UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

CoReviewer:

Sviluppo di un Tool di Summarization per attività di Code Review

RELATORE

Prof. Fabio Palomba

Dott.ssa Giulia Sellitto

Università degli Studi di Salerno

CANDIDATO

Alessandro Zoccola

Matricola: 0512110912

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

"Perché ho ancora più sogni che cassetti"

Abstract

Lo sviluppo di codice sorgente è un processo che deve essere adeguatamente documentato durante tutto il ciclo di vita del software. Durante la scrittura del codice sorgente, questo deve essere sottoposto a processi lunghi di revisione da parte del reviewer; Tali processi si chiamano Code Review.
L'intelligenza artificiale è entrata oramai nelle nostre vite e utilizzarla in maniera efficace può migliorare e velocizzare il nostro lavoro. Modelli linguistici come LLaMA 2, BARD o GPT sono in costante crescita e sono stati appositamente pensati per task come la summarization, o altri task di NLP e linguistica. Lo scopo di questo lavoro di tesi è quello di andare a semplificare il processo di Code Review, in modo da diminuire costi economici e temporali e di migliorare la qualità delle review proposte dalle aziende di sviluppo software. CoReviewer, il tool che abbiamo sviluppato, è un'estensione per web browser che consente al reviewer di ottenere un breve summary che riassuma le modifiche effettuate alla code base dall'autore durante l'ultimo commit della repository e quindi fargli comprendere a pieno tutte le differenza tra la precedente e l'attuale versione della code base.
L'estensione utilizza il modello LLaMA 2 per la generazione del summary, per il Front-End viene utilizzata la libreria React JS, mentre il Back-End è stato sviluppato con il framework Flask.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

Lo sviluppo di software è un processo molto lungo e dettagliato che non termina con il rilascio del prodotto. Una delle fasi fondamentali è quella della manutenzione del codice sorgente e realizzare codice manutenibile è alla base del processo di sviluppo. Proprio per questa motivazione durante l'intero processo gli sviluppatori discutono le modifiche che effettuano alla code base durante le attività di code review. Il reviewer farà una revisione delle modifiche che l'autore gli presenterà, ne valuterà gli effetti e deciderà se queste possono essere integrate o meno nella repository principale. Il processo di revisione del codice può spesso risultare estremamente prolungato e complesso se non svolto nella giusta maniera, ciò causerebbe inefficienze sia dal punto di vista economico che temporale. Tali ostacoli sono principalmente associati alla fase di comprensione delle modifiche apportate dall'autore nella code base.

1.2 Motivazione ed Obiettivi

L'obiettivo di questa tesi è di realizzare un tool che permetta di velocizzare il processo di code review. L'obiettivo viene raggiunto con lo sviluppo di un tool che consenta al reviewer di comprendere, mediante la generazione di un breve summary, le modifiche che sono state fatte alla code base, evitando incomprensioni che potrebbero venire fuori dalla comunicazione tra autore o reviewer, oppure dall'intuito del reviewer leggendo il solo messaggio di commit allegato dall'autore.

1.3 Risultati Ottenuti

I risultati che abbiamo ottenuto possono essere ritenuti soddisfacenti, seppur con un ampio margine di miglioramento. Si ritiene che il tool sviluppato, con le possibili migliorie che sono state indicate nel capitolo dedicato agli sviluppi futuri, possa risultare ancor più determinante nella automatizzazione del processo di code review e che il suo campo di azione possa essere esteso a ben più che un solo tool di summarization di modifiche ad una code base, ma anche ad attività di refactoring o comunque legate alla manutenzione del codice sorgente.

1.4 Struttura della Tesi

La tesi è strutturata in 5 capitoli i cui concetti espressi vengono spiegati brevemente di seguito:

- Nel capitolo 2 viene sottolineata l'importanza della manutenibilità del codice sorgente e viene introdotto il concetto di code review e come queste vengono effettuate. Si parlerà dunque di summarization di codice sorgente, uno dei tanti task che si affrontano con il Natural Language Processing.

- Nel capitolo 3 il focus è sull'implementazione del tool, si discuterà in particolare delle scelte implementative e di design che hanno portato alla realizzazione di un'estensione per web browser. Verranno discusse le tecnologie utilizzate e le scelte che hanno portato a tali scelte.

- Nel capitolo 4 sono presenti le considerazioni sul modello linguistico utilizzato per la realizzazione del summary, in particolare si parlerà della fase di prompt engineering che ha permesso di sottolineare pregi e difetti di LLaMA 2.
- Nell'ultimo capitolo, invece, sono presenti le conclusioni rispetto al lavoro di tesi prodotto ed eventuali sviluppi futuri per ampliare e migliorare il set di applicazioni in cui sarà possibile usare l'estensione CoReviewer.

CAPITOLO 2

Stato dell'arte

2.1 Code Review

2.1.1 Manutenzione del codice sorgente

La fase di manutenzione del codice sorgente è una delle fasi del ciclo di vita del software. [1] Contrariamente a quanto si possa pensare è una delle fasi più importanti, nonché la più costosa. Durante questa fase vengono svolte diverse attività legate alla correzione e alla modifica di funzionalità che il software offre. Tutto ciò mira a migliorare la qualità del prodotto che si sta realizzando.
È necessario riconoscere che alla base di un buon software c'è sempre una buona documentazione, ovvero una descrizione di che descrive le funzionalità di un software, come è stato strutturato e come è stato implementato. Risulta fondamentale lo sviluppo degli sviluppatori di codice documentato e manutenibile. Uno studio effettuato da Lakhotia [2] riporta quanto sia difficile per un programmatore comprendere il codice scritto da altri sviluppatori, in particolare quando non si ha molta familiarità con il linguaggio o se per la prima volta ci si interfaccia con tale code base. La documentazione di ciascun artefatto è fondamentale proprio perché il codice sorgente non verrà letto solo dall'autore stesso, ma anche da altri sviluppatori che

potrebbero dover lavorare con tale funzionalità.
Sulla base di quanto detto, possiamo immaginare quanto sia importante accompagnare il codice con una spiegazione più o meno breve delle sue caratteristiche.

2.1.2 Attività di code review

La code review fa parte delle attività che, durante la fase di manutenzione, garantiscono una qualità del codice sorgente di alto livello e che la sua manutenibilità venga garantita durante tutto il processo di sviluppo software. Lo sviluppo del software non termina con il suo rilascio, bensì il prodotto è in continuo aggiornamento nelle fasi di manutenzione, dove è ancor più centrale il ruolo di una buona documentazione.
È impensabile che il codice sorgente venga revisionato solo al termine del suo sviluppo, un approccio del genere potrebbe causare ingenti danni sia economici che temporali alle aziende, proprio per questo durante il ciclo di vita del software vengono effettuate delle revisioni, le code review.
Una code review consiste nella lettura e comprensione del codice sorgente al fine di valutare la qualità degli artefatti sottoposti a revisione, trovare quanti più problemi possibili ed eventualmente proporre delle soluzioni migliori rispetto a quelle proposte prima che il codice venga messo in produzione.[3]
La CR può essere effettuata da uno o più revisori a seconda della difficoltà del task da revisionare. Alla revisione dovrebbe partecipare anche l'autore del codice sorgente.
La code review può avvenire in diversi momenti, generalmente viene effettuata prima che una nuova porzione di codice venga integrata nel repository principale, sebbene possa essere richiesta in qualsiasi momento dall'autore del codice.
Bisogna tenere a mente che il programmatore dovrà richiedere la code review per modifiche singole o strettamente correlate per le motivazioni sopraelencate, sono quindi raccomandati frequenti commit piccoli rispetto a un unico commit con modifiche a porzioni diverse del codice. [4]
Il revisore, oltre ad tenere la CR, sarà colui che al termine di essa deciderà se approvarla o meno.

