advances in Computers, A. Memon, Ed. Elsevier, 2014, vol. 95, pp. 201–238. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128001608000048 (Citato a pagina 9)

[9] V. Arnaoudova, M. Di Penta, and G. Antoniol, "Linguistic antipatterns: what they are and how developers perceive them," *Empirical Software Engineering*, vol. 21, 01 2015. (Citato a pagina 9)

[10] R. P. Buse and W. R. Weimer, "Learning a metric for code readability," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 546–558, 2010. (Citato a pagina 11)

[11] J. Dorn, "A general software readability model," *MCS Thesis available from (http://www. cs. virginia. edu/weimer/students/dorn-mcs-paper. pdf)*, vol. 5, pp. 11–14, 2012. (Citato alle pagine 13 e 26)