2.1.3 Come si svolge una code review?

Sulla base delle interviste effettuate da Pascarella et al. [3] sono state catalogate alcune situazioni che si verificano durante una code review.
Il primo passo consiste nel far comprendere al revisore il contesto in cui la modifica è stata effettuata. Successivamente il reviewer deve assicurarsi di aver compreso appieno il motivo ed il significato dei cambiamenti effettuate nella repository, per farlo utilizzerà il log scritto dall'autore oppure gli chiederà informazioni al momento.
Il reviewer potrebbe proporre all'autore una soluzione alternativa o chiarimenti sul motivo per cui è stata fatta una scelta implementativa rispetto ad un'altra.
È possibile che lo sviluppatore abbia introdotto all'interno del codice degli antipattern o magari abbia implementato male dei design pattern rendendo meno manutenibile il codice, andando controcorrente rispetto allo scopo della code review.
Altra importante fase è quella di ricerca di quanti più errori possibili in cui lo sviluppatore potrebbe essere incappato, un reviewer è spesso uno sviluppatore esperto e quindi è probabile che si accorga di tali problematiche.
Qualora il reviewer dovesse considerare le modifiche non atomiche, cioè riguardanti porzioni totalmente distaccate del software, è possibile che venga chiesto all'autore di suddividere il processo in più incontri.
Al termine della code review, se il revisore è soddisfatto dei cambiamenti, questi vengono accettati e introdotti nella repository principale, altrimenti, viene rifiutata e l'autore dovrà continuare a lavorarci per una futura code review.

2.2 Automatic Text Summarization e Summarization di codice sorgente

La Automatic Text Summarization è un processo in cui un testo viene condensato in una versione più breve che contiene le informazioni chiave e mantiene il significato del contenuto originale. Negli ultimi anni sono stati effettuti diversi studi per l'applicazione di tale tecnica su diversi domini, ma la nascita di tale processo non è di certo recente, infatti è dal 1950 che tale pratica ha ottenuto molta attenzione. [5]
Inoltre, la summarization non è da intendersi come pratica esclusiva per i testi, infatti

esistono tool che producono un summary anche di file multimediali con immagini o video. La possibilità di avere un modello che realizza dei summary che possono fare da documentazione per il codice sorgente o di un tool che velocizzi una code review riassumendo le modifiche fatte dall'autore sarebbe un vantaggio enorme in termini di risparmio temporale (quindi economico) nella fase di manutenzione di un prodotto software. Negli ultimi anni il problema della summarization è stato preso in esame dagli algoritmi di machine learning, in particolare l'area di studio interessata è il Natural Language Processing (NLP). [6]

2.3 Natural Language Processing

Il Natural Language Processing, o anche detto NLP, è una sottobranca dell'intelligenza artificiale e della linguistica che si occupa della comprensione dei linguaggi naturali (quelli parlati dagli esseri umani) da parte delle macchine. [7]

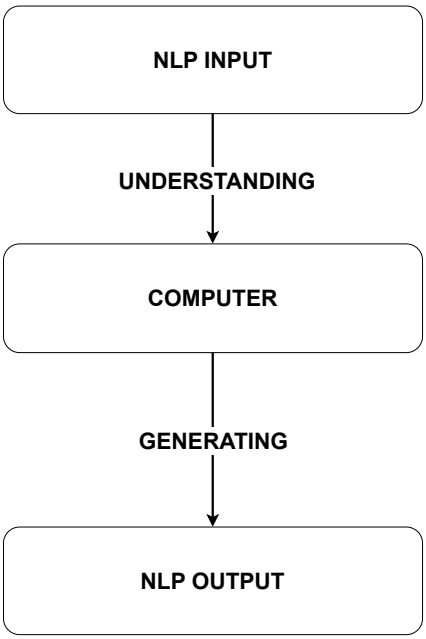

Figura 2.1: Natural Language Processing

L'NLP si occupa principalmente di testi, nel corso degli anni è stato sempre più importante avere un modo per comunicare con le macchina in maniera più naturale e il natural language processing risponde proprio a questa problematica.
Al giorno d'oggi i problemi di NLP si stanno sempre di più intrecciando con le tecni-

che di Machine Learning e Deep Learning, infatti è dal 2020 che si stanno avendo enormi passi in avanti nel mondo dell'interpretazione del linguaggio naturale, infatti, proprio nell'ultimo mese (Marzo 2023) OpenAI ha rilasciato il suo ultimo modello linguistico multimodale basato su reti neurali GPT-4 (Generative Pre-Trained Transformer 4) che secondo i dati rilasciati sembra essere la rete con più parametri che sia mai stata addestrata.
Alcune delle sfide più importanti che deve affrontare l'NLP riguardano la comprensione sintattica e semantica delle frasi del linguaggio naturale e del contesto a cui si fa riferimento, si parla quindi anche di Natural Language Understanding.
Sono diversi i task per cui i modelli di NLP vengono impiegati:

- Analisi dei testi, task che consiste nella estrazione di informazioni da testi.
- Classificazione dei testi, task per etichettare dei testi in base al loro contenuto.
- Text Generation
- Chat Bot, applicazioni che permettono l'interazione con gli umani mediante l'uso di linguaggio naturale.
- Assistenti vocali, applicazioni che permettono l'interazione con gli umani attraverso comandi vocali.
- Traduzioni
- Automatic Summarization, generazione di riassunti partendo da testi, video o codice sorgente.

Noi, chiaramente, ci soffermeremo in particolare su quelli che riguardano la summarization. Per farlo è necessario introdurre i concetti di Natural Language Understanding (NLU) e di Natural Language Generation (NLG). L'NLU è il processo di comprensione di un testo da parte di una macchina per quanto concerne la semantica e la sintassi.[8] L'NLG, invece, è il processo di produzione di testo in linguaggio naturale, il testo generato potrebbe variare da semplici risposte a delle domdande a complessi testi composti da diverse frasi. [8]

2.3.1 Text Summarization con NLP

L'NLP mette a disposizione più approcci per la realizzazione di summary [9]:

- Rules-based System, permette di realizzare un summary del codice sorgente usando delle regole "statiche" ben definite dalla persona che sta sviluppando il tool.
- Approccio Extractive, il quale ci permette di realizzare un riassunto usando degli algoritmi che estraggono le parole più importanti dal testo di partenza e della loro frequenza per metterle insieme e realizzare il sunto.
- Approccio Abstractive, che invece permette la realizzazione del summary in maniera più simile a come farebbe un umano, non vengono utilizzate frasi presenti all'interno del testo ma, una volta definiti i concetti chiave, viene costruito usando termini diversi.

2.3.2 Summarization di codice sorgente

Dallo studio effettuato nel 2010 da Buse et al. [10] abbiamo notato come la realizzazione di un algoritmo che generasse della documentazione per il codice sorgente in maniera automatica generava documentazione che nel 89% dei casi migliore o adatta a sostituire quella prodotta dagli sviluppatori, il dato risulta stupefacente se si pensa al fatto che 2/3 dei log associati alle modifiche non spiegano realmente i cambiamenti fatti sulla repository ma spesso si limita a messaggi brevi e poco precisi. Lo studio è stato fatto su un dataset di 250 messaggi di commit provenienti da 5 progetti diversi. L'algoritmo sviluppato è stato chiamato 'DeltaDoc'. L'algoritmo prende in input due codici sorgente di un programma e produce in output un summary dei cambiamenti che intercorrono tra le due versioni. Deltadoc analizza i due codici sorgente e per prima cosa va a determinare tutti i path (cicli o if/else statement) che sono presenti con le corrispettive azioni, confronta le eventuali differenze e produce un output simile:

If X, do Y instead of Z

L'output prodotto risultava essere troppo lungo (più lungo rispetto a quello scritto dagli sviluppatori), doveva essere sottoposto al processo di summarization, in sostanza le modifiche effettuate venivano riassunte il più possibile e il codice veniva reso quanto più semplice possibile. L'approccio usato da Buse et al. è stato rules-based, quindi sono state usati algoritmi ben definiti per la realizzazione della documentazione. Però, come citato in precedenza, esistono anche altri approcci di NLP per la summarization, che usano tecniche diverse.

2.3.3 Altre tecniche di Text Summarization

Esistono diverse tecniche di text summarization con l'NLP, alcune delle fondamentali sono:

- LSI
- VSM

Queste tecniche utilizzano degli schemi, chiamati "Weighting Schemas", che permettono di classificare l'importanza di una parola o di una frase all'interno di un testo. Tra i fondamentali troviamo log, tf-idf e binary entropy.

LSI, Latent Semantic Indexing

È una tecnica di elaborazione del linguaggio naturale che utilizza a sua volta tecniche di analisi matematica per cercare la correlazione semantica tra parole e termini di un testo. Nel dettaglio utilizza la SVD (Singular Value Decomposition), una tecnica che permette di identificare i concetti latenti, ovvero dei gruppi di parole e frasi che sono correlati tra di loro e che rappresentano i temi principali dei documenti in input. Con alcuni dei concetti latenti (i più importanti) verrà poi realizzato il summary.

VSM, Vector Space Model

Questa tecnica, a differenza di quella spiegata prima, utilizza un modello matematico che rappresenta i documenti come dei vettori in uno spazio multidimensionale. Sostanzialmente viene realizzata una matrice che determina il peso di ciascuna parola

nel documento, maggiore è il peso della parola e maggiore sarà l'impatto della stessa sulla stesura del summary.

2.3.4 Code Summarization con NLP e NLG

Diversi sono stati gli studi effettuati per la realizzazione di tool che offrissero la documentazione del codice sorgente mediante le tecniche precedentemente elencate. Ad esempio, lo studio di Haiduc et al. [9] mostra come con la combinazione di più tecniche si possa ottenere un summary molto accurato della code base, nel caso specifico sono stati presi in considerazione 2 progetti da più di 100k linee di codice. Le valutazioni sono state fatte da degli sviluppatori e i riscontri sono stati molto positivi. Arthur [1] nello studio condotto nel 2020 ha realizzato un tool per la documentazione dei programmi scritti in linguaggio C, il suo software genera due tipologie di documentazione, una si concentra su cosa realizza il codice di una funzione ad un livello di astrazione alto, mentre l'altra si occupa di documentare una funzione riga per riga. Il tool è composto di 3 parti fondamentali:

- Program Preprocessing, modulo che si occupa dell'analisi ed estrazione di informazioni dal codice sorgente.
- Software Word Usage Model (SWUM), modulo core del sistema che si occupa di applicare le regole di Context Free Grammars (CFG) alle righe del codice sorgente al fine di identificare il tipo di istruzione.
- Method Call Graph, che invece si occupa del mapping delle chiamate a funzione tra le funzioni, realizzando un vero e proprio grafo.

Il tool è stato provato con 20 software e i risultati sono stati buoni. Le valutazioni sono state fatte sia da developer esperti che usando ROUGE, un insieme di metriche usato per valutare automatic summarization e traduzioni effettuate da macchine. [11]

2.3.5 Uso di un modello di GPT-3 per Code Summarization

Codex è un modello pre addestrato di GPT-3 pensato per lavorare con il linguaggio naturale e con i linguaggi di programmazione, allenato su miliardi di linee di

codice pubblicate su GitHub. [12] Nello studio effettuato da Khan et al. [12] viene utilizzato prioprio questo modello per la realizzazione di summary di codice sorgente. L'esperimento condotto è stato sviluppato prendendo in considerazione progetti scritti in molteplici linguaggi di programmazione e per ognuno è stato calcolato lo score mediante lo score che usa la scala BLEU. Tale scala permette di misurare automaticametne la qualità di un summary comparandolo al summary generato dagli umani. [13]
L'interazione con il modello è stata fatta mediante prompt, al quale veniva chiesta il summary di un dato codice sorgente. Per ciascun input sono state fatte due prove, zero-shot learning e one-shot learning, che consistono nel dare o meno al modello la label corrispondente all'input dato in pasto.
I risultati nel primo caso non sono stati soddisfacenti, ma nel secondo caso il tool ha raggiunto punteggi molto alti, anche superiori a quelli di modelli simili come CoTexT.

2.4 Code Review Summarization

Al momento, in letturatura, uno degli studi disponibili che riguarda il processo di resoconto delle informazioni fondamentali per lo svolgimento di una code review è quello proposto da Song et al.[4]. Song et al. hanno proposto un sistema che permetta in primo luogo di identificare in una code base i cambiamenti e di classificarli in base alla loro importanza. Non solo, il tool vuole anche velocizzare il processo di review e permettere attraverso l'IDE al reviewer di lasciare un feedback. Lo studio proposto ha avuto i risultati sperati, la valutazione è stata fatta su progetti open source di terze parti e il tool riesce a identificare i cambiamenti importanti e aumentare la produttività del reviewer.

2.5 Research Gap

In letteratura al momento non sono presenti studi che hanno portato alla realizzazione di tool che realizzi un summary delle modifiche fatte alla code base avvalendosi di modelli linguistici già predisposti alla comprensione di codice sorgente come ad

esempio GPT. La realizzazione di un tool che permetta al reviewer di conoscere le modifiche effettuate dall'autore andrebbe a ridurre i tempi necessari per le code review, in quanto non sarà più necessario che l'autore esponga il contesto che sta andando a presentare ed eliminerebbe le eventuali incomprensioni che potrebbero crearsi tra reviewer e autore.

CAPITOLO 3

CoReviewer

3.1 Obiettivo

Dopo un'attenta analisi dello stato dell'arte relativo alle attività di revisione del codice e alle tecniche di NLP per la summarization del codice sorgente, è emerso il vuoto nella ricerca è rappresentato dalla mancanza di uno strumento che possa assistere i revisori durante il processo di code review.
L'obiettivo principale di questa tesi consiste nel sviluppare una soluzione che consenta ai reviewer di ottenere un riassunto sintetico delle modifiche apportate alla codebase. L'implementazione di un tale strumento potrebbe contribuire in modo significativo a ridurre le possibili ambiguità che possono emergere tra l'autore del commit e il revisore durante l'analisi delle modifiche ed inoltre semplificherebbe il processo di revisione, riducendo le tempistiche e, di conseguenza, i costi associati.
L'idea è stata di realizzare un tool che, dopo aver ricevuto un link della repository di cui si sta per iniziare la code review, analizzerà le differenza della codebase tra l'ultimo commit ed il precedente, tali differenze saranno passate ad modello linguistico capace di interpretarle e sintetizzarle e sarà in grado di restituirci un breve summary da far leggere al reviewer. Il nome del tool che è stato realizzato è CoReviewer. Questo nome indica come il tool aiuti, proprio come se fosse un co-pilota, il reviewer nello

svolgere il processo di revisione del codice sorgente mostrandogli il summary delle modifiche effettuate alla code base.

3.2 Scelte di Design

3.2.1 Scelta del tipo di applicativo

Una delle sfide iniziali affrontata è stata quella di determinare il tipo di applicazione più idoneo per realizzare il nostro obiettivo. Durante questa fase di analisi, ci siamo concentrati sull'osservazione del modo in cui le moderne code review si svolgono prevalentemente attraverso strumenti web dedicati al versioning del codice. Questa osservazione ha portato alla conclusione che lo sviluppo di uno strumento complementare all'interfaccia web, ad esempio integrabile con piattaforme come GitHub, sarebbe stato ottimo per il raggiungimento del nostro progetto.
L'idea era anche quella di garantire che il nostro strumento fosse accessibile al maggior numero possibile di persone, indipendentemente dal sistema operativo o dall'architettura del loro computer. Alla luce di queste considerazione, abbiamo preso la decisione di sviluppare un'estensione, un plugin per i web browser che permette di aggiungere funzionalità aggiuntive a tutti coloro che avessero installato l'estensione sul loro browser, rendendo il nostro strumento accessibile e versatile per una vasta gamma di utenti.

3.2.2 Scelta del modello linguistico

Nel 2023 il mondo dei modelli linguistici è davvero molto vasto, esistono, infatti, molti modelli linguistici che avrebbero fatto al caso nostro per la realizzazione dell'obiettivo. Dopo un'accurata ricerca abbiamo notato come diversi modelli fossero utilizzabili solo a fronte di un canone mensile (come ad esempio GPT-4), altri invece erano poco addestrati al riconoscimento e interpretazione del codice sorgente, quindi non sarebbero stati utili alla nostra causa.
La scelta finale è ricaduta sul modello linguistico LLaMA 2, un modello realizzato da Meta e messo a disposizione in maniera completamente gratuita per scopi di ricerca e commerciali. Di LLaMA 2 ne esistono diverse versioni, ognuna con un numero di

parametri di allenamento.
La scelta per la realizzazione dell'estensione è ricaduta sulla versione da 7 miliardi di parametri, in quanto offriva un buon compromesso tra velocità di risposta, spazio necessario per l'hosting e accuratezza delle risposte. Inoltre la scelta di un modello con un maggior numero di parametri avrebbe necessitato di un'altissima potenza di calcolo per la propria esecuzione. In particolare la versione che abbiamo deciso di scaricare è stata allenata proprio alla comprensione e alla generazione di codice sorgente, pertanto l'uso di un modello preaddestrato si è rivelato un vantaggio in termini pratici. I vantaggi nell'uso di un modello open source sono riscontrabili anche nella possibilità di modificare il modello adattandolo a specifici compiti di apprendimento, parleremo di tale possibilità nel capitolo dedicato agli sviluppi futuri.
L'unico svantaggio riscontrato nella scelta del modello offerto da Meta è nell'installazione dello stesso sulla macchina che potrebbe risultare difficoltosa, infatti il modello rilasciato non è perfettamente compatibile con alcune architetture di processori (ARM ad esempio), ma in generale esistono e sono state riportate nel capitolo dedicato le guide per la corretta installazione su ogni macchina.

3.2.3 Architettura

L'architettura selezionata per la realizzazione dell'estensione è di tipo client-server. Questo significa che il sistema è diviso in due componenti principali:

- Client, rappresentato dal web browser che ospita l'estensione, svolge il ruolo di interfaccia utente e invia richieste al server.
- Server, un sistema remoto (che nel nostro caso è stato simulato in locale sulla macchina) in grado di elaborare le richieste del client e restituire le risposte al Front-End.

Nel nostro caso, il server ospita il modello linguistico LLaMA 2, il quale è responsabile di generare il summary richiesto. Quando l'estensione invia una richiesta al server, questo elabora la richiesta utilizzando il modello LLaMA 2 e restituisce il summary risultante, che verrà quindi visualizzato nell'interfaccia dell'estensione direttamente nel browser del cliente. La scelta di questa architettura è stata fatta

per suddividere in maniera efficace i compiti tra client e server e per non obbligare l'utente ad installare sulla propria macchina LLaMA 2. La realizzazione di un web server, invece, è dovuta al fatto che il modello linguistico LLaMA 2 doveva essere hostato per essere richiamato dall'estensione e generare il summary richiesto. L'uso del web server, inoltre, rende scalabile l'applicazione e capace gestire un alto volume di richieste da parte dei client e distribuire efficacemente le risposte.
Discuteremo nelle sezioni successive delle tecnologie utilizzate per lo sviluppo di estensione e del web server e delle motivazioni che hanno portato a tali scelte e dell'installazione di LLaMA 2 su una macchina. Di seguito viene riportato il diagramma architetturale del sistema:

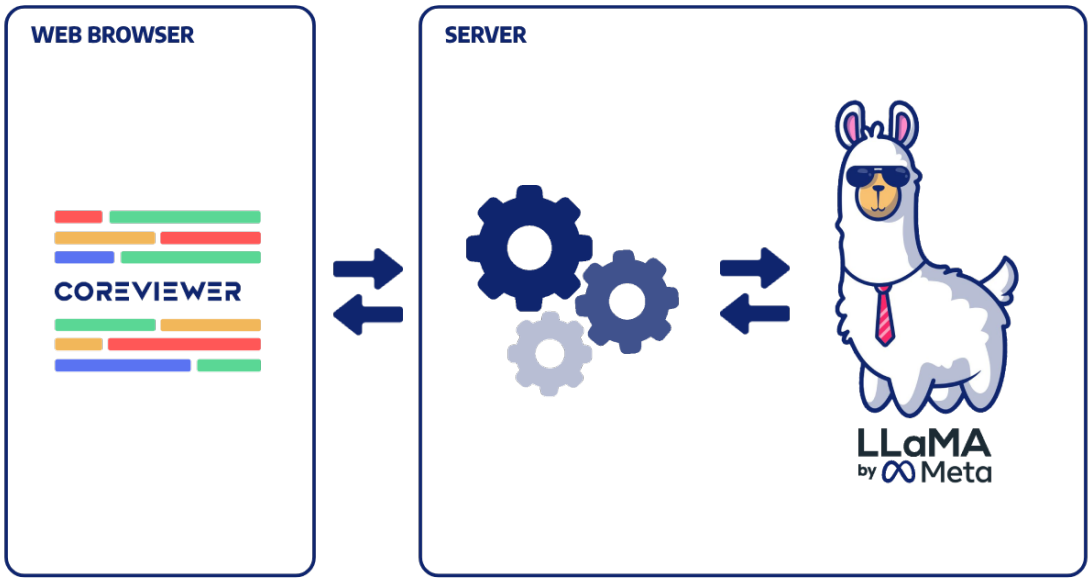

Figura 3.1: Diagramma Architetturale

3.3 Sviluppo dell'estensione

Lo sviluppo dell'estensione si è diviso in 3 fondamentali passi:

- Installazione del modello LLaMA 2 sulla macchina
- Realizzazione del Front-End dell'estensione
- Realizzazione del Back-End dell'estensione e chiamate al modello LLaMA 2

Nelle sezione seguenti saranno affrontati ognuno dei 3 temi spiegando nel dettaglio le varie scelte implementative che sono state prese per realizzare il prodotto finale.

3.3.1 Installazione di LLaMA 2

L'installazione del modello linguistico è effettuabile in maniera gratuita collegandosi al sito web di LLaMA [1], per poter effettuare il download è necessario creare un account developer Meta. In automatico dopo aver richiesto il download si riceverà un email da Meta con i link ai download delle varie versioni di LLaMA 2. Le versioni disponibili differiscono per il numero di parametri di allenamento, per la realizzazione dell'estensione si è optato per la versione a 7 miliardi di parametri, ma sono disponibili versioni con 13, 30, 70 miliardi di parametri di allenamento. Di ciascuna versione era presente anche la variante "Code", cioè allenata prettamente alla generazione e comprensione di codice sorgente.
Dopo aver installato il modello, si procederà con i passi per l'installazione indicati nella repository ufficiale di Meta [2] oppure, nel caso si possegga una macchina con processore ARM è necessario seguire una guida alternativa consultabile in una apposita repository [3]. Per l'installazione del modello sarà necessario avere installato sulla propria macchina l'ultima versione di Python e saranno richiesti diversi GB di spazio libero su disco a seconda della versione che si sceglierà di installare.
Una volta completata la procedura di installazione sarà possibile avviare una conversazione con il modello tramite il terminale offerto dal proprio sistema operativo.

[1] `https://ai.meta.com/llama/`
[2] `https://github.com/facebookresearch/llama`
[3] `https://github.com/ggerganov/llama.cpp`

Sarà compito del web server di estrapolare dalla risposta del modello solo il summary da mostrare all'utente finale.

3.3.2 Front-End

L'estensione è stata sviluppata utilizzando TypeScript e la libreria React JS, una scelta guidata da molteplici considerazioni, in particolarela loro capacità di notevolmente agevolare il processo di sviluppo di un'estensione per browser.
La scelta di preferire TypeScript a JavaScript è stata fatta in quanto la tipizzazione evita errori comuni, garantisce una migliore comprensione del flusso di dati all'interno dell'applicazione e la leggibilità del codice. Ciò si traduce in un codice più affidabile e manutenibile nel tempo.
React JS, invece, è una libreria che rende la realizzazione delle interfacce grafiche semplice e manutenibile. Il codice che si scrive è anche riutilizzabile proprio per caratteristica della libreria che favorisce lo sviluppo di componenti e l'uso di componenti ready to use scaricabili online facilmente. React JS ci permette di inviare richieste al server in maniera asincrona, in modo da non dover bloccare il flusso di esecuzione del codice durante la fase di generazione della risposta da parte del server.
Ciascuna parte dell'interfaccia grafica è stata realizzata con il linguaggio di markup HTML5, mentre gli stili applicati ad ogni elemento del design sono stati realizzati con CSS3. L'interfaccia utente dell'applicazione è stata progettata seguendo le fondamentali "8 regole d'oro dell'interazione Uomo-Macchina" di Ben Shneiderman, con una particolare enfasi sulla regola che sottolinea l'importanza di mantenere lo stato del sistema sempre visibile. L'utente, anche nei momenti di attesa del summary, è consapevole che la generazione sta avvenendo in quanto, una volta dato in input il link e premuto il bottone, sarà mostrato un avviso che lo informa. Tale avviso scomparirà per essere sostituito dal summary una volta ottenuta la risposta dal server.
Inoltre, per garantire la massima chiarezza ed usabilità è stata adottata una filosofia di design minimalista durante il processo di costruzione dell'interfaccia, rifacendoci allo stile estetico usato per l'interfaccia di GitHub. Non solo, l'interfaccia è volutamente semplice in quanto si vuole dare al reviewer la possibilità di ottenere il summary in pochi semplici click e velocizzare il processo il più possibile evitando elementi

superflui sulla schermata.
La UI dell'estensione è composta da elementi essenziali che rendono l'uso semplice ed efficiente. Gli elementi chiave includono un input dedicato all'inserimento del link alla repository oggetto della code review e un pulsante che consente di inoltrare la richiesta di generazione del summary al Back-End. Una volta ricevuta la risposta dal server, il summary generato verrà visualizzato immediatamente in una textarea. La figura 3.2 mostra un mockup disegnato con il software Figma di come si presenterà il sistema una volta sviluppata la sua interfaccia grafica.

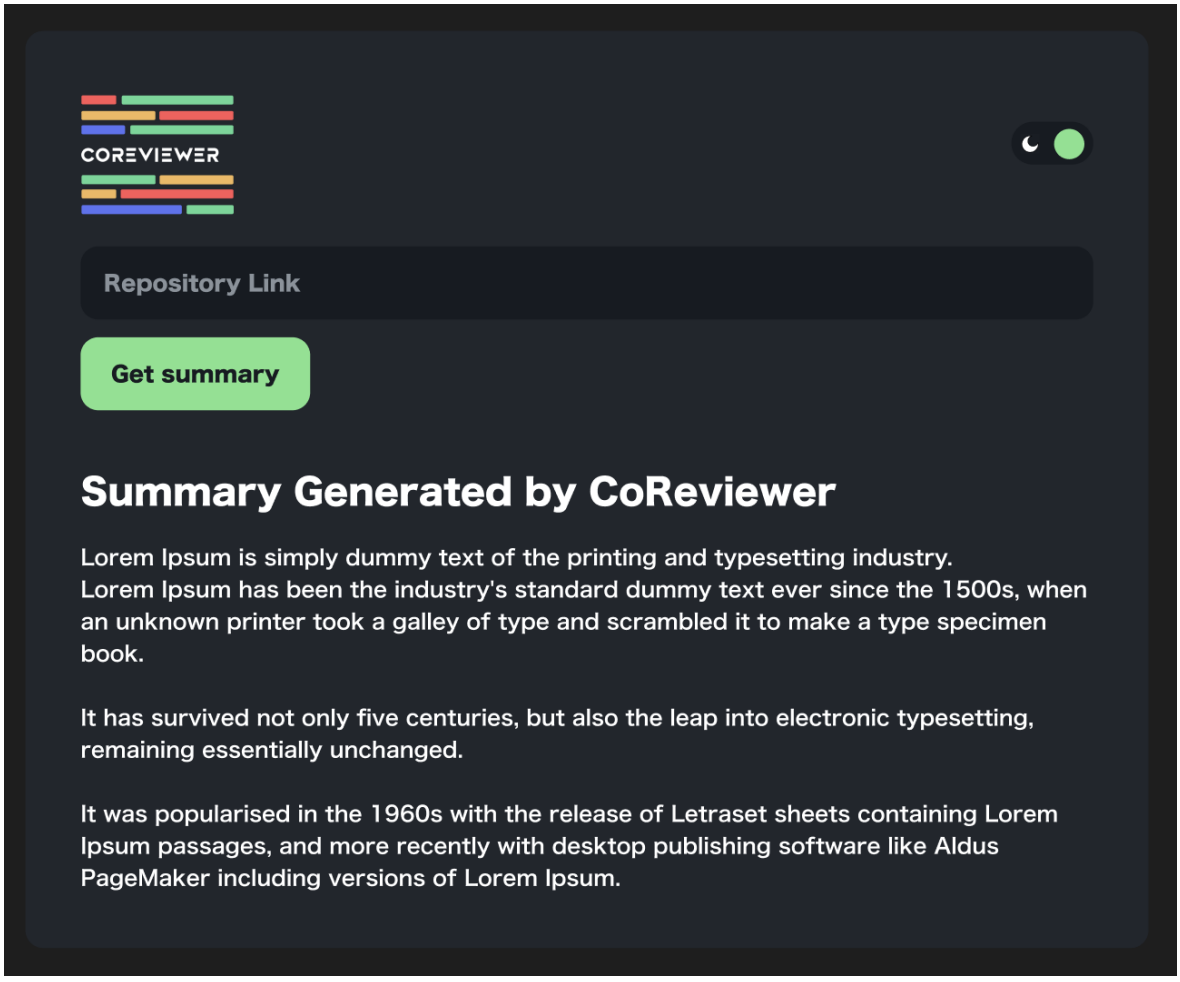

Figura 3.2: Interfaccia Utente dell'estensione

3.3.3 Back-End

Il back-end dell'estensione è implementato utilizzando un web server scritto in Python, nello specifico utilizzando il framework Flask. Questa scelta è stata guidata

da diverse motivazioni chiave:

- Accesso alle librerie Python: Uno dei vantaggi fondamentali nell'utilizzo di Python come linguaggio di sviluppo per il back-end è la vasta disponibilità di librerie e moduli. In particolare, era necessario lavorare con una libreria specifica per l'accesso alle repository, e Python offriva una soluzione pronta all'uso, facilitando così l'integrazione di questa funzionalità nel nostro server.

- Velocità di sviluppo: la scelta di Python come linguaggio e di Flask come framework web ha permesso di sviluppare rapidamente il server, grazie alla sua struttura leggera e alla facilità di implementazione delle route.

- Scalabilità: Flask è altamente scalabile e può gestire applicazioni di grandi dimensioni quando necessario. Ciò ci dà la flessibilità di espandere e migliorare il nostro sistema in futuro.

Il server sviluppato offre due funzioni fondamentali:

- Generare l'input per il modello LLaMA 2

- Invocazione del modello LLaMA 2 per ottenere il summary

- Restituire il summary al Front-end

Nello specifico la generazione dell'input per LLaMA 2 viene fatta mediante uno script in Python, quest'ultimo utilizza la libreria GitPython che permette di interagire con le repository di GitHub.
La libreria, tra le tante funzioni che prevede, consente di analizzare una repository pubblica o privata dando in input il link alla pagina GitHub della stessa. Lo script si occupa di clonare la repository in una cartella temporanea sul server e analizzerà le differenze della codebase che ci sono tra la sua versione corrente e quella precedente all'ultimo commit. In tale modo otteniamo un piccolo report di modifiche (righe aggiunte, righe rimosse, id dell'ultimo e del penultimo commit e file modificati) che successivamente passeremo al modello LLaMa 2 con un appropriato prompt per farci scrivere un summary. L'invocazione del modello è il secondo compito di cui il web server si occuperà; dopo aver installato sul server il modello è sufficiente,

mediante la libreria "os" di Python posizionarci nella cartella sul server su cui è installato il modello e impartire il comando con i rispettivi parametri per avviare il modello e avviare la conversazione col modello che leggerà il prompt della richiesta e provvederà a generare il summary richiesto. L'ultimo compito del web server è quello di restituire al Front-End il summary, questo viene fatto restituendo un oggetto JSON, tramite la funzione "jsonify". La decisione di usare JSON è stata presa poiché è un modo leggero di inviare e ricevere dati e per la facile integrazione che si ha con le tecnologie usate per il FrontEnd.
Di seguito viene riportato il comando usato per generare il summary con il modello LLaMA 2, da notare l'uso della variabile "promptPerLLama" la cui importanza verrà discussa nel capitolo successivo, questa in sostanza contiene la richiesta del summary seguita dalle modifiche generate dallo script sopra descritto:

```
"./main",
"-m", "./models/7B/ggml-model-q4_0.bin",
"-t", "8",
"-n", "512",
"-p", promptPerLLama
```

3.3.4 Operazionalizzazione

Per il testing dell'estensione è stato usato il browser Google Chrome. Tramite il pannello di controllo di gestione delle estensioni è stato possibile caricare la nostra estensione appena realizzata. Poiché l'estensione si collega ad un web server, quest'ultimo dovrà essere hostato o, come nel nostro caso, in esecuzione sulla nostra macchina. Per mandare in esecuzione il web server basterà eseguire il comando "python3 webServer.py" dopo aver scaricato la repository. A questo punto è possibile utilizzare l'estensione realizzata accedendovi dalla barra delle estensioni del browser. Una volta inserito il link alla repository di cui effettuare la code review dopo pochi secondi sarà disponibile al reviewer il summary delle modifiche.
Dopo aver avviato la richiesta se il server lo richiede sarà necessario inserire il proprio username e la propria password dell'account Github, necessario per lavorare con le API di GitHub.

Un esempio di summary generato è il seguente:

-1 +1,3 @@

-console.log("Ciao Mondo!!");

+for(let i=0; i < 10; i++) + console.log("ciao mondo"); +

Added a 'for' loop that logs "ciao mondo" 10 times. Removed log "Ciao Mondo!!"

CAPITOLO 4

Validazione e Discussione

4.1 Testing dell'estensione

Al termine dell'implementazione è iniziata una delle fasi più importanti del ciclo di vita di un prodotto software, il testing. Testare l'estensione nel nostro caso è significato in particolare testare il comportamento del modello su prompt diversi e verificare che le risposte fossero in linea con quanto effettivamente ci si aspettasse. Questa fase di testing ci ha portato diversi spunti di riflessione, in particolare per quanto riguarda alcune problematiche riscontrate con il modello e in generale con la qualità delle risposte che questo ci restituiva. Nelle sezioni seguenti discuteremo dell'importanza della fase di prompt engineering necessaria per il dialogo con il modello e riporteremo come il modello, quando gli venivano somministrati dei prompt non molto chiari, non sempre è stato in grado di fornirci delle risposte adeguate, talvolta addirittura completamente sbagliando la risposta, fornendo summary o dei testi per niente attinenti al contesto a cui ci stavamo riferendo.
Successivamente si parlerà di eventuali "outlier" nelle risposte ottenute, di come siamo arrivati alla scelta del prompt più adatto alla nostra esigenza e discuteremo delle eventuali cause che potrebbero aver portato a comportamenti indesiderati.

4.2 Scelta del prompt adatto

Quando si lavora con modelli linguistici come GPT o LLaMA 2 è importante chiarire al meglio le nostre intenzioni, il modello interpreta il nostro prompt e qualora questo non fosse compreso a pieno le risposte che potremmo ottenere si distaccherebbero da quello che era il obiettivo iniziale.
La generazione di summary di codice sorgente è una tra le tante mansioni in cui modello LLaMA 2 eccelle, l'aspettativa era quindi che il modello producesse un output chiaro e conforme alle nostre aspettative. Ciononostante di fondamentale importanza è stata la fase di studio e ricerca del prompt più adatto alle esigenze del nostro prodotto. Durante la fase di testing dell'estensione abbiamo tentato di generare summary cambiando il prompt e facendo diverse prove su delle repository i cui ultimi commit avevano apportato delle modifiche molto semplici alla codebase. L'obiettivo di questa fase era trovare il prompt che più facesse avvicinare il modello alla risposta "perfetta".

Forma del prompt

Prima di parlare dei vari prompt che abbiamo provato è necessario spiegare brevemente la sua forma; come abbiamo visto nel capitolo precedente il prompt che è stato somministrato era il composto di due parti fondamentali: la richiesta di generazione di summary delle modifiche + il log delle modifiche generato mediante la libreria GitPython. Nella figura 4.1 viene riportato un esempio di log generato dalla libreria nel constesto di una repository il cui ultimo commit aveva apportato la seguente modifica: il codice sorgente alla versione 1.0 era composto di solo una console.log che stampava il testo "Ciao Mondo!!", mentre nella versione 1.1 il listato prevede la cancellazione di quella stampa e l'inserimento di un for loop che stampava per 10 volte sulla console la frase "ciao mondo". Come notiamo oltre all'elenco delle righe rimosse e di quelle aggiunte vengono riportati anche i dati che riguardano il numero delle righe che sono state aggiunte e rimosse, gli id dei commit che sono stati presi in considerazione e i nomi dei file che sono stati modificati dall'ultimo commit.

```
(base) alessz@MacBook-Air-di-Alessandro PythonCodeReviewScript % python3 getCommitData.py
diff --git a/index.js b/index.js
index d95bdd9..e68875e 100644
--- a/index.js
+++ b/index.js
@@ -1 +1,3 @@
-console.log("Ciao Mondo!!");
+for(let i=0; i < 10; i++){
+    console.log("ciao mondo");
+}
\ No newline at end of file
(base) alessz@MacBook-Air-di-Alessandro PythonCodeReviewScript %
```

Figura 4.1: Dati del commit restituiti da GitPython

Test effettuati sul modello

Il primo approccio con il modello è stato fatto facendo delle richieste di summarization in lingua Italiana, ciò ha portato a risposte di LLaMA 2 che non rispettavano le nostre aspettative, infatti nella maggior parte dei casi il modello rispondeva in Inglese nonostante la richiesta di realizzare un summary in una lingua diversa. Questo ci ha precluso l'opportunità di permettere al reviewer di scegliere la lingua in cui ottenere il summary nel caso in cui l'estensione venga utilizzata da un reviewer che non ha conoscenze della lingua Inglese.
Dopo aver notato questo limite di LLaMA 2 a 7 miliardi di parametri abbiamo deciso di testare il modello di LLaMA 2 a 70 miliardi di parametri, utilizzabile gratuitamente al link "https://www.llama2.ai/", ciò ci ha permesso di realizzare che il problema di lingua era un limite della versione con meno parametri di allenamento e che la comprensione e capacità di generazione di risposte di quello potenziato era decisamente migliore. Ciononostante, anche nel modello più allenato, per ottenere delle risposte in lingue diverse dall'Inglese non è sufficiente scrivere il prompt in lingua ma sarà necessario specificare che la risposta la si vuole nella lingua desiderata, comportamento che nella prova con il modello GPT non è stato riscontrato.
Ad ogni modo i risultati del test con i prompt in lingua Italiana non sono stati molto soddisfacenti, sia per i problemi legati alle risposte in lingua diversa ma anche per quanto riguarda i contenuti delle risposte che venivano prodotti. Spesso il focus della risposta si discostava completamente dal summary, generando risposte che erano affini al commit ma non restituivano un summary delle modifiche, bensì una descrizione dei dati del summary, soffermandosi in particolare sull'ID dei commit e

sui file che sono stati modificati, ma tali problematiche saranno discusse nel dettaglio nelle sezioni successive.
Siamo passati quindi alla fase di testing dei prompt in lingua Inglese, della quale possiamo ritenerci soddisfatti. Nella maggior parte dei casi, infatti, le risposte generate da LLaMA 2 erano conformi a quelle che erano le aspettative e quasi sempre sono state rispettate le richieste di generare un summary breve e chiaro di quelle che sono state le modifiche apportate alla codebase dall'ultimo commit. La motivazione per cui le risposte del modello sono state migliori quando l'input era effettuato in lingua Inglese sta nel fatto che la maggior parte dei dati di addestramento per modelli come LLaMA 2 sono in Inglese e ciò consente al modello di comprendere e interpretare meglio tale lingua. Inoltre, visto che nel log generato da GitPython sono presenti anche etichette in lingua Inglese, il modello potrebbe aver interpretato di dover generare la risposta proprio in Inglese piuttosto che in altre lingue.

In seguito, contrariamente a quanto fatto per il test dei prompt in Italiano, le cui varie prove utilizzando input diversi sono state fatte al fine di valutare la capacità del modello nell'interpretare correttamente una richiesta, il testing dei prompt in inglese aveva l'obiettivo di individuare un prompt che consentisse di ottenere il maggior numero di risposte conformi alle aspettative desiderate.

Seguono nella tabella 4.1 tutta la serie di input sia in Italiano che in Inglese di cui sono state effettuate le prove sul modello:

Prompt in lingua Italiana	Prompt in lingua Inglese
Realizza un breve summary delle modifiche avvenute in questi commit + commitData	Tell me what happened in these commits + commitData
Che cos'è accaduto in questi commit + commitData	Describe the changes in these commits + commitData
Genera un riassunto delle modifiche effettuate in questi commit + commitData	Generate a summary of these commits + commitData
Spiegami cos'è avvenuto in questi commit + commitData	Summarize the changes of these commits + commitData
Effettua una code review di questa repository + commitData	Help me as a co reviewer to understand the changes in those commits + commitData
	Tell me more about this commit + commitData

Tabella 4.1: Input dati al modello in italiano con relativa risposta

Le risposte ottenute sono state nella maggioranza dei casi migliori quando l'input era posto in lingua Inglese, il prompt che ha ottenuto la percentuale maggiore di risposte "corrette" è "tell me what happened in this commit + commitData", pertanto la decisione è stata quella di scegliere questo nella soluzione finale proposta. I summary generati dai prompt in lingua Inglese sono risultati in linea con le modifiche effettuate alla codebase e la descrizione delle modifiche fatte corrispondeva con quanto documentato dal modello.

Unusual Behaviors e Outlier

Durante la fase di testing, come intuibile dalla sezione precedente, non sempre il modello ha restituito la risposta che ci aspettavamo, pertanto in questa sezione parleremo dei comportamenti inaspettati che ha avuto il modello durante la nostra

interazione con esso.
Durante le varie prove effettuate sul modello, nella totalità delle circostanze il modello ha generato risposte che presentavano all'interno la replica della domanda posta in input e in alcune occasioni questa non è stata accompagnata dal summary richiesto. Il comportamento appena descritto mostra come il modello non sempre risponde in maniera coerente con quelle che sono le richieste. Alcuni input, in particolare "Tell me more about this commit + commitData" portavano il modello a generare una descrizione della liberia GitPython e in particolare della funzione "diff", cioè quella che permette di generare la differenza tra le due versioni della codebase, di cui abbiamo parlato nel capitolo dedicato alla metodologia.
Infine, come ulteriore situazione inusuale, abbiamo notato come LLaMA 2 provasse ad avere una conversazione con il reviewer, in particolare abbiamo osservato i seguenti comportamenti:

- Il modello ha posto delle domande al reviewer invece di generare il summary richiesto con lo scopo di ottenere delle informazioni supplementari.
- Il modello dopo aver generato il summary ha provato a dare delle soluzioni alternative rispetto a quelle che sono state implementate dallo sviluppatore nell'ultimo commit.

Tali comportamenti, in particolare il secondo, potrebbero essere dei potenziali aggiornamenti dell'estensione che verranno approfonditi maggiormente nel capitolo dedicato agli sviluppi futuri e alle conclusioni.

4.3 Limitazioni e possibili cause

Come abbiamo potuto notare, nonostante le buone risposte ottenute dal modello, questo non è esente da problemi, come ad esempio generare degli output non in linea con il focus del prompt. Le motivazioni di queste limitazioni potrebbero risiedere nella scelta iniziale del modello, probabilmente scegliendo il modello da 70 miliardi di parametri avremmo ottenuto delle risposte accurate nella totalità delle casistiche. La scelta di preferire quello a 7 miliardi di parametri è stato un tradeoff tra velocità di risposta e prestazioni dell'applicazione. Scegliere il modello più allenato richiedeva

maggiore potenza di calcolo e quindi maggiori tempi di risposta rispetto al modello scelto. Altra possibile causa di tali problemi è il fatto che non è stata fatta una fase di fine-tuning del modello mediante l'utilizzo di un dataset adatto alle nostre esigenze. La scelta di non optare per tale opzione è stata fatta in quanto il modello scelto è stato preaddestrato alla comprensione e generazione di codice sorgente, pertanto non è stato ritenuto necessario operare su questo fronte. Infine, poiché il formato in cui la libreria GitPython ha messo in difficoltà il modello in alcune occasioni, una possibile causa dell'inaccuratezza delle risposte potrebbe essere risolta andando a modificare l'input passato al modello in un formato facilmente comprensibile ad esso.

CAPITOLO 5

Conclusioni

Lo studio ha avuto come obiettivo la realizzazione di un tool che facesse da supporto al reviewer durante l'attività di code review, fornendogli un summary delle modifiche a cui è stata sottoposta la codebase dopo la realizzazione di un commit da parte dello sviluppatore.
Il tool che è stato realizzato è un'estensione per web browser, CoReviewer. L'estensione è stata realizzata utilizzando il linguaggio di programmazione TypeScript e la libreria React JS per il frontend. Il backend, invece, è stato realizzato in Python, con il framework Flask.
Il cuore del progetto, ovvero l'artefice della realizzazione del summary è il modello linguistico LLaMA 2 di Meta. Tale modello viene interrogato e, una volta che gli sono stati passati i dati necessari, genererà un breve riassunto delle modifiche fatte alla repository nell'ultimo commit. Al termine dell'implementazione sono stati effettuati i relativi test che hanno portato un esito soddisfacente, ciononostante abbiamo riscontrato alcuni problemi che hanno portato a doverose considerazioni e strategie che potrebbero migliorare l'estensione e ampliare le funzionalità che offre. In particolare gli sviluppi futuri a cui si fa riferimento riguardano la necessità di aggiornare il modello LLaMA 2 a 7 miliardi di parametri con uno più allenato, inoltre si potrebbe pensare di utilizzare un dataset ad hoc per una eventuale fase di fine-tuning e adde-

strare il modello proprio per il nostro scopo.
Un'altra evoluzione a cui sottoporre l'estensione è quella di offrire al reviewer un set di funzionalità più ampio, ad esempio aggiungendo la possibilità di rilevare eventuali anti-pattern all'interno del codice e di effettuarne il refactoring. Il modello risulta avere un potenziale molto grande che nell'ambito di questo progetto è stato utilizzato solamente in minima parte, con i giusti accorgimenti CoReviewer potrebbe offrire al reviewer possibilità ben maggiori e velocizzare i processi aziendali di Code Review in maniera importante.

Bibliografia

[1] M. P. Arthur, "Automatic source code documentation using code summarization technique of nlp," *Procedia Computer Science*, vol. 171, pp. 2522–2531, 2020. (Citato alle pagine 4 e 11)

[2] A. Lakhotia, "Understanding someone else's code: Analysis of experiences." *J. Syst. Softw.*, vol. 23, no. 3, pp. 269–275, 1993. (Citato a pagina 4)

[3] L. Pascarella, D. Spadini, F. Palomba, M. Bruntink, and A. Bacchelli, "Information needs in contemporary code review," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–27, 2018. (Citato alle pagine 5 e 6)

[4] M. Song and Y.-W. Kwon, "Which code changes should you review first?: A code review tool to summarize and prioritize important software changes," *Journal of Multimedia Information System*, vol. 4, no. 4, pp. 255–262, 2017. (Citato alle pagine 5 e 12)

[5] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut, "Text summarization techniques: a brief survey," *arXiv preprint arXiv:1707.02268*, 2017. (Citato a pagina 6)

[6] "Natural language processing," https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP. (Citato a pagina 7)

[7] A. Chopra, A. Prashar, and C. Sain, "Natural language processing," *International journal of technology enhancements and emerging engineering research*, vol. 1, no. 4, pp. 131–134, 2013. (Citato a pagina 7)

[8] C. Dong, Y. Li, H. Gong, M. Chen, J. Li, Y. Shen, and M. Yang, "A survey of natural language generation," *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–38, 2022. (Citato a pagina 8)

[9] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *2010 17th Working Conference on Reverse Engineering*. IEEE, 2010, pp. 35–44. (Citato alle pagine 9 e 11)

[10] R. P. Buse and W. R. Weimer, "Automatically documenting program changes," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 33–42. (Citato a pagina 9)

[11] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81. (Citato a pagina 11)

[12] J. Y. Khan and G. Uddin, "Automatic code documentation generation using gpt-3," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–6. (Citato a pagina 12)

[13] Y. Graham, "Re-evaluating automatic summarization with bleu and 192 shades of rouge," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 128–137. (Citato a pagina 12)